# Learning Generalizable Models for Vehicle Routing Problems via Knowledge Distillation

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Recent neural methods for vehicle routing problems always train and test the deep models on the same instance distribution (i.e., uniform). To tackle the consequent cross-distribution generalization concerns, we bring the *knowledge distillation* to this field and propose an *Adaptive Multi-Distribution Knowledge Distillation* (AMDKD) scheme for learning more generalizable deep models. Particularly, our AMDKD leverages various knowledge from multiple teachers trained on exemplar distributions to yield a light-weight yet generalist student model. Meanwhile, we equip AMDKD with an adaptive strategy that allows the student to concentrate on difficult distributions, so as to absorb hard-to-master knowledge more effectively. Extensive experimental results show that, compared with the baseline neural methods, our AMDKD is able to achieve competitive results on both unseen in-distribution and out-of-distribution instances, which are either randomly synthesized or adopted from benchmark datasets (i.e., TSPLIB and CVRPLIB). Notably, our AMDKD is generic, and consumes less computational resources for inference.

## 1 Introduction

The *Vehicle Routing Problem* (VRP) is a class of NP-hard combinatorial optimization problems with a wide variety of practical applications, such as freight delivery [1], last-mile logistics [2] and ride-hailing [3]. For decades, the problem has been studied intensively in computer science and operations research, with numerous exact and (approximate) heuristic algorithms proposed [4–7]. Although the heuristic algorithms are usually preferred in practice given their relatively high computational efficiency, they heavily rely on hand-crafted rules and domain knowledge, which still leave much room for improvement. As a promising alternative, deep (reinforcement) learning could be used to automatically learn a heuristic (or policy) for VRPs in an *end-to-end* fashion, which has aroused widespread attention in recent years [8–15]. Compared to the traditional ones, the learned heuristics based on deep models could further reduce computational costs while ensuring desirable solutions.

However, existing deep models suffer from inferior generalization with respect to distributions of node coordinates. To be concrete, they often train and test neural networks on instances of the same distribution, mostly the uniform distribution, where deep models are able to achieve competitive results more efficiently than the traditional heuristics (e.g., [13]). Nevertheless, when the learned policy is applied to infer the *out-of-distribution* (OoD) instances, the solution quality is usually low. This cross-distribution generalization issue inevitably hinders the applications of deep models, especially because the real-world VRP instances may follow various and even unknown distributions.

A number of preliminary attempts have been made to tackle this generalization issue for VRPs, which mainly leverage (automatic) data augmentation [16–19] and distributionally robust optimization [20], respectively. However, they are not optimal in our view, as the former always starts with a specified single distribution which may limit the resulted performance, while the latter needs to manually

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

define major and minor instances. Different from them, in this paper, we aim to enhance the cross-distribution generalization by transferring various policies learned from respective distributions into one, where we exploit *knowledge distillation* to learn more generalizable models for VRPs.

Specifically, we propose a generic *Adaptive Multi-Distribution Knowledge Distillation* (AMDKD) scheme for training a light-weight model with favorable cross-distribution generalization performance. To impart broad yet specialized knowledge, we exploit multiple teachers that are trained on respective *exemplar* distributions. Our AMDKD then leverages those teachers to train a shared student model in turns, inspired by the knowledge learning paradigm of human-beings. Meanwhile, we also equip the AMDKD with an adaptive strategy to track the real-time learning performance of the student model on every exemplar distribution, and allow the student to concentrate on absorbing those hard-to-master knowledge, so as to strengthen the effectiveness of the learning.

Accordingly, our contributions are summarized as follows: (1) We bring the *knowledge distillation* to the field of *neural combinatorial optimization* and aim at improving the cross-distribution generalization for solving VRPs, which offers a promising perspective to transfer knowledge/policy learned from multiple models into one; (2) We propose the *Adaptive Multi-Distribution Knowledge Distillation* (AMDKD) scheme, where we distill diverse knowledge from multiple teachers trained on exemplar distributions to yield a light-weight yet generalist student model, and also present an adaptive strategy for the student to better assimilate hard-to-master knowledge from difficult distributions; (3) We apply our generic AMDKD to two representative deep models, i.e., AM [11] and POMO [13], respectively. Results show that, while consuming less computational resources for inference, our AMDKD performs favorably against the backbone models as well as other generalization methods for deep models, especially on the benchmark dataset, i.e., TSPLIB and CVRPLIB. By further coupling with the efficient active search (EAS) [21] during inference, our AMDKD achieves the state-of-the-art performance. We also conduct a series of ablation studies to verify our designs.

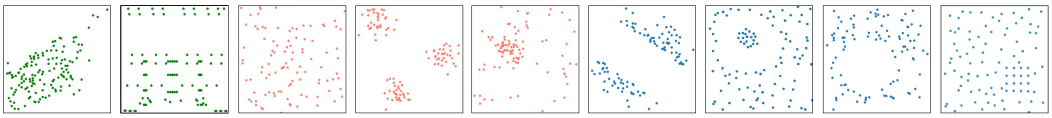
## 2 Related work

Recently, neural methods based on deep models for VRPs have aroused widespread interest, which are categorized as neural *construction* models and neural *improvement* models in general.

**Neural construction models.** They exploit deep (reinforcement) learning to autoregressively construct the solution in an end-to-end fashion. Vinyals et al. [8] introduced the first RNN-based Pointer Network (Ptr-Net) to solve TSP based on supervised learning. Bello et al. [9] then exploited reinforcement learning to train the Ptr-Net for TSP. In [10], the Ptr-Net was then extended to solve CVRP. Different from the above RNN-based methods, Kool et al. [11] proposed the well-known Attention Model (AM) based on the Transformer architecture [22]. Subsequently, many studies extended AM for routing problems, such as [13, 14, 23–25]. As a representative, Kwon et al. [13] proposed the POMO (Policy Optimization with Multiple Optima) and achieved significantly better performance. In a recent work, POMO was adapted to an efficient active search (EAS) framework to further boost the performance during inference [21]. Besides, graph neural network (GNN) are also utilized to more effectively learn and identify the problem features [1, 12, 26–28].

