

Appendix

A Related Works

Traditional methods for MOCOP. MOCOPs are significantly harder to solve than their single-objective counterparts. Classic cases such as the Traveling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP) are already NP-hard, and adding multiple objectives only magnifies the difficulty [15]. Exact algorithms quickly become impractical for large instances or for problems that possess a vast Pareto set [17, 58]. Consequently, research has shifted toward heuristic methods that deliver high-quality Pareto fronts within reasonable time limits [63, 24]. Among these, multi-objective evolutionary algorithms (MOEAs) are widely used. They fall into two main categories: dominance-based MOEAs [11, 10, 46] and decomposition-based MOEAs [64, 14, 25]. Despite their popularity, MOEAs typically demand extensive manual design: practitioners must select and tune crossover, mutation, and selection operators along with many hyperparameters [62, 60, 50], and this labor-intensive hand-engineering often limits overall performance.

Neural methods for MOCOP. Inspired by the success of DRL in SOCOPs, recent research extends neural methods to MOCOPs by solving a series of scalarized SOCOPs. These neural solvers generally follow two paradigms: *one-to-one* and *one-to-many*. The one-to-one paradigm trains or fine-tunes a separate neural model for each scalarized subproblem. Some approaches apply DRL algorithms individually and transfer parameters across models to accelerate convergence [57, 33]. Others adopt architectures such as Pointer Networks [52] and Attention Models [28, 29], and optimize them using evolutionary strategies [47, 65]. To promote generalization across subproblems, meta-learning techniques are introduced [66]. However, this paradigm suffers from high training overhead and the burden of maintaining multiple models. In contrast, the one-to-many paradigm employs a single neural network to handle all subproblems. This includes hypernetwork-based DRL frameworks that condition on weight vectors [38, 34], and conditional neural heuristics that incorporate instance features, weight information, and problem size [13, 5].

Preference optimization. Preference optimization methods, such as Direct Preference Optimization (DPO) [44] and Identity Preference Optimization (IPO) [2], have gained significant traction, particularly in large language model (LLM) training, due to their ability to directly optimize human preferences without relying on explicit reward modeling as required in reinforcement learning from human feedback (RLHF). These methods are typically designed for pairwise preference data, where human annotators identify a preferred output over a less preferred one in response to a given prompt. Another line of research explores simpler preference optimization objectives that do not depend on a reference model [20, 61]. Among them, SimPO [41] proposes using the average log-probability of a generated sequence as an implicit reward. This approach aligns more closely with the model’s generation process and improves computational and memory efficiency by eliminating the need for a separate reference model. Some studies [36, 37] have explored applying preference optimization to SOCOPs, but its application to MOCOPs has been rarely investigated.

B Details of MOCOP

Here we elaborate on the problem definitions for the three typical MOCOPs, i.e., MOTSP, MOCVRP and MOKP.

MOTSP. An MOTSP instance involves multiple cost matrices, aiming to identify a set of tours, i.e., node sequences, that are Pareto optimal. For instance, a κ -objective TSP instance \mathcal{G} with $n + 1$ nodes is featured by cost matrices $C^i = (c_{j,k}^i)$, with $i \in \{1, \dots, \kappa\}$ and $j, k \in \{0, \dots, n\}$. The κ objectives are defined as follows,

$$\begin{aligned} \min_{\pi \in \mathcal{X}} F(\pi) &= \min(f_1(\pi), f_2(\pi), \dots, f_\kappa(\pi)), \\ \text{with } f_i(\pi) &= c_{\pi_n \pi_0}^i + \sum_{j=0}^{n-1} c_{\pi_j \pi_{j+1}}^i, \end{aligned} \tag{6}$$

where $\pi = (\pi_0, \pi_2, \dots, \pi_n)$ with $\pi_j \in \{0, \dots, n\}$. \mathcal{X} represents all feasible solutions (i.e., tours), ensuring each node is visited exactly once. This paper considers the Euclidean MOTSP following [15, 33, 38]. Each node j has a 2κ -dim feature vector $o_j = [loc_j^1, loc_j^2, \dots, loc_j^\kappa]$, where $loc_j^i \in \mathbb{R}^2$ is the coordinate for the i -th objective. The objective $f_i(\pi)$ is defined as $f_i(\pi) = \|loc_{\pi_n}^i - loc_{\pi_0}^i\|_2 + \sum_{j=0}^{n-1} \|loc_{\pi_j}^i - loc_{\pi_{j+1}}^i\|_2$.

