Cross-Problem Learning for Solving Vehicle Routing Problems
Zhuoyi Lin\textsuperscript{1}, Yaoxin Wu\textsuperscript{2*}, Bangjian Zhou\textsuperscript{3}, Zhiguang Cao\textsuperscript{4}, Wen Song\textsuperscript{5}, Yingqian Zhang\textsuperscript{2} and Senthilnath Jayavelu\textsuperscript{1}
\textsuperscript{1}Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR)
\textsuperscript{2}Eindhoven University of Technology
\textsuperscript{3}Unidt
\textsuperscript{4}Singapore Management University
\textsuperscript{5}Shandong University
zhuoyi.lin@outlook.com, wyxacc@hotmail.com, Bangjian.zhou@unidt.com, zhiguangcao@outlook.com, wensong@email.sdu.edu.cn, yqzhang@tue.nl, J_Senthilnath@i2r.a-star.edu.sg

Abstract
Existing neural heuristics often train a deep architecture from scratch for each specific vehicle routing problem (VRP), ignoring the transferable knowledge across different VRP variants. This paper proposes the cross-problem learning to assist heuristics training for different downstream VRP variants. Particularly, we modularize neural architectures for complex VRPs into 1) the backbone Transformer for tackling the travelling salesman problem (TSP), and 2) the additional lightweight modules for processing problem-specific features in complex VRPs. Accordingly, we propose to pre-train the backbone Transformer for TSP, and then apply it in the process of fine-tuning the Transformer models for each target VRP variant. On the one hand, we fully fine-tune the trained backbone Transformer and problem-specific modules simultaneously. On the other hand, we only fine-tune small adapter networks along with the modules, keeping the backbone Transformer still. Extensive experiments on typical VRPs substantiate that 1) the full fine-tuning achieves significantly better performance than the one trained from scratch, and 2) the adapter-based fine-tuning also delivers comparable performance while being notably parameter-efficient. Furthermore, we empirically demonstrate the favorable effect of our method in terms of cross-distribution application and versatility. The code is available: https://github.com/Zhuoyi-Lin/Cross_problem_learning.

1 Introduction
Recently, there has been a growing trend toward applying neural methods based on deep learning to solve Combinatorial Optimization Problems (COPs), commonly known as Neural Combinatorial Optimization (NCO) [Bengio et al., 2021]. Among the studied COPs, the Vehicle Routing Problems (VRPs) are often favoured and chosen to verify the effectiveness of the NCO methods, especially the Traveling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP). On the one hand, VRPs are widely applied in real-world scenarios such as logistics, and drone delivery [Wang and Sheu, 2019; Konstantakopoulos et al., 2022]. On the other hand, VRPs are known to be NP-complete problems, and many of them are challenging to be solved efficiently. With the advances of deep learning and its power to automatically learn neural heuristics, NCO methods have demonstrated notable promise against traditional heuristics [Kool et al., 2018; Kwon et al., 2020; Li et al., 2021; Luo et al., 2023]. To further strengthen NCO methods, a number of recent endeavors have been paid to enhance generalization capabilities, which attempt to ameliorate the performance of the neural heuristics in solving the VRP instances with distributions or sizes unseen during training [Geisler et al., 2022; Bi et al., 2022; Qiu et al., 2022].

While gaining promising outcomes, current NCO methods generally learn a neural heuristic solely for each specific VRP, which brings about several issues. Firstly, such a learning paradigm would not be practical, since considerable models need to be trained from scratch when a number of VRPs are supposed to be solved, such as TSP and its variants like Orienteering Problem (OP) and Prize Collecting TSP (PCTSP). Secondly, the mainstream neural heuristics for VRPs are mostly developed based on heavy deep architectures like the Transformer model, and thus training neural heuristic even for a middle-sized problem may incur massive computational overhead and memory cost [Kwon et al., 2020; Kim et al., 2022b; Luo et al., 2023]. Lastly, the learned neural heuristic for a VRP tends to be ignored entirely when another VRP needs to be solved, with potentially transferable knowledge wasted. In practice, many VRP variants share the same or similar problem structure (e.g., VRPs are always featured by customer locations denoted by node coordinates), and only differ in a few constraints in their mathematical formulations. Accordingly, the corresponding neural heuristics are only distinct in a few layers of their respective neural architectures. Hence a more efficient training paradigm could
leverage common parts of neural networks to transfer learned parameters, e.g., the part of neural network trained for TSP could be re-used to train the neural heuristic for OP.

