CORectifier: Hierarchically Rectifying Policy Rollouts for Reinforcement Learning in Combinatorial Solvers

Wenzheng Pan[1†], **Nuoyan Chen**[1], **Jiaxi Liu**[1], **Jiale Ma**[12], **Junchi Yan**[12*]
[1]Sch. of Computer Science & Sch. of Artificial Intelligence, Shanghai Jiao Tong University
[2]Shanghai Innovation Institute
{pwz1121,cny123222,ecthelionliu,heatingma,yanjunchi}@sjtu.edu.cn

Abstract

In Neural Combinatorial Optimization (NCO), autoregressive Reinforcement Learning (RL) naturally preserves sequential feasibility but often suffers from terminal-only feedback and weak action-level credit assignment. We propose CORectifier, an online, feasibility-preserving, successor-following rollout-rectification mechanism for RL-based constructive solvers. During training, hierarchical batch-, trajectory-, and segment-level gates selectively replace policy actions with feasible successors from a limited set of reference trajectories, after which the policy continues from the rectified state. This Rectified RL (RRL) paradigm helps develop optimality-aware and sample-efficient RL learners while maintaining their sequential-decision manner for constraint satisfaction, delivering a new perspective to hybridize RL and supervised or imitation learning for NCO. Sufficiently extensive experiments on TSP, ATSP, PCTSP, CVRP, KP, JSSP and SMTWTP, across synthetic and real-world benchmarks, sparse and perturbed references, controlled mechanism ablations, independent training seeds, and zero-shot CVRP generalization up to 5,000 nodes show consistent improvements over the corresponding vanilla RL training baselines and competitive performance relative to broader neural solvers. Code is available at https://github.com/Thinklab-SJTU/CORectifier.

1 Introduction

Combinatorial Optimization (CO) [1] serves as the foundation for numerous real-world applications such as routing and scheduling [2–6], yet remains a core challenge owing to its inherent NP-hardness. In recent years, Neural Networks (NNs) have been actively employed to learn solving CO problems (COPs), giving rise to the growing NCO community [7, 8]. This integration yields practical advantages of both solution quality and computational efficiency [9], particularly when the tasks are specifically distributed, where deep models perform well in automatic discovery of effective heuristics from data.

Within the NCO landscape, supervised learning (SL)-based methods [10–12] represent the most straightforward paradigm. They directly generate neural predictions (e.g., probability heatmaps) that minimize the distance from reference solutions and then guide the solution construction or search processes. While SL-based methods exhibit satisfactory in-distribution performance, they face key limitations: beyond the need to acquire large amounts of supervised data via oracle solvers (Gurobi, Concorde, LKH, etc.), their indispensable reliance on handcrafted decoders [13–17] largely hampers applicability to COPs with more complex constraints (e.g., PCTSP, CVRP). In parallel, Reinforcement Learning (RL) for CO has emerged as a dominant paradigm [18–22], with main strengths that its sequential decision-making manner provides greater flexibility to enforce intricate constraints during solution construction, thereby enabling better extensibility to realistic scenarios.

*Correspondence to: Junchi Yan. †Project lead, who is also a visiting student at Shanghai Innovation Institute.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