**Neural improvement models.** They iteratively improve the current solution by exploiting deep (reinforcement) learning to assit the local search. Chen and Tian [29] proposed the NeuRewriter that learned a policy to partially rewrite the current solution. Hottung and Tierney [30] leveraged deep networks to learn to perform the large neighborhood search. Wu et al. [31] used the Transformer to learn to perform 2-opt operation [6]. Ma et al. [15] further upgraded the deep model of Wu et al. [31] to dual-aspect collaborative Transformer (DACT) with much superior performance. Lu et al. [32] learned to select operators for CVRP and outperformed LKH3 [4]. Generally, improvement methods consume much longer inference time than construction ones to achieve higher solution quality.

**Cross-distribution generalization.** End-to-end neural methods for VRPs often train and test the models on the same instance distribution. Although the state-of-the-art deep models perform close to or even surpass the strong traditional heuristics, they are incapable to generalize the learned policy to other distributions, which seriously impairs their practical applications. Therefore, the cross-distribution issue has gradually gained more attention [16–20]. Among the early attempts in this line of reasearch, Xin et al. [17] proposed a GAN-based framework to generate hard-to-solve instances for training the model. Wang et al. [19] leveraged the game theory and proposed the Policy Space Response Oracle (PSRO) framework to simultaneously learn a trainable solver and an instance



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 1: TSP instances following various distributions from the literature: (a) gr137, (b) lin105, (c) Uniform, (d) Cluster, (e) Mixed, (f) Expansion, (g) Implosion, (h) Explosion, (i) Grid, where instances (a) and (b) are from TSPLIB [33]. In this paper, we consider instances following distributions (c)-(e) for training and other unseen distributions (f)-(i), as well as benchmark datasets for testing.

generator. Zhang et al. [16] designed a hardness-adaptive TSP instance generator and adopted curriculum learning to train the model. These methods primarily emphasized on data augmentation or are limited to TSP. Different from them, Jiang et al. [20] adopted CNN to acquire distribution-aware features and exploited group DRO (Distributionally Robust Optimization) to enhance model robustness against distributions. However, this method needs to manually define major and minor instances. Note that we acknowledge that the generalization to different problem scales (or sizes) is also important, which we leave as the future research direction.

# 3 Preliminaries and notations

We first present the formulation of the studied VRPs and classic distributions. Then we introduce how deep models are used to solve them, followed by the basic rationale of knowledge distillation.

## 3.1 VRPs and their distributions

We define VRPs over a complete graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $v_i \in \mathcal{V}$ represents the (customer) node, $e(v_i, v_j) \in \mathcal{E}$ represents the edge between two nodes, and $C[e(v_i, v_j)]$ represents the cost (we use *length* in this paper) of the edge. By referring tour $\tau$ (a.k.a. solution) to a permutation of nodes in $\mathcal{V}$, the objective is usually to find the optimal tour $\tau^*$ with the least total length over a finite search space $S$ containing all possible tours, which could be formulated as Eq. (1) in general,

$$\tau^* = \arg\min_{\tau' \in \mathcal{S}} L(\tau' | \mathcal{G}) = \arg\min_{\tau' \in \mathcal{S}} \sum_{e(v_i, v_j) \in \tau'} C\left[e(v_i, v_j)\right]. \tag{1}$$

For different VRP variants, such objective may be subject to different problem-specific constraints, where multiple sub-tours may exist in a valid tour $\tau$. Following the recent literature [11, 13, 21], we focus on two representative VRPs, i.e., TSP and CVRP, respectively. A feasible tour for TSP considers a vehicle visiting each node in $\mathcal{V}$ once and only once. As an extension from a vehicle to a fleet, CVRP considers an extra depot node $v_0$ and a capacity limit $Q$ for any given vehicle, where each customer node $v_i (i = 1, ..., n)$ is associated with a demand request $\delta_i$. A feasible tour for CVRP consists of multiple sub-tours, each of which represents a vehicle in the fleet departing from the depot, serving a subset of nodes, and finally returning to the depot. The total demand of each sub-tour must not exceed the capacity $Q$, and all nodes except for the depot must be visited once and only once.

In Figure 1, we present a number of exemplary TSP instances following various distributions from the literature [11, 20, 34] including TSPLIB [33]. As can be observed, the node coordinates of a VRP instance in real-world application may follow complicated and even unknown distribution, which considerably intensifies the hardness for solving. While the recent neural methods report superior performance to the traditional heuristics on some fixed distributions, it is unfortunate that they are more sensitive to distribution shift. It is thus of great importance and interest to develop powerful deep models that can simultaneously handle as many diverse distributions as possible.

## 3.2 Tour construction by deep models

We focus on neural construction methods for VRPs, which usually exploit deep neural networks to sequentially construct the tour. In light of this, the solving procedure is mostly modelled as a Markov Decision Process (MDP), where the Transformer styled [11] architectures following the encoder-decoder structure are often adopted as the policy network. Typically, the encoders project the

nodes of the instance into node embeddings for feature extraction. Afterwards, the decoder builds the tour $\tau$ based on learned node embeddings and the partial tour $a_{1:t-1}$ constructed previously. At time step $t$ of the MDP, the decoder picks an unvisited node as the action $a_t$ where invalid ones are masked for feasibility. The procedure is repeated until the whole tour is completed, which is factorized as,

$$p_\theta(\tau|\mathcal{G}) = \prod_{t=1}^{\ell} p_\theta(a_t|a_{1:t-1}, \mathcal{G}), \tag{2}$$

where $\ell$ denotes the final time step in MDP and $\theta$ refers to trainable parameters. For TSP, $\ell = n$; for CVRP, $\ell \geq n$ since the depot can be visited more than once[1]. The total reward is defined as the negative of the tour length, i.e., $-L(\tau|\mathcal{G})$, which is essentially consistent with the objective in Eq. (1).