This paper mainly answers the question “whether the trained neural heuristic for a VRP could benefit the training for other similar variants, and how could it be deployed?” To this end, we first pre-train a neural heuristic for TSP using the Transformer model, and then leverage the pre-trained model as the backbone to foster the learning of neural heuristics for other (relatively) complex VRP variants via fine-tuning. Specifically, we first modularize neural architectures for complex VRPs based on the backbone Transformer model for TSP, where additional lightweight modules beyond the backbone are applied to process problemspecific features in complex VRPs. Then, we directly apply the pre-trained backbone for TSP [Kool et al., 2018; Kwon et al., 2020] in the training process of neural heuristics for the complex VRPs, by fully fine-tuning the backbone along with problem-specific modules. Finally, we propose three adapter-based fine-tuning methods to further increase the efficiency of parameter usage. Extensive experiments substantiate that the fully fine-tuned Transformers on complex VRPs significantly outperform Transformers trained from scratch on each of the VRPs. Adapter-based fine-tuning methods are re-used in downstream tasks. Also, the knowledge learned during pre-training is transferred to fine-tuning process to facilitate more efficient training.

2 Related Work

In this section, we introduce the literature on neural heuristics for solving VRPs, and briefly discuss the pre-training-then-fine-tuning works in different fields.

2.1 Neural Heuristics for VRP

Current neural heuristics for VRPs mostly learn policies to construct a solution in an autoregressive way. Among diverse neural construction heuristics [Bello et al., 2017; Nazari et al., 2018; Dai et al., 2017; Ma et al., 2023], the first breakthrough is made by the Transformer [Vaswani et al., 2017], which is cast as Attention Model (AM) to solve VRP variants such as TSP, OP and PCTSP [Kool et al., 2018]. After that, numerous neural heuristics are proposed based on AM for solving VRPs [Xin et al., 2021a; Kwon et al., 2020; Hottung et al., 2022]. Among them, Kwon et al. significantly ameliorate AM with policy optimization with multiple optima (POMO) [Kwon et al., 2020], and enlighten considerable follow-up works with Transformers to solve VRPs [Kim et al., 2022b; Kwon et al., 2021; Choo et al., 2022]. As alternatives, some works endeavor to predict probabilities of edges in optimal solutions to VRPs, which are then leveraged to efficiently construct high-quality solutions with search algorithms such as greedy search, sampling, and Monte Carlo tree search [Fu et al., 2021; Qiu et al., 2022; Hudson et al., 2022]. On the other hand, neural improvement heuristics aim to learn policies to iteratively improve an initial but complete solution. Inspired by traditional local search algorithms, a few works learn local operations such as 2-opt and swap to promote the solution [Wu et al., 2021; Ma et al., 2021; Wang et al., 2021; Kim et al., 2022a]. Inspired by specialized algorithms or solvers for VRPs, some literature attempts to learn repair or destroy operations in Large Neighborhood Search (LNS) [Hottung and Tierney, 2020; Gao et al., 2020]. Deep learning is also applied to enhance solvers like Lin-Kernighan-Helsgaun (LKH) [Xin et al., 2021b; Kim et al., 2021] and Hybrid Genetic Search (HGS) [Santana et al., 2022].

2.2 Pre-Training, Fine-Tuning

The pre-training-then-fine-tuning paradigm has been widely verified effective in domains of Natural Language Processing (NLP) [Dong et al., 2019; Devlin et al., 2018], Graph Neural Network (GNN) [Qiu et al., 2020; Hu et al., 2020], Computer Vision (CV) [Bao et al., 2022; Chen et al., 2020], etc. In this line of work, large models such as Transformers are first trained on basic tasks [Dong et al., 2019; Ribeiro et al., 2020; Hu et al., 2020; You et al., 2020; Dosovitskiy et al., 2021; Touvron et al., 2021]. Then the pre-trained model is fine-tuned to solve downstream tasks, with the neural network adapted by introducing additional structures or layers (which possess far fewer parameters than the pre-trained model) [Houlsby et al., 2019; Pfeiffer et al., 2021; Zhao et al., 2020; Kitaev et al., 2020; Hu et al., 2021]. In this manner, the pre-training can be conducted on considerable unlabeled data to learn general patterns and features, which are re-used in downstream tasks. Also, the knowledge learned during pre-training is transferred to fine-tuning process to facilitate more efficient training.