MOCVRP. An MOCVRP instance consists of n customer nodes and one depot node. Each node has a 3-dim feature vector $o_j = [loc_j, \delta_j]$, where loc_j and δ_j , for $j \in \{0, \dots, n\}$, correspond to the coordinates and demand of node j . Notably, for the depot node, $o_0 = [loc_0, \delta_0]$, the demand δ_0 is set to 0. Vehicles with a capacity of Q ($Q > \delta_i$) are employed to serve all customers in multiple routes, with each route commencing and concluding at the depot. The problem must satisfy the following constraints: 1) Each customer is visited exactly once, and 2) The total demand of customers in each route must not exceed the vehicle's capacity. In our study, we focus on the bi-objective CVRP, aligning with prior research [30, 38]. Specifically, we aim to minimize two objectives: the total length of all routes and the length of the longest route (i.e., the makespan).

MOKP. An MOKP instance consists of $n + 1$ items, where each item j is characterized by a 2-dim feature vector $o_j = [w_j, p_j]$, representing its weight w_j and profit vector p_j , for $j \in \{0, \dots, n\}$. The profit vector $p_j \in \mathbb{R}^\kappa$ corresponds to κ distinct objective values associated with item j . A knapsack with a capacity of C is provided, where $C > w_j$ for each individual item. The goal is to select a subset of items to place into the knapsack while satisfying the following constraint: the total weight of selected items must not exceed the knapsack capacity C . In our study, we focus on the bi-objective knapsack problem, consistent with previous research [40, 67]. Specifically, we aim to simultaneously maximize two objectives: the sum of the first and second profit components across the selected items.

C MOCOP Solvers with POCCO

Existing neural MOCOP solvers are based on a transformer-based architecture that consists of an encoder and a decoder. The encoder is used to generate embeddings for all nodes based on the instance \mathcal{G} and the weight vector λ . The decoder is used to decode a sequence of actions based on these embeddings in an iterative fashion. To demonstrate the versatility of our POCCO, we integrate it into two SOTA neural methods, WE-CA [5] and CNH [13], yielding POCCO-W and POCCO-C.

C.1 POCCO-W

Given an instance \mathcal{G} comprising $n + 1$ nodes with Z -dimensional features $\{o_i\}_{i=0}^n \subset \mathbb{R}^Z$ and a weight vector λ , first obtains initial embeddings by applying separate linear projections to the node features and the weight vector:

$$h_i^0 = W^o o_i + b^o, \quad \forall i \in \{0, \dots, n\}, \quad h_\lambda^0 = W^\lambda \lambda + b^\lambda, \quad (7)$$

where $W^o \in \mathbb{R}^{d \times Z}$, $W^\lambda \in \mathbb{R}^{d \times \kappa}$, and $b^o, b^\lambda \in \mathbb{R}^d$ are learnable parameters, with embedding dimension $d = 128$. POCCO-W integrates the weight embedding into each node embedding in a feature-wise fashion within the encoder. To ensure harmonious interaction, the weight and node embeddings are updated simultaneously. The encoder itself comprises $L = 6$ transformer layers, each layer applying a conditional attention sublayer, followed by a residual Add & Norm (skip connection with instance normalization), a fully connected feed-forward sublayer, and a second Add & Norm.

Specifically, the conditional attention sublayer first condition node embeddings on the weight embedding via a feature-wise affine transform:

$$\gamma = W^\gamma h_\lambda^{l-1}, \quad \beta = W^\beta h_\lambda^{l-1}, \quad h'_i = \gamma \circ h_i^{l-1} + \beta, \quad \forall i \in \{0, \dots, n\}, \quad (8)$$

where W^γ and W^β are trainable matrices; \circ denotes element-wise multiplication. Then, the weight and node embeddings are updated via the Multi-Head Attention (MHA) mechanism with 8 heads and an Add & Norm, as follows:

$$\hat{h}_\lambda = \text{IN}(h_\lambda^{l-1} + \text{MHA}(h_\lambda^{l-1}, \{h_\lambda^{l-1}, h'_0, \dots, h'_n\})), \quad (9)$$

$$\hat{h}_i = \text{IN}(h_i^{l-1} + \text{MHA}(h'_i, \{h_\lambda^{l-1}, h'_0, \dots, h'_n\})), \quad \forall i \in \{0, \dots, n\}. \quad (10)$$

900 Afterwards, a fully connected feed-forward sublayer and another Add & Norm are employed to yield
 901 the weight embedding h_λ^l and the node embeddings $\{h_i^l\}_{i=0}^n$, as follows:

$$h_i^l = \text{IN}(\hat{h}_i + \text{FF}(\hat{h}_i)), \quad (11)$$

$$h_\lambda^l = \text{IN}(\hat{h}_\lambda + \text{FF}(\hat{h}_\lambda)), \quad (12)$$