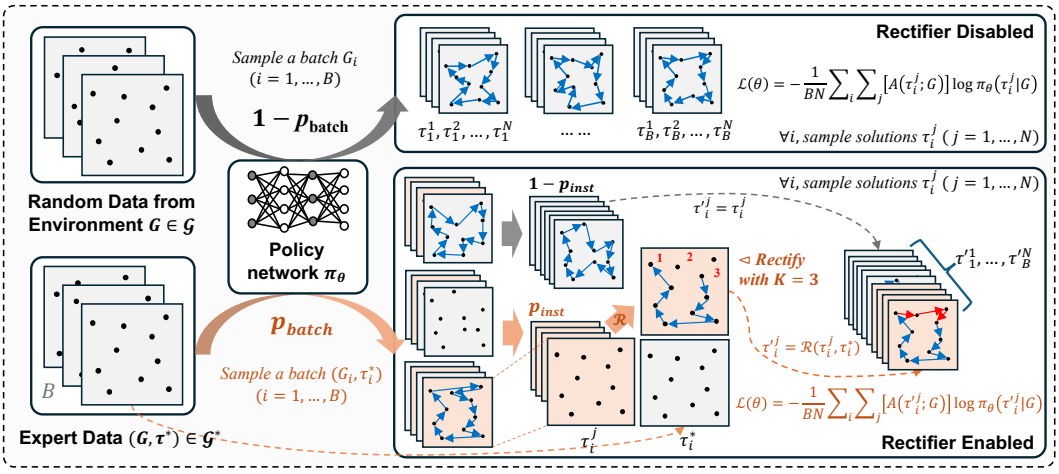


Figure 1: RRL stage of CORectifier. In general, B-sized batches are sampled from labeled data with prob. p_{batch}, and for each, N tours are sampled, of which a p_{inst} fraction undergo rectification if enabled. $A(\cdot)$: the advantage function; K: the (varying) rectified segment length.

While promising, early RL-based approaches face fundamental bottlenecks that severely constrain their practical utility: **1) Ineffective training due to sub-optimal exploration.** Once the model converges to a plateau of decent solution quality, it becomes extremely difficult to generate improved solutions via random exploration in vast combinatorial action spaces. Thus, the probability of sampling trajectories that yield meaningful reward gains (e.g., shorter TSP tours) drops sharply, which is critical as the RL policies evolve themselves upon the advantage metric of sampled solutions over a batch-wise average baseline [19]. **2) Inefficient sampling caused by sparse rewards.** Rewards for learning COPs are typically delayed, as they are only computed after an entire trajectory is generated. This phenomenon is analogous to autoregressive language models, where prediction quality deteriorates as the generated sequence extends farther from the initial context, causing cumulative errors and instability. **3) Poor scalability and limited evaluation scope.** As widely acknowledged in prior literature [16, 23, 24], the aforementioned limitations of RL-based CO solvers are exacerbated with increasing problem size, leaving them unable to handle graph-based COPs even with over 100 nodes. Consequently, experimental evaluations of these RL-based methods have conventionally been restricted to comparisons within the same RL family and on only quite small-scale instances [18, 20, 25, 26], resulting in an incomplete assessment against other NCO paradigms (heatmap-guided, unsupervised, generative solving, etc.) and larger scales, e.g., TSP/ATSP ≥ 200.

To resolve these challenges, drawing inspiration from successful practices in broader ML domains, e.g., interleaved Imitation Learning (IL) and RL in robotics [27, 28], and Teacher Forcing (TF) in sequential prediction [29–32], we propose a rectified reinforcement learning paradigm for discrete optimization that incorporates hierarchical supervised signals to regularize the random exploration for combinatorial policy learning while ensuring strict compliance with task-specific constraints. During training, CORectifier probabilistically replaces segments of the model's predicted trajectory with high-quality ones derived from reference solutions. Operating at a hierarchical granularity (flexibly controlled by hyper-parameters), expert-guided segments of varying lengths can therefore take effect at different solving statuses, avoiding the common short-sightedness of former constructive methods.

Specifically, the model is first optionally trained via pure IL for few steps, followed by the rectified RL stage (depicted in Fig. 1): local rectifiers commence their work once overfitting is observed in the IL phase. A small fraction of the sampled batches/instances fetch the rectified trajectories, whose higher-than-average rewards make them distinguishable, enabling the model to repeatedly escape local optima and be continuously guided toward exploring solution spaces with better quality. Unlike previous IL or SL attempts that used static reference trajectories or matrices, our proposed paradigm leverages reference data more efficiently at the intra-instance level, as a single training sample (e.g., an optimal tour for a TSP graph) can generate multiple arbitrary guiding segments, as opposed to vanilla SL where each solved training instance contributes value only as an entire unit, dramatically improving the data efficiency. Furthermore, as a general-purpose framework,

CORectifier can be orthogonally applied on top of existing RL-based methods to deliver consistent performance improvements, functioning as a plug-in learning mechanism. We shall also make the reusable reference datasets publicly available for future research to facilitate transparent comparison.