In this paper, the cross-problem learning method brings a similar pre-training-then-fine-tuning paradigm to the field of learning to route. A Transformer is pre-trained for solving TSP by DRL with massive unlabeled data, which is then leveraged to fine-tune Transformers for downstream VRPs.
We describe a VRP instance on a graph \( G = \{ V, E \} \) with \( V = \{ v_i \}_{i=0}^{n} \) signifying the node set and \( E = \{ (v_i, v_j) | v_i, v_j \in V, v_i \neq v_j \} \) signifying the edge set. Specially, \( v_0 \) denotes the depot if needed, e.g., in OP. On the graphs of VRP instances, nodes are often featured by coordinates \( \{ x_i \}_{i=0}^{n} \) to gain the tour length, and other problem-specific attributes (e.g. prizes \( \{ p_i \}_{i=1}^{n} \) in OP). A solution to a VRP instance is a tour \( \pi \), which is a sequence of nodes in \( V \). Given a cost/utility function \( \mu(\cdot) \), the VRP instance is solved by searching the tour with the least cost or highest utility, i.e., \( \pi^* = \arg \min_{\pi \in \mathcal{Y}} \mu(\pi|G) \) or \( \arg \max_{\pi \in \mathcal{Y}} \mu(\pi|G) \), from all feasible tours \( \mathcal{Y} \). Concretely, the cost function for TSP defines the (Euclidean) length of a tour and the utility function for OP defines the total prize gathered at each node.

The solution to a VRP instance is feasible if it satisfies the problem-specific constraints. For example, a feasible solution to a TSP instance is a tour that goes through every node on the graph exactly once. A solution to OP is feasible if the length of a tour is not larger than a given value. Other VRPs like PCTSP and CVRP are defined on graphs in a similar fashion.

### 3.1 VRP Description

In this section, we describe VRPs including TSP, OP, PCTSP and CVRP, which are used in experiments to evaluate our method. Afterwards, we introduce the general DRL method to learn construction heuristics with Transformers. Finally, we analyze the paucity of existing neural heuristics.

### 3.2 Transformer Based Construction Heuristics

Formally, the process of constructing a solution to an instance on \( G \) is cast as a Markov Decision Process (MDP). Given the state (i.e., the instance and partial solution) and action candidates (i.e., nodes that can be added into the partial solution), the policy (i.e., neural construction heuristic) is learned by an encoder-decoder based Transformer with parameters \( \theta \). At each step of the solution construction, the Transformer takes in raw features of the state and infers probabilities of each node, by which a node is selected in a greedy or sampling way. The probabilistic chain rule of constructing a tour \( \pi \) can be expressed as \( p_{\theta}(\pi|G) = \prod_{t=1}^{T} p_{\theta}(\pi_t|G, \pi_{<t}) \), where \( \pi_t \) and \( \pi_{<t} \) refer to the selected node and current partial solution at step \( t \). \( T \) denotes the maximum number of steps. Given the reward defined as the negative cost (or utility) of a tour, the REINFORCE [Williams, 1992] algorithm is often leveraged to update the policy with the following equation:

\[
\nabla_{\theta} \mathcal{L}(\theta|G) = \mathbb{E}_{p_{\theta}(\pi|G)}[(\mu(\pi) - b(G))\nabla_{\theta} \log p_{\theta}(\pi|G)],
\]

where \( \mathcal{L}(\theta|G) = \mathbb{E}_{p_{\theta}(\pi|G)}[\mu(\pi)] \) is the expected cost and \( b(\cdot) \) is a baseline function to reduce the variance of the estimation and raise the training efficiency [Sutton and Barto, 2018].

### 4 The Proposed Method

The cross-problem learning for solving VRPs falls within the pre-training-then-fine-tuning paradigm. Given the Transformer for a basic VRP (TSP in this paper), we train it to learn neural heuristics for solving the basic VRP with a DRL algorithm. Then, the pre-trained Transformer as the backbone is applied to train neural heuristics for downstream VRPs by fine-tuning. According to the modularization of Transformers, we propose different fine-tuning methods, which fully fine-tune the backbone with problem-specific modules, or only fine-tune lightweight problem-specific modules with small adapter networks. In the below, we elaborate on the modularization of Transformers for VRPs, the pre-training method for TSP by DRL, and different fine-tuning methods in detail.