902 Given the eventual node embeddings $\{h_i\}_{i=0}^n$ and weight embedding h_λ output by the encoder, the
 903 decoder autoregressively computes the probability of node selection over T steps. At decoding step
 904 $t \in \{1, \dots, T\}$, the advanced context vector h_c is produced by a MHA layer with 8 heads based on
 905 the context embedding v_c and eventual node embeddings as follows:

$$h_c = \text{MHA}(v_c, \{h_\lambda, h_0, \dots, h_n\}), \quad (13)$$

906 **Context embedding.** For MOTSP, the context embedding v_c is obtained by concatenating the
 907 embeddings of the first and last visited nodes, and all previously visited nodes are masked when
 908 computing selection probabilities. In MOCVRP, v_c consists of the embedding of the last visited node
 909 together with the remaining vehicle capacity, while nodes that have already been visited or whose
 910 demand exceeds the remaining capacity are masked. In MOKP, the context embedding combines
 911 the graph embedding $\hat{h} = \frac{1}{n+1} \sum_{i=0}^n h_i$ with the remaining knapsack capacity, masking both items
 912 that are already selected and those whose weight exceeds the remaining capacity when computing
 913 selection probabilities.

914 The context vector h_c is fed through the CCO block to produce a glimpse g_c based on Eq. 2 (i.e.,
 915 $g_c = \text{CCO}(h_c)$). This glimpse g_c is then used in a compatibility layer to compute unnormalized
 916 compatibility scores α as follows:

$$\alpha_i = \begin{cases} -\infty, & \text{if node } i \text{ is masked,} \\ C \cdot \tanh\left(\frac{g_c^\top (W^K h_i)}{\sqrt{d}}\right), & \text{otherwise,} \end{cases} \quad (14)$$

917 where C is set to 50 and W^K is a learnable weight matrix. Finally, the node-selection probability for
 918 the scalarized subproblem is given by:

$$P_\theta(\pi_t \mid \pi_{1:t-1}, s) = \text{Softmax}(\alpha). \quad (15)$$

919 C.2 POCCO-C

920 POCCO-C mirrors the overall design of POCCO-W but (i) replaces every conditional-attention
 921 sublayer in the encoder with a dual-attention sublayer and (ii) enriches the context vector h_c via a
 922 problem-size embedding (PSE) layer. Specifically, each of the L encoder layers in POCCO-C consists
 923 of a dual-attention sublayer, followed by a residual Add & Norm, a fully connected feed-forward
 924 sublayer, and a second Add & Norm. Concretely, at layer l the dual-attention and first Add & Norm
 925 operate as:

$$\hat{h}_\lambda = \text{IN}(h_\lambda^{l-1} + \text{MHA}(h_\lambda^{l-1}, \{h_\lambda^{l-1}, h_0^{l-1}, \dots, h_n^{l-1}\})), \quad (16)$$

$$\hat{h}_i' = \text{MHA}(h_i^{l-1}, \{h_\lambda^{l-1}, h_0^{l-1}, \dots, h_n^{l-1}\}) + \text{MHA}(h_i^{l-1}, \{h_\lambda^{l-1}\}), \quad \forall i \in \{0, \dots, n\}. \quad (17)$$

$$\hat{h}_i = \text{IN}(h_i^{l-1} + \hat{h}_i'), \quad \forall i \in \{0, \dots, n\}. \quad (18)$$

926 Besides, POCCO-C employ the sinusoidal encoding based on sine and cosine functions to yield the
 927 PSE as follows,

$$\begin{aligned} \text{PSE}(\xi, 2i) &= \sin(\xi/10000^{2i/d}), \\ \text{PSE}(\xi, 2i+1) &= \cos(\xi/10000^{2i/d}), \end{aligned} \quad (19)$$

928 where $\xi(\xi = n+1)$ and $i(i \in \{0, \dots, 63\})$ mean the problem size and dimension. The resulting
 929 d-dimensional PSEs are then processed using two linear layers with trainable matrices $W_{\xi 1} \in \mathbb{R}^{d \times 2d}$
 930 and $W_{\xi 2} \in \mathbb{R}^{2d \times d}$. POCCO-C inject the size information by adding the results from PSE to $\{h_i\}_{i=0}^n$
 931 from the encoder such that,

$$h_i^\xi = h_i + h_\xi, \text{ with } h_\xi = (\text{PSE}(\xi, \cdot)W_{\xi 1})W_{\xi 2}. \quad (20)$$

Algorithm 1 Preference-driven MOCO

Input: Instance distribution $\tilde{\mathcal{G}}$, weight vector distribution $\tilde{\lambda}$, number of training steps E , batch size B , number of tours K per subproblem;