Major contributions of this paper: **1)** We propose a new learning paradigm for solving COPs with a novel rectification mechanism to resolve RL-methods' sample efficiency and reward sparsity. **2)** To our best knowledge, CORectifier is among the earliest attempts to explore the synergy between RL and SL/IL (as in Fig. 5) *within the NCO realm*, with hierarchical guiding signals operated at a triad of batch-wise, instance-wise, and intra-instance levels. **3)** We adapt our paradigm to three mainstream RL4CO models and achieve consistently enhanced training and solution quality across cases. **4)** We conduct extensive experiments on 7 classic COPs with different types and complexities (TSP, ATSP, PCTSP, CVRP, KP, SMTWTP, JSSP), showing up to **59.7%** and **26.5%** gains over RL/SL methods, and improving quality on TSP-500 (a *sufficiently* challenging scale for RL solvers) by up to **89.8%**.

2 Methodology

2.1 Preliminary: Reinforcement Learning for Combinatorial Optimization

General Problem Formulation on Graphs. Following established convention [10, 13, 33, 34], we focus on graph-based COPs where each instance $G = (\mathcal{V}, \mathcal{E}, w)$ is sampled from an instance distribution $\mathcal{D}_{\mathcal{G}}$. Here, $\mathcal{V} = \{1, \dots, M\}$ is the node set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, and $w : \mathcal{E} \to \mathbb{R}_{\geq 0}$ denotes edge weights. Specific definitions for all covered COPs are in Appendix A.1.

The goal of a CO solver is to find a feasible solution trajectory $\tau \in \Omega_G$, where Ω represents the feasible region defined by problem-specific constraints. For TSP, $\tau = [v_1, v_2, ..., v_M]$ is a permutation of $\mathcal{V}$, and the solver aims to minimize the objective function $c(\tau; G)$, e.g., for TSP, the total travel distance of the solved Hamiltonian cycle: $c(\tau; G) = \sum_{i=1}^{M-1} w(v_i, v_{i+1}) + w(v_M, v_1)$.

Learning a Policy. In RL-based NCO, a constructive solver is modeled as a stochastic policy $\pi_\theta(a_t \mid s_{t-1}; G)$ parameterized by θ, where $s_{t-1} = (G, \tau_{t-1})$ is the decision state before step t and $\tau_{t-1} = [a_1, \dots, a_{t-1}]$ is the partial trajectory generated so far. For TSP, $a_t \in \mathcal{V} \setminus \{a_1, \dots, a_{t-1}\}$ denotes the next unvisited node. Starting from the initial state s_0, the policy induces a distribution over feasible trajectories:

$$\pi_\theta(\tau \mid G) = \prod_{t=1}^{M'} \pi_\theta(a_t \mid s_{t-1}; G), \qquad \tau = [a_1, \dots, a_{M'}] \in \Omega_G. \tag{1}$$

Action masks ensure feasibility by assigning zero probability to invalid actions. RL maximizes $\mathcal{J}(\theta) = \mathbb{E}_{G \sim \mathcal{D}_{\mathcal{G}},\, \tau \sim \pi_\theta(\cdot|G)}[R(\tau; G)]$, where $\mathcal{D}_{\mathcal{G}}$ is the training distribution. By REINFORCE [35],

$$\begin{aligned} \nabla_\theta \mathcal{J}(\theta) &= \mathbb{E}_{G \sim \mathcal{D}_{\mathcal{G}},\, \tau \sim \pi_\theta(\cdot|G)}[A(\tau; G) \nabla_\theta \log \pi_\theta(\tau \mid G)] \\ &= \mathbb{E}_{G,\tau}\left[A(\tau; G) \sum_{t=1}^{M'} \nabla_\theta \log \pi_\theta(a_t \mid s_{t-1}; G) \right], \end{aligned} \tag{2}$$

where $A(\tau; G) = R(\tau; G) - b(G)$ and $b(G)$ is a baseline for variance reduction. Given a mini-batch of B instances and N sampled trajectories per instance, we use