### 3.3 Knowledge distillation

Knowledge distillation [35, 36] is a kind of teacher-student training paradigm that aims at transferring knowledge from a (group of) complex teacher model(s) $\theta^{\text{T}}$, to a succinct student model $\theta^{\text{S}}$. The recent research findings reveal that knowledge distillation is not only able to effectively learn a lighter student network from larger teacher network(s) [37, 38], but also has potential to improve the generalization [39–41] even over its teacher(s) [42, 43]. Typically, the teacher models are pre-trained, and the student model compares the output of teacher model(s) with its own and considers it as a supervisory signal. Formally, the student network is trained with the goal of minimizing a weighted combination of the distillation loss $\mathcal{L}_{\text{KD}}$ and the original task loss $\mathcal{L}_{\text{Task}}$ as follows,

$$\mathcal{L} = \alpha\mathcal{L}_{\text{Task}} + (1-\alpha)\mathcal{L}_{\text{KD}}, \tag{3}$$

where $\alpha \in [0, 1]$. The $\mathcal{L}_{\text{KD}}$ with respect to one or more teachers ($N_T \geq 1$) is generally formulated as,

$$\mathcal{L}_{\text{KD}} = \frac{1}{N_T} \sum_{x \in \mathcal{X}} \sum_{i=1}^{N_T} \phi\left[p_{\theta_i^{\text{T}}}(x), p_{\theta^{\text{S}}}(x)\right], \tag{4}$$

where $x$ is the training data from $\mathcal{X}$ and $\phi(\cdot)$ evaluates the dissimilarity between the teacher and the student, such as the Kullback-Leibler divergence $\text{KL}(p_{\theta^{\text{T}}} \| p_{\theta^{\text{S}}}) = \sum_i p_{\theta^{\text{T}}}(\log p_{\theta^{\text{T}}} - \log p_{\theta^{\text{S}}})$.

## 4 Methodology

Consider what happens in a classroom when students study multiple modules. Typically, they (students) learn only one module (exemplar distribution) at a time from a professional teacher, and then go on to the next one until all modules are mastered. When students do poorly on a quiz after class (validation dataset), they will be required to study that module more frequently in order to become generalists. Motivated by this, we propose the *Adaptive Multi-Distribution Knowledge Distillation* (AMDKD) scheme that could efficiently learn more generalizable models for VRPs.

Previously, deep models were primarily trained on a single distribution, e.g., the Uniform in [13–15, 21], and their cross-distribution generalization is far from satisfactory. More recently, attempts have been made to expand the training set by augmenting atypical (minor) [20] or hard instances [16, 17]. Different from them, our AMDKD allows expert knowledge for tackling distinct distributions to be transferred into a single yet generalist student model via distillation. With a light network and high inference speed, the resulted model is supposed to perform favorably against existing methods. Moreover, our AMDKD is generic, and could be used to boost various deep models for VRPs.

### 4.1 Overall structure and teacher (pre)-training

The overview framework of our AMDKD is illustrated in Figure 2. Given an existing deep model (e.g., AM [11]), AMDKD first performs teacher training to obtain a teacher model (or directly use pre-trained ones) for each distribution listed in a set of *exemplar* ones. Then in each knowledge transfer epoch, AMDKD picks a specific distribution $d$ and its teacher $\theta_d^{\text{T}}$ to train a student network. To facilitate effective learning, the likelihood of picking each distribution is adaptively updated

[1] The sub-tours for CVRP are constructed sequentially. At each step, the agent selects an unvisited node whose demand is smaller than the remaining capacity or returns to the depot for full replenishment.

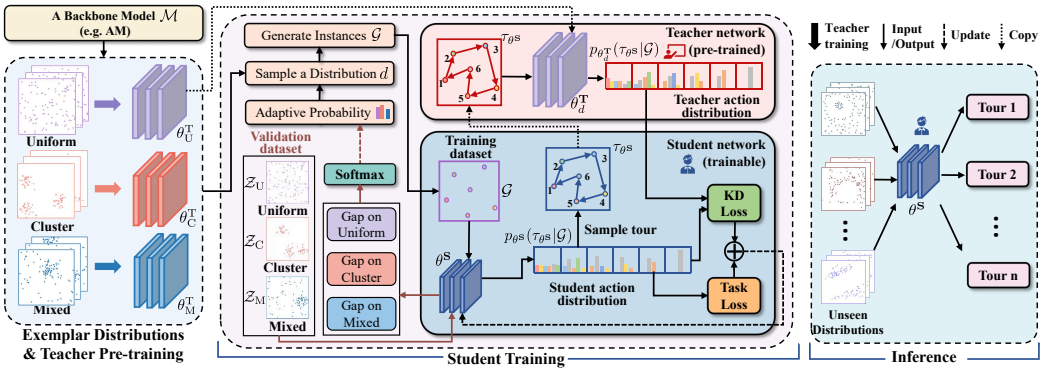



Figure 2: Framework of our AMDKD scheme. From left to right: teacher pre-training, student training and inference. The Uniform is selected as the current exemplar distribution for an example.

according to the current performance of the student on a validation dataset. Finally, AMDKD performs on-policy distillation, allowing the student network $\theta^{\mathrm{S}}$ to sample RL trajectories (i.e., tours $\{\tau_{\theta^{\mathrm{S}}}\}$) for training, and calculates the two losses (i.e., KD loss and Task loss) for an update.

We consider Uniform, Cluster, and Mixed (mixture of uniform and cluster) as exemplar distributions to train teacher networks in this paper. Consequently, a set of well-trained teachers with parameters $\theta^{\mathrm{T}} = \{\theta^{\mathrm{T}}_{\mathrm{U}}, \theta^{\mathrm{T}}_{\mathrm{C}}, \theta^{\mathrm{T}}_{\mathrm{M}}\}$ are attained for each distribution. Despite only limited distributions are exploited throughout training, the learned student network is expected to absorb generic and robust knowledge from teachers that may effectively generalize to unseen distributions such as Expansion, Implosion, Explosion, and Grid (visualization in Figure 1). Our motivation here is that each teacher only needs to master a specific distribution, and by working together via our distilling scheme, they can educate a generalist student. Note that the exemplar distributions used here could also be substituted with others (we will discuss in Appendix C). Since our designed distillation scheme is model-agnostic, the teacher network can follow most of existing architectures. In this paper, we assess our scheme by applying it to two representative construction methods, i.e., AM [11] and POMO [13], respectively.

### 4.2 Adaptive multi-distribution student training

Given the well-trained teacher networks on exemplar distributions, our AMDKD student selects a teacher and a distribution in each knowledge transfer epoch and gradually learns to make appropriate decisions. Note that there is another line of works that argue for the simultaneous use of multiple teachers in the distillation process (e.g., [36, 44]). In their strategy, all teachers are engaged in each epoch to provide a weighted loss to train the student as aforementioned in Section 3.3. However, such design may require teachers to be trained on a homogeneous task primarily with supervised learning, which does not suit our heterogeneous distributions with reinforcement learning (see Appendix C for results). Below we detail the student network and the adaptive multi-distribution distilling strategy.