Output: The trained policy network θ ;

```

1: Initialize policy network  $\theta$ .
2: for  $e = 1$  to  $E$  do
3:    $\lambda_b \sim \text{SAMPLEWEIGHTVECTOR}(\tilde{\lambda}); \mathcal{G}_b \sim \text{SAMPLEINSTANCE}(\tilde{\mathcal{G}}), \forall b \in \{1, \dots, B\}$ 
4:    $\pi^{j,b} \sim \text{SAMPLESOLUTIONS}(p_\theta(\cdot | \mathcal{G}_b, \lambda_b)), \forall j \in \{1, \dots, K\}, \forall b \in \{1, \dots, B\}$ 
5:    $y_{j,p}^b \sim \text{PAIRWISEPREFERENCE}(1_{[\pi^{j,b} \prec \pi^{p,b}]}), \forall j, p \in \{1, \dots, K\}, \forall b \in \{1, \dots, B\}$ 
6:   Calculate gradient  $\nabla_\theta \mathcal{L}(\theta)$  according to Eq. (5)
7:    $\theta \leftarrow \text{ADAM}(\theta, \nabla_\theta \mathcal{L}(\theta))$ 
8: end for
  
```

932 Then, POCCO-C produces the advanced context vector h_c based on the context embedding v_c and
 933 the size-injected node embeddings $\{h_i^\xi\}_{i=0}^n$ through a MHA layer with 8 heads as follows,

$$h_c = \text{MHA}\left(v_c, \{h_0^\xi, \dots, h_n^\xi\}\right). \quad (21)$$

934 C.3 Gating Mechanism

935 We employ subproblem-level gating, routing each context vector h_c independently to a subset of
 936 experts. Let d be the hidden dimension and $W_G \in \mathbb{R}^{d \times (m+1)}$ the trainable gating weights in POCCO.
 937 Given a batch of context vectors $X = \{h_c^b\}_{b=1}^B, X \in \mathbb{R}^{B \times d}$, where B is batch size, the gating
 938 network computes the score matrix $H = X W_G \in \mathbb{R}^{B \times (m+1)}$. Subproblem b (with context vector
 939 h_c^b) is routed to expert E_j according to the i -th row of H . Each subproblem selects the top k experts
 940 by score. For example, $k = 2$ routing sends each subproblem to the two highest-scoring experts.

941 D Training Algorithm

942 The training algorithm is provided in Algorithm 1. To train the model with preference learn-
 943 ing, we first sample a batch of instances and weight vectors $\{(\mathcal{G}_b, \lambda_b)\}_{b=1}^B$ (as in Line 3). Then,
 944 for each scalarized subproblem $(\mathcal{G}_b, \lambda_b)$, we construct a set of winning-losing solution pairs
 945 $(\pi^{j,b}, \pi^{p,b}), \forall j, p \in \{1, \dots, K\}$ (as in Lines 4-5). Training then proceeds by maximizing the
 946 likelihood of the winning solutions while minimizing that of the losing ones (as in Lines 6-7).

947 E Decomposition Approaches

948 The major decomposition techniques include weighted-sum, Tchebycheff, and penalty-based bound-
 949 ary intersection (PBI) approaches [64, 42], respectively.

950 **Weighted-sum Approach.** Given an MOCOP, the i th subproblem (i.e., SOCOP) is defined with the
 951 i th weight vector λ_i , such that,

$$\min \quad g_w(\pi | \lambda_i) = \sum_{j=1}^{\kappa} \lambda_i^j f_j(\pi), \quad \pi \in \mathcal{X} \quad (22)$$

952 **Tchebycheff Approach.** It minimizes the maximal distance between objectives and the ideal
 953 reference point, such that,

$$\min \quad g_t(\pi | \lambda_i, z^*) = \max_{1 \leq j \leq \kappa} \left\{ \lambda_i^j |f_j(\pi) - z_j^*| \right\}, \quad \pi \in \mathcal{X} \quad (23)$$

954 where $z^* = (z_1^*, \dots, z_\kappa^*)^\top$ signifies the ideal reference point with $z_j^* \leq \min \{f_j(\pi) | \pi \in \mathcal{X}\}$. It
 955 guarantees that the optimal solution in Eq. (23) under a specific (but unknown) weight vector λ_i
 956 could be a Pareto optimal solution [8].

957 **PBI Approach.** This approach formulates the i th subproblem of an MOCOP as follows,

$$\begin{aligned} \min \quad & g_p(\pi|\lambda) = d_1 + \alpha d_2 \\ \text{where} \quad & d_1 = \frac{\|(F(\pi) - z^*)^\top \lambda\|}{\|\lambda\|} \\ & d_2 = \|F(\pi) - (z^* + d_1 \lambda)\|, \pi \in \mathcal{X} \end{aligned} \quad (24)$$

958 where $\alpha > 0$ is a preset penalty item and z^* is the ideal reference point as defined in the Tchebycheff
959 approach.