$$\widehat{\nabla_\theta \mathcal{J}} = \frac{1}{BN} \sum_{i=1}^{B} \sum_{j=1}^{N} A_{i,j} \sum_{t=1}^{M'_i} \nabla_\theta \log \pi_\theta(a_{i,j,t} \mid s_{i,j,t-1}; G_i), \tag{3}$$

where $A_{i,j} = R(\tau_{i,j}; G_i) - b(G_i)$. For POMO-style training, $b(G_i)$ is typically the average reward of the N trajectories sampled for instance G_i. Here, $R(\tau; G)$ is the reward function (e.g., $R(\tau; G) = -c(\tau; G)$ for TSP), M is the problem size, and M' is the decision length[2]. Optimizing $\mathcal{J}(\theta)$ becomes increasingly difficult for large M: rewards are usually available only after a complete trajectory is constructed, and informative positive-advantage samples become rarer as the policy approaches a local optimum, leading to weaker exploration and less stable policy improvement.

[2]For simplicity, we use M in later sections when $M' = M$, as in TSP.

2.2 CORectifier: Rectified Policy Rollouts with Hierarchical Trajectory Guidance

Overview. Our method aims to alleviate the above issues in RL-based NCO, complementary to recent attempts [25, 26] that improve how the advantage signal $A(\cdot;\cdot)$ is computed. Building on Sec. 2.1, we keep the standard RL-based NCO setting but augment vanilla RL with a labeled subset $\mathcal{G}^* \subset \mathcal{G}$, where each $G \in \mathcal{G}^*$ is paired with a high-quality reference trajectory τ_G^*. Rectification is useful when the reference outperforms the current policy. We state this as an empirical margin condition:

$$\Delta_\theta(G) := \mathbb{E}_{\tau\sim\pi_\theta(\cdot|G)}\left[c(\tau;G)\right] - c(\tau_G^*;G), \qquad \min_{G\in\mathcal{G}^*} \Delta_\theta(G) \geq \epsilon_\theta > 0. \tag{4}$$

Here, ϵ_θ is a policy-dependent margin rather than a global optimality guarantee. It captures the common training regime where reference solutions, produced by strong exact or heuristic solvers, are better than trajectories sampled by the current policy. As the policy improves or when weaker references are used, this margin may shrink; in such cases, the advantage-weighted update in Sec. 2.2 reinforces only rectified rollouts that achieve positive realized advantage. Unlike vanilla IL, which uses the full τ_G^* as a rigid target, CORectifier treats τ_G^* as a source of decomposable reference actions that can be injected locally into RL rollouts.

Whereas conventional RL updates policies via full trajectory gradients (Eq. 2), CORectifier introduces a rectification operator $\mathcal{R} : (\tau, \tau^*, \mathcal{M}) \mapsto \tau'$ that modifies partial segments of policy-generated trajectories τ using τ^*, guided by a hierarchical mask $\mathcal{M} \in \{0,1\}^{N\times M}$ (N: parallel trajectories per instance). The rectified trajectory τ' preserves feasibility and inherits local reference structure from τ_G^*, while keeping the remaining rollout policy-driven.

By injecting segmented signals that guide exploration toward high-quality regions, it addresses two limitations of prior works: 1) IL's early overfitting to full expert trajectories; 2) RL's reward sparsity in arbitrary exploration. Given reward function $R(\cdot;G)$, a high-level view of CORectifier is to improve the expected reward of rectified rollouts, which leads to the following rectified-return formulation:

$$\max_\theta \mathbb{E}_{G\sim\mathcal{D}_\mathcal{G},\, \tau\sim\pi_\theta(\cdot|G),\, \tau'=\mathcal{R}(\tau,\tau_G^*,\mathcal{M})}\left[R(\tau';G)\right]. \tag{5}$$

The Two-Stage Framework. CORectifier employs a two-stage training pipeline to progressively integrate supervised guidance into RL, balancing stability and generalization. A compact workflow is in Alg. 1 in Appendix B, and a PyTorch-style implementation of core parts is in Appendix C.2.