**Student network.** In our design, we stipulate that the student shares a similar architecture with its teachers, but with much fewer parameters (and thus a higher inference speed). Specifically, the dimension of the node embeddings is curtailed from 128 (teacher) to 64 (student), resulting in a reduction of 61.8% and 59.2% in the model parameters for the adaption with AM and POMO, respectively. We also considered lowering the number of encoder layers, but the results were below the expectation. Detailed results and discussions are presented in Section 5.3.

**The adaptive multi-distribution distilling strategy.** As established, our AMDKD exploits an adaptive probability to select teachers in turn for each epoch. At the beginning, we select each distribution with an equal probability. After epoch $E'$ (a hyper-parameter), the probability would be adaptively adjusted according to the performance of the student on the given validation datasets $\mathcal{Z}_{\mathrm{U}}, \mathcal{Z}_{\mathrm{C}}, \mathcal{Z}_{\mathrm{M}}$ (each with 1,000 instances) for each exemplar distribution. The likelihood of selecting distribution $d \in \{U, C, M\}$ is proportional to the exponent value of the gaps to the LKH solver [4],

$$p^{\text{adaptive}}(d) = \begin{cases} \text{Softmax}\left(\text{AvgGap}\left[\{\tau_{\theta^{\mathrm{S}}}\}|_{\mathcal{Z}_d}, \{\tau_{\text{solver}}\}|_{\mathcal{Z}_d}\right]\right), & \text{if } E \geq E' \\ \frac{1}{|\mathcal{D}|}, & \text{otherwise} \end{cases} \tag{5}$$

**Algorithm 1** Adaptive Multi-Distribution Knowledge Distillation (AMDKD)

**Input:** A backbone model $\mathcal{M}$ (e.g., AM), exemplar distributions $\mathcal{D}$ (e.g., $\mathcal{D} = \{U, C, M\}$).
Randomly initialize teacher networks $\theta_d^{\text{T}}$ ($\forall d \in \mathcal{D}$) and student network $\theta^{\text{S}}$ according to $\mathcal{M}$;
Perform teacher training or leverage pre-train ones (if any) to attain well-learned $\theta_d^{\text{T}}$ ($\forall d \in \mathcal{D}$);
**for** epoch $= 1, 2, ..., E$ **do**
  Pick distribution $d$ and its teacher $\theta_d^{\text{T}}$ with an adaptive probability according to Eq. (5);
  **for** step $= 1, 2, ..., T$ **do**
    Let student $\theta^{\text{S}}$ sample tours $\tau_{\theta^{\text{S}}}^i$ for each $\{\mathcal{G}_i\}_{i=1}^B$ according to its own policy $p_{\theta^{\text{S}}}$;
    Get $\nabla\mathcal{L}_{\text{Task}}$ by estimating Eq. (6) as per the original design of $\mathcal{M}$;
    Get $\nabla\mathcal{L}_{\text{KD}}$ by computing the gradients of Eq. (7);
    $\theta^{\text{S}} \leftarrow \theta^{\text{S}} + \eta\nabla\mathcal{L}$ where $\nabla\mathcal{L} \leftarrow \alpha\nabla\mathcal{L}_{\text{Task}} + (1-\alpha)\nabla\mathcal{L}_{\text{KD}}$.
  **end for**
**end for**

where $|\mathcal{D}|$ is the number of exemplar distributions; $\{\tau_{\theta^{\text{S}}}\}|_{\mathcal{Z}_d}$ and $\{\tau_{\text{solver}}\}|_{\mathcal{Z}_d}$ refer to the tour set generated by the student $\theta^{\text{S}}$ and the LKH solver on dataset $\mathcal{Z}_d$, respectively. The validation sets are fixed, hence the LKH solver only needs to run once to attain $\{\tau_{\text{solver}}\}|_{\mathcal{Z}_d}$ for $d \in \{U, C, M\}$.

**Loss function.** Following Eq. (3), we leverage the task loss ($\mathcal{L}_{\text{Task}}$) and the KD loss ($\mathcal{L}_{\text{KD}}$) to jointly train the student network with a weight $\alpha = 0.5$. Pertaining to $\mathcal{L}_{\text{Task}}$, the reinforcement learning loss with respect to AM and POMO are expressed as follows,

$$\mathcal{L}_{\text{Task}} = -J(\theta^{\text{S}}|d) = -\mathbb{E}_{\mathcal{G}\sim d, \tau\sim p_{\theta^{\text{S}}}(\tau|\mathcal{G})}[L(\tau|\mathcal{G})], \tag{6}$$

where the training instances $\mathcal{G}$ are sampled following the selected distribution $d$, and the tours $\tau$ are constructed via the student network $\theta^{\text{S}}$ according to Eq. (2). In AM and POMO, the REINFORCE algorithm [45] is used to estimate the gradients for the above loss function and we follow exactly the same way as per their original designs. Pertaining to $\mathcal{L}_{\text{KD}}$, it is defined to encourage the student to imitate how a teacher network sequentially selects the nodes. Specifically, given the instance $\mathcal{G}$ sampled from distribution $d$ and a tour $\tau_{\theta^{\text{S}}}$ constructed by the student $\theta^{\text{S}}$ following its own policy, our AMDKD leverages $p_{\theta_d^{\text{T}}}(\tau_{\theta^{\text{S}}}|\mathcal{G})$ suggested by the teacher network $\theta_d^{\text{T}}$, to compute $\mathcal{L}_{\text{KD}}$ that measures the similarity of the probability distributions between teacher and student using the KL divergence,

$$\mathcal{L}_{\text{KD}} = \frac{1}{B}\sum_{i=1}^{B}\sum_{a_j\in\tau_{\theta^{\text{S}}}} p_{\theta_d^{\text{T}}}(a_j|\mathcal{G}_i)\left(\log p_{\theta_d^{\text{T}}}(a_j|\mathcal{G}_i) - \log p_{\theta^{\text{S}}}(a_j|\mathcal{G}_i)\right). \tag{7}$$

We summarize our AMDKD in Algorithm 1. Note that our AMDKD follows the on-policy scheme where the tours $\tau_{\theta^{\text{S}}}$ are output by the student that currently performs learning. As an alternative, such tours can also be output by the teacher (i.e., off-policy scheme). However, it performs inferior to our on-policy one (see Section 5.3). Finally, we note that the student learned by our AMDKD could also be coupled with the *efficient active search* (EAS) [21] during inference to further boost the performance. Accordingly, the resulting AMDKD+EAS becomes a new state-of-the-art solver.