960 F Hypervolume

961 Hypervolume (HV) is a widely-used indicator to evaluate approximate Pareto solutions to MOCOPs.
962 Formally, the HV for a set of solutions \mathcal{P} is defined as the volume of the subspace, which is weakly
963 dominated by the solutions in \mathcal{P} and bounded by a reference point r^* , such that,

$$\text{HV}(\mathcal{P}) = \zeta^\kappa(\{r \in \mathbb{R}^\kappa \mid \exists \pi \in \mathcal{P}, \pi \prec r \prec r^*\}), \quad (25)$$

964 where ζ^κ denotes the Lebesgue measure on the κ -dimensional space, i.e., the volume for a m -
965 dimensional subspace [54]. Since the range of objective values varies among different problems, we
966 report the normalized HV $\bar{H}(\mathcal{P}) = \text{HV}(\mathcal{P}) / \prod_{i=1}^\kappa |r_i^* - z_i|$, where the ideal point $z = (z_1, \dots, z_\kappa)$
967 satisfies $z_i < \min\{f_i(\pi) \mid \pi \in \mathcal{P}\}$ (or $z_i > \max\{f_i(\pi) \mid \pi \in \mathcal{P}\}$ for maximization), $\forall i \in$
968 $\{1, \dots, \kappa\}$. All methods share the same r^* and z for a MOCOP, as given in Table 3, and we report
969 the average $\bar{H}(\mathcal{P})$ over all test instances in this paper.

Table 3: Reference points and ideal points.

Problem	Size	r^*	z
Bi-TSP	20	(20, 20)	(0, 0)
	50	(35, 35)	(0, 0)
	100	(65, 65)	(0, 0)
	150	(85, 85)	(0, 0)
	200	(115, 115)	(0, 0)
Bi-CVRP	20	(30, 4)	(0, 0)
	50	(45, 4)	(0, 0)
	100	(80, 4)	(0, 0)
Bi-KP	50	(5, 5)	(30, 30)
	100	(20, 20)	(50, 50)
	200	(30, 30)	(75, 75)
Tri-TSP	20	(20, 20, 20)	(0, 0)
	50	(35, 35, 35)	(0, 0)
	100	(65, 65, 65)	(0, 0)

970 G Instance Augmentation

971 To further improve the performance of POCCO at the inference stage, we apply the instance
972 augmentation proposed in [29], which is also used in PMOCO [38]. The rationale of in-
973 stance augmentation is that an instance of Euclidean VRPs can be transformed into different
974 ones, that share the same optimal solution, e.g., by flipping coordinates for all nodes in an in-
975 stance. Given a coordinate (x, y) in a VRP, there are eight simple transformations, i.e., $(x', y') =$
976 $(x, y); (y, x); (x, 1-y); (y, 1-x); (1-x, y); (1-y, x); (1-x, 1-y); (1-y, 1-x)$. In our paper, we
977 adopt these transformations for each objective, respectively. Hence, we could have 8 transformations
978 for Bi-CVRP (since there is only one coordinate for each node), $8^2 = 64$ transformations for Bi-TSP
979 and $8^3 = 512$ transformations for Tri-TSP, respectively.

Table 4: Performance on KroAB Instances

Method	KroAB100			KroAB150			KroAB200		
	HV	Gap	Time	HV	Gap	Time	HV	Gap	Time
WS-LKH	0.7022	-0.23%	2.3m	0.7017	-0.59%	4.0m	0.7430	-0.83%	5.6m
MOEA/D	0.6836	2.43%	5.8m	0.6710	3.81%	7.1m	0.7106	3.57%	7.3m
NSGA-II	0.6676	4.71%	7.0m	0.6552	6.08%	7.9m	0.7011	4.86%	8.4m
MOGLS	0.6817	2.70%	52m	0.6671	4.37%	1.3h	0.7083	3.88%	1.6h
PPLS/D-C	0.6785	3.15%	38m	0.6659	4.54%	1.4h	0.7100	3.65%	3.8h
DRL-MOA	0.6903	1.47%	10s	0.6794	2.61%	12s	0.7185	2.50%	18s
MDRL	0.6881	1.78%	9s	0.6831	2.08%	11s	0.7209	2.17%	16s
EMNH	0.6900	1.51%	9s	0.6832	2.06%	11s	0.7217	2.06%	16s
PMOCO	0.6878	1.83%	9s	0.6819	2.25%	12s	0.7193	2.39%	17s
CNH	0.6947	0.84%	16s	0.6892	1.20%	19s	0.7250	1.61%	22s
POCCO-C	0.6965	0.59%	30s	0.6925	0.73%	40s	0.7302	0.91%	50s
WE-CA	0.6948	0.83%	9s	0.6924	0.75%	12s	0.7317	0.71%	16s
POCCO-W	0.6981	0.36%	20s	0.6946	0.43%	31s	0.7345	0.33%	40s
MDRL-Aug	0.6950	0.80%	10s	0.6890	1.23%	16s	0.7261	1.47%	25s
EMNH-Aug	0.6958	0.69%	10s	0.6892	1.20%	16s	0.7270	1.34%	25s
PMOCO-Aug	0.6937	0.98%	11s	0.6886	1.29%	18s	0.7251	1.60%	30s
CNH-Aug	0.6980	0.37%	17s	0.6938	0.54%	26s	0.7303	0.90%	37s
POCCO-C-Aug	0.6999	0.10%	32s	0.6959	0.24%	48s	0.7334	0.47%	1.1m
WE-CA-Aug	0.6990	0.23%	10s	0.6957	0.27%	20s	0.7349	0.27%	31s
POCCO-W-Aug	<u>0.7006</u>	0.00%	22s	<u>0.6976</u>	0.00%	39s	<u>0.7369</u>	0.00%	59s