#### 4.1 Modularization of Transformers for VRPs

Most neural heuristics for VRPs are developed from Transformers [Kool et al., 2018; Kwon et al., 2020; Kwon et al., 2021] with similar neural architectures. To learn heuristics
for TSP, the coordinates of customer nodes \( \{x_i\}_{i=1}^n \) are linearly projected to initial embeddings \( \{h_0^i\}_{i=1}^n \). The encoder in a Transformer processes \( \{h_0^i\}_{i=1}^n \) to derive the advanced node embeddings \( \{h_f^i\}_{i=1}^n \), which are then used in the decoder to output probabilities of nodes to be selected, as shown in Figure 1(a). Based on the Transformer for TSP, we mod-ularize neural architectures for complex VRPs into the backbone Transformer (for TSP) and problem-specific modules.

Taking OP as an example (shown in Figure 1(b)), the Transformer resorts to additional modules (i.e., linear layers) to process the coordinate of depot \( x_0 \) and prizes on customer nodes \( \{p_i\}_{i=1}^n \), respectively. The linearly transformed prizes are added to the initial embeddings of coordinates to complement the information on customer nodes in OP. Then the resulting embeddings \( \{h_0^{0i}\}_{i=1}^n \) are concatenated with the initial embedding of the depot node \( h_0^0 \), which together are taken as input to the decoder. Similarly, the decoder uses one additional module (i.e., linear layer) to inject the dynamic feature, i.e., the remaining max length \( T_i \) at the \( t \)-th step, into the query vector for the subsequent computation. Note that we also dump the unnecessary information in TSP from the query embedding, e.g., the embedding of the firstly selected node, which is always the depot and thus senseless in OP.

Consequently, the Transformer for OP is modularized into the backbone Transformer (for TSP) plus the three additional linear layers to process additional features. Similarly, we can modularize Transformers for other VRPs such as PCTSP and CVRP. Taking PCTSP as an example, we only need modules to transform 1) the depot’s coordinate, prizes, and penalties before the encoder, and 2) the remaining prize in the decoder.

4.2 Pre-Training on TSP

Given the above modularization, we propose to first train the backbone Transformer for TSP, and then fine-tune the problem-specific modules for downstream tasks. The rationale behind the use of TSP as the basic task for pre-training is that 1) as showed in the modularization, the Transformer for TSP is included in Transformers for other VRPs (e.g., OP, PCTSP); 2) as a basic VRP, TSP is merely featured by node coordinates, which are also included in other VRPs. Hence, we can leverage the backbone Transformer for TSP to learn useful node representations to reflect the common knowledge in VRPs (e.g., node locations and their distances), and then insert the backbone in Transformers for downstream VRPs.

In this paper, we take Transformer based AM as the backbone to pre-train the neural heuristic for TSP, since AM is able to solve a wide range of VRPs with strong performance. We illustrate the neural architecture of AM in Figure 1(a), and refer interested readers to the original work in [Kool et al., 2018] for more details. We will focus more on the usage of the pre-trained AM in fine-tuning for downstream VRPs.

4.3 Full Fine-Tuning for VRPs

A straightforward fine-tuning method is to leverage the pre-trained AM for TSP as a warm start for training a Transformer for any downstream VRP. Specifically, we load the parameters of pre-trained AM to the corresponding part of the Transformer, and leave the parameters on the extra problem-specific modules randomly initialized. In this paper, we uniformly initialize their parameters within the range \( (-1/\sqrt{d}, 1/\sqrt{d}) \), where \( d = 128 \) is the hidden dimension in AM. Taking the OP as an example, we apply three linear layers \( LL_k(z) = W_k \cdot z + b_k \), \( k = \{0, 1, 2\} \) to process the coordinate of depot node, prizes of customer nodes, and total prize gathered, respectively. The trainable parameters \( W_0 \in \mathbb{R}^{2 \times d}; W_1, W_2 \in \mathbb{R}^{1 \times d}; b_0, b_1, b_2 \in \mathbb{R}^d \) are uniformly initialized. During training, we tune parameters in both the backbone and additional problem-specific modules with the same optimizer and learning rate, following the RE-INFORCE mentioned in Section 3.2. We empirically observe that such simple fine-tuning is able to significantly increase the training efficiency, in comparison to existing methods that train neural heuristics from scratch for each VRP.