# 5 Experiments

We conduct experiments on TSP and CVRP with $n = 20$, 50, and 100 nodes following the convention [13, 15]. As aforementioned, we adopt Uniform, Cluster and Mixed (mixture of uniform and cluster) as exemplar distributions for training; Expansion, Implosion, Explosion, and Grid as the unseen distributions for testing. We also test the methods on instances from TSPLIB [33] and CVRPLIB [46]. For the above 7 distributions, we follow [11, 20, 34] to generate the respective instances (details are presented in Appendix A). All experiments are conducted on a machine with NVIDIA GeForce RTX 3090 GPU cards and Intel Xeon Silver 4216 CPU at 2.10GHz. Our implementation code in PyTorch and the used data will be made publicly available.

**Training.** We apply our AMDKD to two representative deep models, i.e., AM [11] and POMO [13], termed as AMDKD-AM and AMDKD-POMO, respectively. For both TSP and CVRP, we use a batch size $B$ = 512 for training[2] and hyper-parameters $T$ = 250, $E'$ = 500 for AMDKD-AM and $T$ = 20, $E'$

[2]For training AMDKD-POMO on TSP100 and CVRP100, we use $B$ = 128 due to GPU memory constraint.

Table 1: Distillation effectiveness of AMDKD on three exemplar distributions.

| | Model | Size (Mb) | $n = 20$ | | | | $n = 50$ | | | | $n = 100$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $G_U$ | $G_C$ | $G_M$ | Avg. | $G_U$ | $G_C$ | $G_M$ | Avg. | $G_U$ | $G_C$ | $G_M$ | Avg. |
| TSP | Gurobi | - | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | AM(U) | 0.68 | 0.09% | 0.26% | 0.19% | 0.18% | 0.59% | 2.24% | 1.36% | 1.39% | 2.10% | 7.49% | 4.06% | 4.55% |
| | AM(C) | 0.68 | 0.17% | 0.10% | 0.27% | 0.18% | 1.41% | 0.80% | 2.14% | 1.45% | 3.76% | 6.97% | 4.39% | 5.04% |
| | AM(M) | 0.68 | 0.15% | 0.16% | 0.13% | 0.15% | 1.19% | 1.71% | 0.87% | 1.26% | 3.08% | 5.65% | 2.55% | 3.76% |
| | AMDKD-AM | **0.26** | 0.02% | 0.06% | 0.05% | **0.04%** | 0.25% | 1.64% | 0.86% | **0.91%** | 1.21% | 5.63% | 3.55% | **3.46%** |
| | POMO(U) | 1.20 | 0.00% | 0.01% | 0.01% | 0.01% | 0.04% | 0.42% | 0.21% | 0.22% | 0.17% | 1.97% | 0.92% | 1.02% |
| | POMO(C) | 1.20 | 0.00% | 0.00% | 0.01% | 0.00% | 0.09% | 0.07% | 0.21% | 0.12% | 0.41% | 0.29% | 0.83% | 0.51% |
| | POMO(M) | 1.20 | 0.00% | 0.01% | 0.00% | 0.00% | 0.08% | 0.17% | 0.08% | 0.11% | 0.77% | 1.17% | 0.34% | 0.76% |
| | AMDKD-POMO | **0.49** | 0.00% | 0.00% | 0.00% | **0.00%** | 0.05% | 0.05% | 0.09% | **0.06%** | 0.36% | 0.37% | 0.43% | **0.39%** |
| CVRP | LKH | - | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | AM(U) | 0.68 | 1.98% | 1.99% | 1.98% | 1.98% | 2.53% | 4.33% | 2.99% | 3.28% | 3.10% | 9.87% | 4.57% | 5.85% |
| | AM(C) | 0.68 | 1.62% | 1.43% | 1.74% | 1.60% | 3.08% | 2.75% | 3.35% | 3.06% | 4.27% | 3.89% | 4.93% | 4.36% |
| | AM(M) | 0.68 | 2.09% | 2.19% | 2.05% | 2.11% | 2.74% | 3.17% | 2.31% | 2.74% | 3.95% | 6.26% | 3.41% | 4.54% |
| | AMDKD-AM | **0.26** | 0.53% | 0.59% | 0.64% | **0.59%** | 1.61% | 2.66% | 1.92% | **2.07%** | 2.08% | 5.06% | 3.01% | **3.38%** |
| | POMO(U) | 1.20 | 0.36% | 0.49% | 0.51% | 0.45% | 0.80% | 1.53% | 1.07% | 1.13% | 0.95% | 2.34% | 1.31% | 1.53% |
| | POMO(C) | 1.20 | 0.41% | 0.40% | 0.54% | 0.45% | 1.16% | 0.93% | 1.07% | 1.05% | 0.93% | 1.28% | 1.21% | 1.14% |
| | POMO(M) | 1.20 | 0.36% | 0.51% | 0.40% | 0.42% | 1.22% | 1.34% | 0.85% | 1.14% | 1.89% | 2.07% | 0.96% | 1.64% |
| | AMDKD-POMO | **0.49** | 0.35% | 0.40% | 0.41% | **0.39%** | 0.81% | 0.97% | 0.89% | **0.89%** | 1.06% | 1.36% | 0.99% | **1.13%** |

= 1 for AMDKD-POMO, respectively. The total numbers of needed epochs vary with the problem size. In specific, regarding AMDKD-AM, we use 2,000, 5,000, 10,000 for problem sizes 20, 50, 100, respectively (both TSP and CVRP); regarding AMDKD-POMO, we use 5,000 (TSP-20), 10,000 (CVRP-20,TSP-50,TSP-100), and 30,000 (CVRP-50, CVRP-100). The Adam optimizer is used with learning rate 1e-4. Training time also varies with the problem size. Taking CVRP-100 as an example, one epoch takes about 0.4 minutes for AMDKD-AM and 1.4 minutes for AMDKD-POMO.

**Inference.** For AMDKD-AM, it samples 1,280 solutions following the orginal design [11]; and for AMDKD-POMO, we adopt the greedy rollout with $\times 8$ augments for the best performance of POMO [13]. All our experiments are conducted on a dataset containing 10,000 test instances per considered distribution. Unless otherwise stated, we display the gaps between the average tour length of each method and the strong traditional solvers, i.e., Gurobi [5] for TSP and LKH [4] for CVRP.