Table 5: Performance on Bi-TSP150 and Bi-TSP200 Instances

Method	Bi-TSP150			Bi-TSP200		
	HV	Gap	Time	HV	Gap	Time
WS-LKH	0.7149	-1.23%	13h	0.7490	-1.23%	22h
MOEA/D	0.6809	3.58%	2.4h	0.7139	3.51%	2.7h
NSGA-II	0.6659	5.71%	6.8h	0.7045	4.78%	6.9h
MOGLS	0.6768	4.16%	22h	0.7114	3.85%	38h
PPLS/D-C	0.6784	3.94%	21h	0.7106	3.96%	32h
DRL-MOA	0.6901	2.28%	36s	0.7219	2.43%	1.2m
MDRL	0.6922	1.98%	36s	0.7251	2.00%	1.1m
EMNH	0.6930	1.87%	37s	0.7260	1.88%	1.1m
PMOCO	0.6910	2.15%	42s	0.7231	2.27%	1.3m
CNH	0.6985	1.09%	50s	0.7292	1.45%	1.4m
POCCO-C	0.7011	0.72%	1.5m	0.7333	0.89%	2.5m
WE-CA	0.7008	0.76%	45s	0.7346	0.72%	1.3m
POCCO-W	0.7033	0.41%	1.4m	0.7371	0.38%	2.4m
MDRL-Aug	0.6976	1.22%	37m	0.7299	1.35%	1.1h
EMNH-Aug	0.6983	1.12%	39m	0.7307	1.24%	1.1h
PMOCO-Aug	0.6967	1.35%	40m	0.7283	1.57%	1.2h
CNH-Aug	0.7025	0.52%	41m	0.7343	0.76%	1.2h
POCCO-C-Aug	0.7043	0.27%	55m	0.7366	0.45%	1.5h
WE-CA-Aug	0.7044	0.25%	42m	0.7381	0.24%	1.2h
POCCO-W-Aug	<u>0.7062</u>	0.00%	45m	<u>0.7399</u>	0.00%	1.4h

H Detailed Results on Benchmark Instances

The detailed out-of-distribution generalization results are presented in Table 4 further confirming the exceptional generalization ability of our POCCO.

I Experimental Results on the Larger Problem Sizes

The results on Bi-TSP150/200, summarized in Table 5 show that POCCO-W consistently achieves the best generalization performance compared to all neural baselines and classical MOEAs.

986 J Hyperparameter Study

987 **Effects of the β .** Fig 5 shows how the temperature parameter β in the preference learning loss affects
 988 performance (HV) across four benchmark tasks. A moderate value of β consistently yields the best
 989 results. For the three bi-objective problems (Bi-TSP100, MOCVRP100, Bi-KP100) performance
 990 peaks around $\beta = 3.5$; larger or smaller values provide no additional benefit, and in Bi-KP100 an
 991 overly large $\beta = 5$ sharply degrades HV. For the tri-objective problem (Tri-TSP100), performance
 992 improves up to $\beta = 4.5$ and then declines slightly. Overall, these trends justify the default settings
 993 adopted in our experiments: $\beta = 3.5$ for bi-objective tasks and $\beta = 4.5$ for tri-objective tasks.