4.4 Adapter-Based Fine-Tuning for VRPs

Despite the handy implementation of full fine-tuning, it still depends on massive parameters to be trained for every downstream VRP. Given a broad spectrum of VRP variants, it would not be realistic to train and store many heavy models with limited memory resources. An ideal alternative is that we only store problem-specific modules along with small adapter networks, while freezing the parameters of the pre-trained backbone. Despite similar adaptation techniques
in other domains [Zhang et al., 2020; Sung et al., 2022; Hu et al., 2021], we take an early step to propose three fine-tuning methods with adapter networks for solving VRPs.

The adapter network in the inside tuning comprises two linear layers with a LeakyReLU (LR) activation function in between such that:

\[ \text{APT}_\text{in}(h_i) = W_i^p \cdot LR(W_{0i}^i \cdot h_i + b_{0i}^i) + b_{1i}^i, \]

where \( i \in \{0, \ldots, n\} \) is the node index. We set the trainable parameters \( W_{0i}^i \in \mathbb{R}^{d \times (d/2)}; W_{1i}^i \in \mathbb{R}^{(d/2) \times d}; b_{0i}^i \in \mathbb{R}^{d/2}; b_{1i}^i \in \mathbb{R}^d \), which are uniformly initialized with the range \((-1/\sqrt{d}, 1/\sqrt{d})\). We put the above adapter network right after the self-attention layer and fully-connected layer in the backbone Transformer’s encoder, as shown in Figure 2(a).

Instead of adjusting intermediate embeddings in the encoder, the side tuning adjusts the output from the encoder, with the following neural architecture:

\[ h_i' = \text{BN}(h_i + \text{MSL}_i(h_0, \ldots, h_n)), \]
\[ h_i'' = \text{BN}(h_i' + \text{FL}(h_i')), \]
\[ \text{APT}_\text{out}(h_i'') = \text{BN}(LR(W_i^2 \cdot \text{BN}(LR(W_i^1 \cdot h_i'' + b_{0i}^1)) + b_{1i}^1)), \]

where \( i \in \{0, \ldots, n\} \) is the node index. In Eqs. (3) and (4), we use the similar neural structure in the encoder of AM (once rather than \( L \) times), where it processes node embeddings by Multi-head Self-attention Layer (MSL) and Fully-connected Layer (FL), with either followed by skip connection and batch normalization (BN). More details of these components can be found in [Kool et al., 2018]. In Eq. (5), the node embeddings are further evolved by a linear layer, the LeakyReLU activation and batch normalization, where \( W_{0i}^i, W_{1i}^i \in \mathbb{R}^{d \times d}, b_{0i}^1, b_{1i}^1 \in \mathbb{R}^d \) are trainable parameters and uniformly initialized. This adapter network is put beside the encoder and evolves the initial node embeddings \( \{h_i^0\}_{i=0}^n \) into \( \{z_i^n\}_{i=0}^n \), which are added into the output from the encoder (with frozen backbone weights), as shown in Figure 2(b).

Inspired by Low-Rank Adaptation (LoRA) in [Hu et al., 2021], we design a low-rank decomposition to adjust the output from any matrix in the pre-trained Transformer,

\[ \text{APT}_\text{lo}(h_i) = W_p \cdot h_i + B_p A_p \cdot h_i, \]

where \( i \in \{0, \ldots, n\} \) is the node index. \( W_p \in \mathbb{R}^{d \times d} \) represents the pre-trained weight matrix in the backbone Transformer for TSP; \( B_p \in \mathbb{R}^{d \times r} \) and \( A_p \in \mathbb{R}^{r \times d} \) are two trainable matrices with \( r = 2 \ll d \), which we initialize by 0 and Gaussian(0,1). We apply Eq. (6) to adjust the output from query, key, and value matrices in the encoder of pre-trained Transformer, with one example shown in Figure 2(c). Meanwhile, we exert LoRA on the weight matrix in the initial linear projection for better results we can find.

5 Experiments

We empirically evaluate cross-problem learning from different perspectives. Firstly, we compare it with other baselines to show its performance on OP and PCTSP. Then, we elaborate on the key designs in the fine-tuning methods. Finally, we demonstrate the effectiveness of our method, in terms of cross-distribution application and versatility.