### 5.1 Effectiveness studies of AMDKD

We first study the effectiveness of our AMDKD when applied to backbone model AM and POMO for TSP and CVRP, respectively. In Table 1, we display the gaps of the learned student and their respective teachers on unseen instances following three exemplar distributions (denoted as $G_U$, $G_C$, and $G_M$) and also report the parameter sizes of each model. We can see that when the teacher model is trained on a specific distribution, it does not generalize well on the other ones (especially for AM), resulting in poor overall performance (if referring to the average gaps). In contrast, our AMDKD could alleviate this issue effectively. While the learned student model by AMDKD reduces the size of the teacher model from 0.68 to 0.26 Mb (a 61.8% reduction) for AM and from 1.20 to 0.49 Mb (a 59.2% reduction) for POMO, it still exhibits improved overall performance for both TSP and CVRP on all the three sizes. Pertaining to AMDKD-AM, our AMDKD student not only significantly outperforms its three teachers for both TSP and CVRP, but also exhibits even lower gaps on the distribution where the teacher was previously trained in most cases. Pertaining to AMDKD-POMO, although POMO itself presents a better generalization, our AMDKD can still boost its overall performance with a much lighter student network for both TSP and CVRP. The above results validate that our AMDKD does not overfit to a single distribution, and successfully learns a lightweight yet generalist student model under the guidance of multiple teachers with expertise in different distributions.

### 5.2 Generalization studies of AMDKD

We now assess the cross-distribution generalization of our AMDKD. In addition to the two backbone models, we also compare with the following methods, i.e., **1) other deep models** including- 1.a) DACT [15], a state-of-the-art neural *improvement* model that learns to perform local search; 1.b) LCP [14] (TSP only), a hybrid two-stage method that combines *improvement* and *construction* strategy; and **2) other methods specialized for generalization** including- 2.a) HAC [16] (TSP only),

Table 2: Generalization on unseen in-distribution (ID) and out-of-distribution (OoD) instances.

| | Model | Size (Mb) | $n = 20$ | | | $n = 50$ | | | $n = 100$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $G_{ID}$ | $G_{OoD}$ | Time* | $G_{ID}$ | $G_{OoD}$ | Time* | $G_{ID}$ | $G_{OoD}$ | Time* |
| TSP | Gurobi | - | 0.00% | 0.00% | 0.01s (7s) | 0.00% | 0.00% | 0.08s (51s) | 0.00% | 0.00% | 0.7s (7.6m) |
| | HAC# | 0.68 | 0.11% | 0.07% | 0.03s (48s) | 1.05% | 0.57% | 0.09s (4m) | 4.68% | 2.97% | 0.19s (16m) |
| | LCP# | 2.03 | 0.11% | 0.03% | 1.2s (43m) | 6.70% | 0.99% | 1.5s (2.8h) | 32.70% | 8.24% | 2.4s (6.4h) |
| | DACT(T=1,280)# | 0.27 | 0.11% | 0.08% | 12s (2.1m) | 0.31% | 0.34% | 19s (7.2m) | 2.37% | 3.13% | 28s (23m) |
| | AM# | 0.68 | 0.18% | 0.10% | 0.03s (48s) | 1.48% | 0.74% | 0.09s (4m) | 4.15% | 2.84% | 0.19s (16m) |
| | AMDKD-AM | 0.26 | 0.04% | 0.02% | 0.03s (35s) | 0.91% | 0.37% | 0.08s (2.8m) | 3.46% | 1.87% | 0.17s (11m) |
| | POMO# | 1.20 | 0.00% | 0.00% | 0.03s (5s) | 0.07% | 0.05% | 0.09s (16s) | 0.45% | 0.39% | 0.13s (1.1m) |
| | AMDKD-POMO | 0.49 | 0.00% | 0.00% | 0.02s (4s) | 0.06% | 0.05% | 0.04s (11s) | 0.39% | 0.41% | 0.12s (50s) |
| | AMDKD+EAS† | 0.49 | 0.00% | 0.00% | 5s (4.5m) | 0.01% | 0.01% | 12s (28m) | 0.11% | 0.10% | 28s (2.3h) |
| CVRP | LKH3 | - | 0.00% | 0.00% | 7.7s (1.3h) | 0.00% | 0.00% | 31s (5.3h) | 0.00% | 0.00% | 56s (9.6h) |
| | DACT(T=1,280)# | 0.27 | 0.47% | 0.42% | 28s (4.3m) | 3.28% | 3.16% | 55s (14m) | 9.11% | 9.38% | 1.5m (34m) |
| | AM# | 0.68 | 2.00% | 1.98% | 0.05s (1.2m) | 3.39% | 2.84% | 0.12s (5m) | 5.42% | 4.24% | 0.26s (18m) |
| | AMDKD-AM | 0.26 | 0.59% | 0.55% | 0.05s (49s) | 2.07% | 1.76% | 0.11s (3.6m) | 3.38% | 2.79% | 0.23s (13m) |
| | POMO# | 1.20 | 0.42% | 0.39% | 0.05s (7.8s) | 0.92% | 0.94% | 0.10s (18s) | 1.14% | 1.21% | 0.19s (1.3m) |
| | AMDKD-POMO | 0.49 | 0.39% | 0.36% | 0.05s (6.1s) | 0.89% | 0.90% | 0.08s (15s) | 1.13% | 1.21% | 0.18s (1.1m) |
| | AMDKD+EAS† | 0.49 | -0.04% | -0.06% | 9s (7.8m) | 0.07% | 0.04% | 20s (37m) | -0.03% | -0.04% | 40s (3.3h) |

* We report the average time to solve one instance, and the total time to solve 10,000 instances in (·) with batch parallelism allowed (one GPU).
# The corresponding model is trained on a mixed training dataset that contains instances from all the three exemplar distributions.
† For EAS, we adopt its EAS-lay version for demonstration since it is the best for CVRP according to [21].

a fine-tuning framework for AM by leveraging instances with different hardness; 2.b) DROP [20], a *distributionally robust optimization* based method to enhance POMO; 2.c) GANCO [17], a framework that enhances AM by learning a *generative adversarial network* to produce hard-to-solve training instances; and 2.d) PSRO (a.k.a. LIH) [19], an *improvement* method with a *game theory* based policy space response oracle framework. More implementation details of them are provided in Appendix B. Note that for DROP, GANCO, and PSRO, we only compare the results on benchmark reported in their original papers (see Table 3), since their codes (for re-training) are not publicly available.