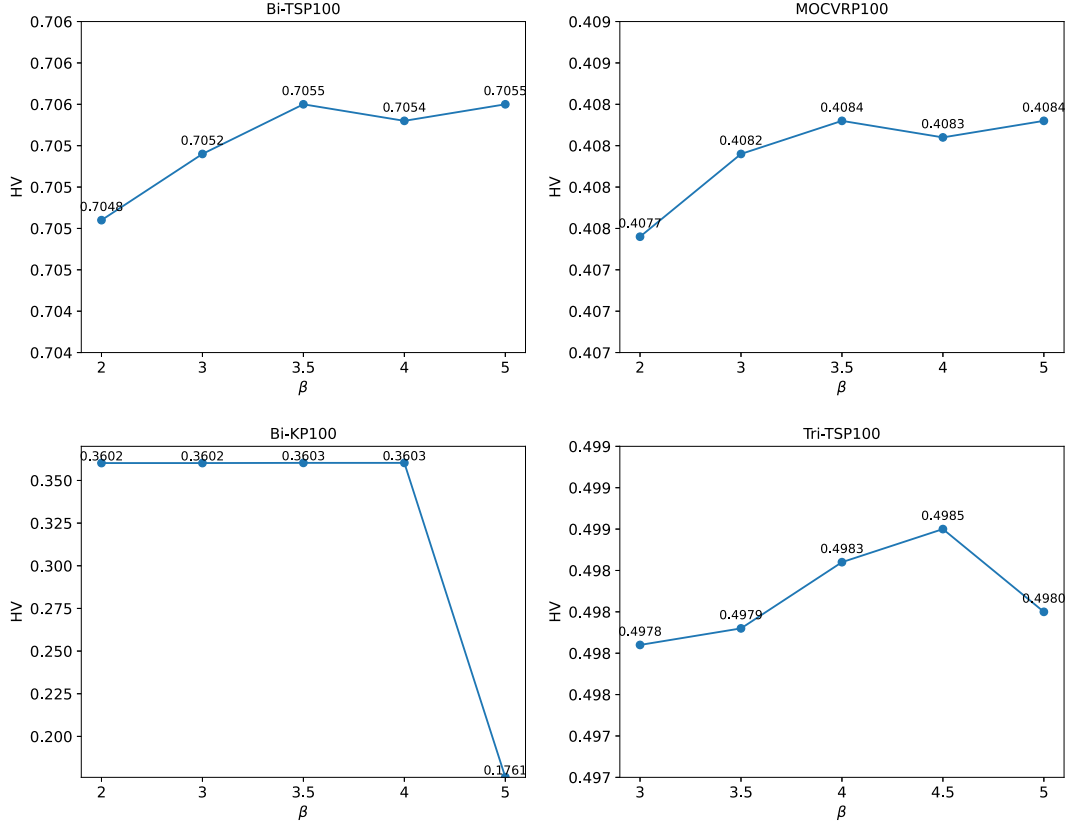


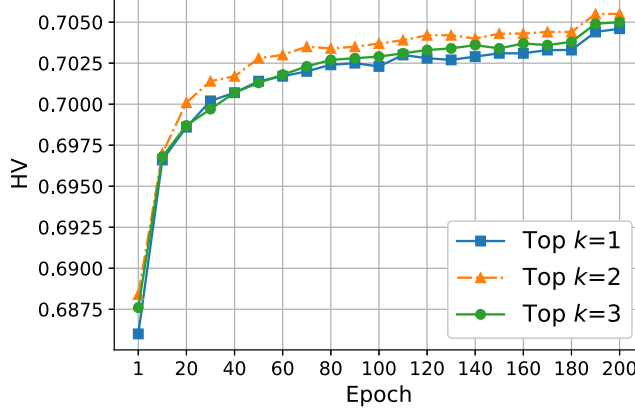
Figure 5: Effects of the β .

994 **Effects of the number of CCO layers.** The CCO block sits in the decoder, whose sequence length T
 995 grows with problem size. Each additional CCO layer therefore adds a full set of gating and expert
 996 computations at every decoding step. Hence, stacking too many CCO layers can inflate runtime
 997 disproportionately. Table 6 reports the trade-off. Using two CCO layers yields the highest HV on both
 998 Bi-TSP100 and Bi-TSP200, but increases inference time by roughly 60% and 130 %, respectively. A
 999 third layer further slows inference while slightly reducing HV. We therefore adopt a single-layer CCO
 1000 as the default, which preserves most of the performance gain while keeping computation modest.
 1001 For larger instances or latency-sensitive applications, the one-layer setting offers a favorable balance
 1002 between solution quality and speed.

1003 **Effects of Top k .** Fig 6 plots validation HV on Bi-TSP100 for $k \in \{1, 2, 3\}$. Selecting two experts
 1004 per token ($k=2$) converges fastest and attains the highest final HV. A single expert ($k=1$) limits model
 1005 capacity, leading to slower early progress and a slightly lower plateau. Choosing three experts ($k=3$)
 1006 increases computation relative to $k=2$ yet offers no benefit and even marginally reduces HV, likely
 1007 because the additional expert dilutes specialization and weakens sparsity. We therefore adopt $k=2$ as
 1008 the default, balancing performance and efficiency.