Following the existing works, we generate the coordinates of nodes by uniformly sampling from the unit square \([0, 1] \times [0, 1]\). For OP, we set prizes all equal to 1. For PCTSP, we randomly sample prizes and penalties from Uniform distributions, following [Kool et al., 2018]. Meanwhile, we generate different-sized instances for OP and PCTSP, by setting the number of customer nodes to \( n = 20, 50, \) and 100. We then use the Transformer based AM in [Kool et al., 2018] to implement the cross-problem learning, and maintain most of the original experimental settings for a fair comparison. Specifically, we train the model with 2500 batches in each epoch and use 512 instances in each batch (except for the problem with \( n = 100 \), where we use 256 instances). For each epoch, the training instances are generated on the fly, and we use 10,000 instances for the validation. Then the paired t-test (\( \alpha = 5\% \)) is conducted after each epoch to replace the baseline model, according to the improvement significance. For all problems and sizes, we train 100 epochs for the full convergence of the training curves. The learning rate is set as \( 10^{-4} \) in all our methods, as the one for the original AM\(^2\). The server with Intel Core i9-10940X CPU and NVIDIA GeForce RTX 2080Ti is used in our experiments.

We compare our method with AM [Kool et al., 2018], the DRL based method with a strong performance on a variety of VRPs. We deploy the cross-problem learning based on its Transformer models. We also compare various baselines as follows. Regarding OP, we include 1) Gurobi, the state-of-the-art exact solver for various COPs [Gurobi Optimization, LLC, 2022]; 2) Compass, the specialized Genetic Algorithm (GA) for OP [Kobeaga et al., 2018]; 3) Tsili, a classic randomized construction heuristic for OP with the manually crafted node probabilities [Tsiligirides, 1984]. As for PCTSP, besides Gurobi, we run two additional baselines: 4) ILS, the iterated local search metaheuristic that is widely used for routing problems; and 5) OR-Tools, a mature metaheuristics based solver for routing problems.

5.1 Comparison

We test all methods with 10,000 instances of OP and PCTSP, two typical VRPs that aim to maximize the total prize and minimize the distance plus the total penalty in the tour, respectively. We report the results of AM and Tsili by two decoding strategies, i.e., greedy search and sampling. For our method, we only report the better results derived by sampling. The sampling size is 1,280 for these methods, following [Kool et al., 2018]. For adapter-based fine-tuning, we only report results derived by inside tuning, since it significantly outstrips side tuning and LoRA (see Section 5.2). We compare the methods by different metrics, including the average objective value, average gap (to the best solutions), total runtime, and the number of trained parameters (for learning based methods). In Table 1, we can observe that our method with either full fine-tuning or inside tuning consistently out-

\(^2\)We tried to assign different learning rates to adapter networks and problem-specific modules for fine-tuning, which did not result in better performance.
performs AM with either greedy search or sampling, with significantly smaller gaps and similar runtime. While Gurobi achieves the optimal solutions for small problems, it cannot derive solutions for middle-sized OP and PCTSP ($n=50, 100$). Instead, we run Gurobi as a heuristic with a 30s limit, but it is inferior to our methods on OP ($n=50, 100$) or fails to obtain any solutions on PCTSP ($n=100$). While Compass and ILS are specialized algorithms for OP and PCTSP respectively, our method achieves comparable results in all cases, with most gap differences smaller than 1%. Moreover, our methods significantly surpass Tsili on OP ($n=20, 50, 100$), and outperform OR-Tools on PCTSP ($n=100$). Lastly, the inside tuning costs far fewer parameters for training, in contrast to AM and the full fine-tuning, which reduces considerable memory resources for solving each VRP (except the backbone Transformer for TSP). Besides transferring knowledge between problems with the same (Uniform) distribution, the cross-problem learning can effectively solve downstream VRPs with different distributions. Results on such cross-distribution applications are provided in Appendix B.