Given that baselines AM, POMO, DACT, and LCP were originally trained on the uniform distribution (resulting in poor generalization), we re-train the above models (with #) on a mixed dataset containing instances from the three exemplar distributions, to ensure a fair comparison . We also apply this mixed dataset as the initial training instances for HAC (with #). We assess the following metrics, 1) parameter size of each model; 2) average gap on unseen in-distribution instances, i.e., $G_{ID}$; 3) average gap on unseen out-of-distribution instances, i.e., $G_{OoD}$; 4) average time for solving an individual instance and total time required to solve 10,000 instances with parallelism using one GPU card[3].

As can be observed in Table 2, our AMDKD is able to achieve competitive results on both unseen in-distribution and out-of-distribution instances, and performs favourably against all baselines. Pertaining to TSP, our AMDKD-AM significantly outperforms the baseline AM# on all sizes in terms of both $G_{ID}$ and $G_{OoD}$ with higher inference speed. Meanwhile, it also consistently beats the recent baseline HAC# (designed to boost the generalization of AM) on all sizes in terms of both two gaps and the time efficiency. For the backbone model POMO, which suffers from less generalization concern, the baseline POMO# can already attain near-optimal solutions to TSP instances with a desirable generalization performance. Nevertheless, our AMDKD is able to achieve comparable (slightly better) generalization results with nearly half of the model size and higher inference speed, which is also generic to boost other deep models. Moreover, by reducing the network dimension from 128 (POMO#) to 64 (AMDKD-POMO), POMO# performs inferior to ours (see Appendix C). Besides, both our AMDKD-AM and AMDKD-POMO can significantly outperform *improvement* methods including DACT and LCP with lower gaps while exhibiting much faster inference, which verifies the significance of our methods for practical use. Pertaining to CVRP, similar patterns can be observed, where our AMDKD-AM and AMDKD-POMO consistently deliver favourable generalization performance against all the baselines including AM# and POMO#, and require less time to infer. Furthermore, we show that by coupling our AMDKD-POMO model with the recent *efficient active search* (EAS) [21], the resulting AMDKD+EAS (T=100) attains considerably superior performance that even surpasses LKH3 on CVRP-100 with much shorter time. Given the advances of active search for a particular instance to be inferred, AMDKD+EAS allows the model learned by

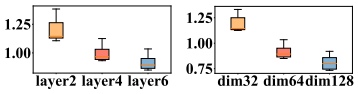
[3]Even so, we note that the time of traditional solvers and deep models might be still difficult to be fairly compared due to different implementations (C vs Python) and computing devices (CPU vs GPU).

Table 3: Generalization performance on selected instances ($100 \leq n \leq 200$) from benchmark datasets.

| | PSRO | AM | GANCO | HAC | AM# | **AMDKD-AM** | POMO | DROP | POMO# | **AMDKD-POMO** | **AMDKD+EAS** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TSPLIB | 4.47% | 42.63% | 4.87% | 6.06% | 41.75% | **3.53%** | 29.73% | 10.79% | 1.13% | **1.06%** | **0.73%** |
| CVRPLIB | - | 29.36% | - | - | 13.88% | **7.43%** | 14.19% | 8.67% | 6.80% | **4.38%** | **1.22%** |

Table 4: Ablation studies of AMDKD designs.

| Model | Components $p^{\text{adaptive}}$ | $\mathcal{L}_{\text{KD}}$ | $\mathcal{L}_{\text{Task}}$ | $\tau_\theta$s | CVRP-50 Avg. Gap | TSP-50 Avg. Gap |
|---|---|---|---|---|---|---|
| *w/o* adaptive | | ✓ | ✓ | ✓ | 0.94% | 0.69‰ |
| *w/o* $\mathcal{L}_{\text{KD}}$ | ✓ | | ✓ | ✓ | 1.06% | 0.66‰ |
| *w/o* $\mathcal{L}_{\text{Task}}$ | ✓ | ✓ | | ✓ | 1.00% | 0.69‰ |
| off-policy | ✓ | ✓ | ✓ | | 10.42% | 0.80‰ |
| AMDKD | ✓ | ✓ | ✓ | ✓ | **0.89%** | **0.63‰** |




(a) (b)

Figure 3: Ablation studies of student network. (a) effects of different layers, (b) effects of different embedding dimensions.

our AMDKD with strong generalization to continuously improve its performance during inference (longer T will yield even better performance), which leads to a new state-of-the-art hybrid solver.

We now evaluate the generalization of our AMDKD on the benchmark, i.e., TSPLIB and CVRPLIB, which contain various instances of unknown distributions and larger size. As shown in Table 3, our AMDKD-AM significantly boosts the generalization of AM and also outperforms the baseline AM# as well as other generalization methods that take AM as the backbone model (GANCO and HAC). Similarly, our AMDKD-POMO also significantly beats the baseline methods including POMO, POMO# and DROP. Notably, the improvements over POMO# is more clear on these benchmark instances following various unknown distribution (when compared with Table 2), especially on the harder CVRPLIB instances. Meanwhile, we note that the AMDKD+EAS achieves the smallest gaps for both TSP and CVRP among all baselines. We refer to Appendix D for full results and discussions.

### 5.3 Ablation studies of AMDKD

**Effects of different components.** We conduct ablation studies on CVRP-50 and TSP-50 to verify the designs of our AMDKD, where we consider removing the proposed adaptive strategy, the KD loss $\mathcal{L}_{\text{KD}}$, and the task loss $\mathcal{L}_{\text{Task}}$, respectively. As displayed in Table 4, the use of adaptive strategy further enhances the learning effectiveness, and the two losses play a prominent role in the phase of gradient update. We also verify that when compared with our on-policy scheme, the off-policy variant (as mentioned in Section 4.2) saliently impairs the distillation performance.

**Effects of student network architectures.** In Figure 3, we exhibit the boxplots of the averaged gaps when using different student network architectures (different dimensions of the entire network and different number of layers in its encoder) of AMDKD-POMO for solving CVRP-50. As revealed, the larger the model, the better its performance. Hence, we acknowledge that our AMDKD can be further boosted when the student (64) shares the same architecture with its teachers (128).