Table 6: Effects of the number of CCO block layers.

Method	Bi-TSP100		Bi-TSP200	
	HV	Time	HV	Time
POCCO-W	0.7055	36 s	0.7371	2.4 m
2 CCO layers	0.7058	59 s	0.7379	5.6 m
3 CCO layers	0.7049	85 s	0.7352	7.9 m

Figure 6: Effects of Top k .

1009 K Experimental Results of Scalarized Subproblems

1010 To assess subproblem optimality, we report scalarized objective values for three representative
 1011 weight vectors $\lambda = (1, 0)$, $(0.5, 0.5)$, $(0, 1)$ on Bi-TSP100 (Table 7). We also compare against
 1012 single-objective solvers LKH and POMO, each tuned to the corresponding subproblem. Among
 1013 neural MOCOP methods, POCCO-W attains the smallest optimality gaps across all weight settings.
 1014 Ablating CCO (POCCO-Aug w/o CCO) increases optimality gaps across all settings, and eliminating
 1015 preference learning as well (WE-CA-Aug) further degrades performance. Notably, POCCO-W-Aug
 1016 even outperforms POMO-Aug on the subproblem with $\lambda = (0, 1)$.

Table 7: Performance comparison under different weight settings (ω_1, ω_2) .

Method	$\lambda = (1, 0)$		$\lambda = (0.5, 0.5)$		$\lambda = (0, 1)$	
	Obj	Gap	Obj	Gap	Obj	Gap
WS-LKH	7.7632	0.00%	17.3094	0.00%	7.7413	0.00%
POMO-Aug	7.7659	0.03%	17.4421	0.77%	7.7716	0.39%
WE-CA-Aug	7.8132	0.64%	17.4994	1.10%	7.7888	0.61%
POCCO-Aug w/o CCO	7.7970	0.44%	17.4629	0.89%	7.7772	0.46%
POCCO-W-Aug	7.7827	0.25%	17.4460	0.79%	7.7602	0.24%

1017 L Summary of Decomposition-based Neural MOCOP Solvers

1018 Table 8 compares key features of decomposition-based neural MOCOP solvers. POCCO, which
 1019 establishes new SOTA performance, is a plug-and-play framework that augments any existing solver.
 1020 It inserts a CCO block that learns a diverse ensemble of policies, adding parameters yet surpassing
 1021 prior methods. Crucially, each subproblem activates only two experts (one of which may be a
 1022 parameter-free identity expert), so the extra computational load remains minimal. POCCO also
 1023 employs a pairwise preference learning approach that further boosts performance without introducing
 1024 additional parameters.

Table 8: Summary of the decomposition-based neural MOCOP solvers.

Method	Learning method	Paradigm	#Parameters
DRL-MOA	Transfer learning+RL	one-to-one	133.37M
MDRL	Meta learning+RL	one-to-one	133.37M
EMNH	Meta learning+RL	one-to-one	133.37M
PMOCO	RL	one-to-many	1.50M
CNH	RL	one-to-many	1.63M
POCCO-C	Preference learning	one-to-many	2.16M
WE-CA	RL	one-to-many	1.47M
POCCO-W	Preference learning	one-to-many	2.00M

1025 M Broader Impacts

1026 POCCO offers several positive societal impacts. By dynamically routing computation and learning
 1027 from preference signals, it accelerates multi-objective decision-making in logistics, manufacturing,
 1028 and energy planning, reducing costly trial-and-error loops and boosting operational efficiency. Its
 1029 conditional-computation design activates only the required network capacity for each subproblem,
 1030 cutting FLOP counts and energy consumption relative to dense models of comparable accuracy.
 1031 Finally, by encouraging exploration and adaptive capacity allocation, POCCO broadens the applica-
 1032 bility of neural solvers in complex optimization tasks, advancing both AI and operations research and
 1033 enabling practitioners to tackle larger, real-world problems with fewer computational resources.

1034 N Licenses for Existing Assets

1035 The used assets in this work are listed in Table 9, which are all open-source for academic research.
 1036 We will release our source code with the MIT License.

Table 9: Used assets, licenses, and their usage.

Type	Asset	License	Usage
Code	LKH [19]	Available for academic use	Evaluation
	DRL-MOA [33]	MIT License	Evaluation
	MDRL [66]	MIT License	Evaluation
	EMNH [6]	MIT License	Evaluation
	CNH [13]	MIT License	Revision
	WE-CA [5]	MIT License	Revision
Datasets	Chen et al. [5]	MIT License	Evaluation