**5.2 Ablation Study**

In this part, we conduct an ablation study to demonstrate the impact of key designs for the proposed fine-tuning methods. Given the same pre-trained backbone, we first compare the inside tuning, the side tuning, and LoRA. Then, we show the sensitivity of the inside tuning approach by varying activation functions, including Sigmoid, Tanh, and LeakyReLU. With the changed components, we draw training curves on PCTSP ($n=20, 50, 100$) and display them in Figure 3. To investigate the effects of different training methods, we also add the curves for the training from scratch and the full-tuning. As shown, although side tuning is equipped with a larger adapter network (i.e., 332K vs. 199K), it is significantly inferior to inside tuning on all sizes. On the other hand, despite the discounting performance of LoRA on PCTSP ($n=20$), it achieves similar results with side tuning for $n=50$ and is significantly superior to side tuning for $n=100$ with only 127K parameters. In addition, the activation function influences the performance, and the inside tuning with LeakyReLU outperforms the one with Sigmoid or Tanh. Notably, the curves of the inside tuning drop faster in the early epochs than the original AM trained from scratch on PCTSP ($n=50, 100$), and the inside tuning with LeakyReLU achieves similar convergences to AM on all sizes, with far fewer trained parameters.

**5.3 Versatility**

The cross-problem learning is versatile to different deep models for VRPs. We further deploy it with POMO [Kwon et al., 2020], another Transformer-based neural network which
is developed from AM [Kool et al., 2018] and achieves the state-of-the-art performance in solving TSP and CVRP. More details of POMO can be found in Appendix C. Specifically, we keep all the experimental settings with the same hyperparameters for training the original POMO. For our fine-tuning methods, we train POMO with the batch size 64, and the instances generated on the fly. For each epoch, we train 100,000 instances with them divided into batches. The deep model is trained with 100 epochs, when we observe that the training curves already converge. For the learning rate, we set $\eta = 10^{-4}$ for the problem size $n = 20$, and $\eta = 10^{-5}$ for $n = 50, 100$ to avoid overfitting in the training process. As shown in Figure 4, the full fine-tuning generally converges faster and better than POMO trained from scratch. Furthermore, the inside tuning attains close performance to POMO, saving 79.59% parameters (i.e., 0.256M vs. 1.254M), which is more significant than the saving with AM (i.e., 71.33%).

We demonstrate the results of POMO and our methods by two inference settings, i.e., with and without the instance augmentation (denoted as no aug. and $\times 8$ aug., respectively.) [Kwon et al., 2020]. We evaluate all methods on 10,000 CVRP instances for each of the sizes $n = 20, 50, 100$. We list the average objective value, average gap (to the best solution), total inference time and the number of trained parameters in Table 2. As shown, our method with the full fine-tuning consistently performs better than POMO across all problem sizes. In particular, when the number of nodes $n$ is small, i.e., $n = 20$, POMO finds solutions with an average gap of 0.29% compared with the full fine-tuning method. On the other hand, our method with the inside tuning performs slightly worse than POMO (e.g. 0.06% vs. 0.39% for $n = 100$). This indicates that the adapter-based methods are able to achieve comparable performance with much fewer parameters for different deep learning backbones. As for the side tuning method, its performance is comparable to the inside tuning for $n = 20, 50$, but becomes drastically inferior for $n = 100$. Compared with side tuning, LoRA achieves comparable performance for $n = 20$, 50 and better performance (i.e., 0.88% vs. 1.33%) for $n = 100$ by saving 90.27% (i.e., 0.122M vs 1.254M) parameters. To further evaluate our methods, we apply them to solve benchmark instances in CVRP-LIB. The results show that the full fine-tuning significantly outperforms POMO on large-scale instances with sizes up to 1000. Detailed results are presented in Appendix D.

### 6 Conclusion and Future Work

This paper presents a cross-problem learning method for solving VRPs, which assists heuristics training for downstream VRPs by using the pre-trained Transformer for a basic VRP. We modularize Transformers for VRPs based on the backbone (Transformer) for TSP, and propose different fine-tuning methods to train the backbone or problem-specific modules in the target VRP. Results show that the full fine-tuning significantly outperforms Transformer trained from scratch, and the adapter-based fine-tuning delivers comparable performance, with far fewer parameters. We also verify the favorable cross-distribution application and versatility of our method, along with the efficacy of key components. In the future, we will improve our methods with more advanced techniques such as neural architecture search. We also plan to apply our method to solve other COPs, such as job shop scheduling problem or bin packing problem.

![Table 2: Inference results with POMO.](image)

![Figure 4: Training curves with POMO on CVRP (n = 20, 50, 100), respectively.](image)
Acknowledgments

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031), and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

References


[Qiu et al., 2020] Jiezhuo Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In KDD, pages 1150–1160, 2020.