## 6 Conclusions and future work

In this paper, we propose an Adaptive Multi-Distribution Knowledge Distillation (AMDKD) scheme to alleviate the cross-distribution generalization issue of deep models for VRPs. Different from the existing generalization methods, we aim to enhance the distribution generalization by transferring various policies learned from exemplar distributions into one via an efficient knowledge distillation scheme. To facilitate effective learning, we design an adaptive strategy to train a single yet generalist student network by leveraging multiple teachers in turns. The experiment results exhibit competitive performance of our AMDKD in generalizing to other unseen out-of-distribution instances (randomly generated or from benchmarks), which also consumes less computational resources. While our AMDKD is generic, a potential limitation is that its boost is not guaranteed to be always significant across all unseen distributions for all backbone models. In future, we will investigate, 1) generalizing AMDKD to larger problem sizes; 2) considering the *improvement* models like DACT [15] as the backbone (though the used EAS in our AMDKD is also a type of *improvement* scheme); and 3) performing online distillation to jointly and efficiently train the teachers and the student models [47].

# References


[1] Lu Duan, Yang Zhan, Haoyuan Hu, Yu Gong, Jiangwen Wei, Xiaodong Zhang, and Yinghui Xu. Efficiently solving the practical vehicle routing problem: A novel joint learning approach. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3054–3063, 2020.

[2] Grigorios D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational research*, pages 1–30, 2020.

[3] Zhiwei Qin, Xiaocheng Tang, Yan Jiao, Fan Zhang, Zhe Xu, Hongtu Zhu, and Jieping Ye. Ride-hailing order dispatching at DiDi via reinforcement learning. *INFORMS Journal on Applied Analytics*, 50(5):272–286, 2020.

[4] Keld Helsgaun. LKH-3 (version 3.0.7), 2017. URL http://webhotel4.ruc.dk/~keld/research/LKH-3/.

[5] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

[6] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6: 791–812, 1958.

[7] Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems, 1997.

[8] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700, 2015.

[9] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Machine Learning (Workshop)*, 2017.

[10] Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V Snyder. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9861–9871, 2018.

[11] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[12] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. arxiv preprint arxiv:1906.01227, ArXiV, 2019.

[13] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198, 2020.

[14] Minsu Kim, Jinkyoo Park, and joungho kim. Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems*, volume 34, pages 10418–10430, 2021.

[15] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In *Advances in Neural Information Processing Systems*, volume 34, pages 11096–11107, 2021.

[16] Zeyang Zhang, Ziwei Zhang, Xin Wang, and Wenwu Zhu. Learning to solve travelling salesman problem with hardness-adaptive curriculum. In *AAAI Conference on Artificial Intelligence*, 2022.

[17] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Generative adversarial training for neural combinatorial optimization models, 2022. URL https://openreview.net/forum?id=9vsRT9mc7U.

[18] Simon Geisler, Johanna Sommer, Jan Schuchardt, Aleksandar Bojchevski, and Stephan Günnemann. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *International Conference on Learning Representations*, 2022.

[19] Chenguang Wang, Yaodong Yang, Congying Han, Tiande Guo, Haifeng Zhang, and Jun Wang. A game-theoretic approach for improving generalization ability of TSP solvers, 2022. URL https://openreview.net/forum?id=7AssAnH5vyJ.

[20] Yuan Jiang, Yaoxin Wu, Zhiguang Cao, and J. Zhang. Learning to solve routing problems via distributionally robust optimization. In *AAAI Conference on Artificial Intelligence*, 2022.

[21] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 6000–6010, 2017.

[23] Bo Peng, Jiahai Wang, and Zizhen Zhang. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *International Symposium on Intelligence Computation and Applications*, pages 636–650, 2019.

[24] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *AAAI Conference on Artificial Intelligence*, pages 12042–12049, 2021.

[25] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[26] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6351–6361, 2017.

[27] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *AAAI Conference on Artificial Intelligence*, 2021.

[28] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. arxiv preprint arxiv:2102.11756, ArXiV, 2021.

[29] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 32, pages 6281–6292, 2019.

[30] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *European Conference on Artificial Intelligence*, 2020.

[31] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[32] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2019.

[33] Gerhard Reinelt. TSPLIB-A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[34] Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. Evolving diverse tsp instances by means of novel and creative mutation operators. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 58–71, 2019.

[35] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arxiv preprint arxiv:1503.02531, ArXiV, 2015.

[36] Chuhan Wu, Fangzhao Wu, and Yongfeng Huang. One teacher is enough? pre-trained language model distillation from multiple teachers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4408–4413, 2021.

[37] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, 2020.

[38] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. FastBERT: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044, 2020.

[39] Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A Alemi, and Andrew G Wilson. Does knowledge distillation really work? In *Advances in Neural Information Processing Systems*, volume 34, pages 6906–6919, 2021.

[40] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1607–1616, 10–15 Jul 2018.

[41] Hossein Mobahi, Mehrdad Farajtabar, and Peter Bartlett. Self-distillation amplifies regularization in hilbert space. In *Advances in Neural Information Processing Systems*, volume 33, pages 3351–3361, 2020.

[42] Minghao Hu, Yuxing Peng, Furu Wei, Zhen Huang, Dongsheng Li, Nan Yang, and Ming Zhou. Attention-guided answer distillation for machine reading comprehension. In *EMNLP*, 2018.

[43] Glen Berseth, Cheng Xie, Paul Cernek, and Michiel Van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. In *International Conference on Learning Representations*, 2018.

[44] Andrei A Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *ICLR (Poster)*, 2016.

[45] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.

[46] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[47] Qiushan Guo, Xinjiang Wang, Yichao Wu, Zhipeng Yu, Ding Liang, Xiaolin Hu, and Ping Luo. Online knowledge distillation via collaborative learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11020–11029, 2020.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We provide the data generation instructions in Appendix A. Our code in PyTorch together with pre-trained models will be released on Github soon.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Appendix C.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Yes, see [4, 5, 11, 13–17, 19–21, 33, 34, 46].
   (b) Did you mention the license of the assets? [Yes] See Appendix E. We will use the MIT license (open-source license) for releasing our code and data.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use randomly generated data.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We use randomly generated data.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]