Stage 1 (Optional): Pre-training with Imitation Learning. To initialize the policy with reasonable prior knowledge and avoid random exploration in unpromising regions, we first pre-train using vanilla imitation learning (IL) on fully labeled instances $\mathcal{G}^*$ (with $\{\tau^*\}$). This stage aims to align the policy with expert behavior, providing a sensible starting point for subsequent RL refinement.

In this phase, for each batch of labeled instances $\{G_i\}_{i=1}^B \subseteq \mathcal{G}^*$ (with batch size B), we compute the IL loss by maximizing the model-predicted log-likelihood of expert trajectories $\{\tau_{G_i}^*\}$, i.e., $\mathcal{L}_{\text{IL}}(\theta)$:

$$\mathcal{L}_{\text{IL}}(\theta) = -\mathbb{E}_{G\in\mathcal{G}^*} \log \pi_\theta(\tau_G^* \mid G) \approx -\frac{1}{B}\sum_{i=1}^{B} \log \pi_\theta(\tau_{G_i}^* \mid G_i). \tag{6}$$

Here, $\log \pi_\theta(\tau_{G_i}^* \mid G_i) = \sum_{t=1}^{M} \log \pi_\theta(a_{i,t}^* \mid s_{i,t-1}^*; G_i)$ is the log-probability of the expert trajectory $\tau_{G_i}^* = [a_{i,1}^*, \ldots, a_{i,M}^*]$, with $s_{i,t}$ being the state at step t (e.g., nodes visited up to $t-1$ for TSP). This stage aims to quickly adapt the policy to basic patterns from expert trajectories.

Stage 2: Hierarchical Rectified Reinforcement Learning (RRL). The core training phase integrates hierarchical rectification into RL, dynamically injecting expert segments into policy-generated trajectories, balancing three aspects: 1) *Exploration*: free policy sampling to discover novel solutions, 2) *Exploitation*: leveraging learned patterns to refine good solutions, and 3) *Expert guidance*: incorporating reference segments from τ_G^* through three gated operations:

1) Batch Level: Probabilistically Sourced Training Data. At each batch iteration, we probabilistically select between labeled and unlabeled instances using a batch-wise rectification probability $p_{\text{batch}} \in (0,1)$. Formally, with $\mathcal{G}$ denoting the entire space of unlabeled problem instances, we sample

$$\{G_i\}_{i=1}^B \sim \begin{cases} \mathcal{G}^* & \text{with probability } p_{\text{batch}} \\ \mathcal{G}\setminus\mathcal{G}^* & \text{with probability } 1-p_{\text{batch}} \end{cases}. \tag{7}$$

While $\mathcal{G}^*$ remains fixed as a training set with off-the-shelf high-quality reference solutions (e.g., form [34]), batches from $\mathcal{G} \setminus \mathcal{G}^*$ are randomly generated on the fly from the infinite problem space, allowing the policy to observe unseen instances while periodically leveraging high-quality guidance.

2) Instance Level: Trajectory Sampling and Rectification Candidates. For each instance G_i in the batch, we sample $N = \min(M, 200)$ (the truncation is for memory efficiency) parallel trajectories $\mathcal{T}_i = \{\tau_i^j\}_{j=1}^N \sim \pi_\theta(\cdot \mid G_i)$ as in [19]. Prior to the calculation of the REINFORCE baseline, a subset $\widehat{\mathcal{T}}_i \subseteq \mathcal{T}_i$ is randomly selected for rectification, where the proportion of rectified trajectories is controlled by p_{inst} (instance-wise rectification probability):

$$|\widehat{\mathcal{T}}_i|/|\mathcal{T}_i| = p_{\text{inst}}. \tag{8}$$

This controls the density of expert guidance within parallel samples corresponding to each graph instance in a mini-batch. It ensures that partial trajectories stand out against the mean baseline with distinct advantages, while preventing the policy from being overwhelmed by reference signals.

3) Intra-instance Level: Masked Rectifiers with Varied Lengths. For trajectories in $\widehat{\mathcal{T}}_i$ (w.r.t. G_i), we generate a binary rectifier mask $\mathcal{M}_{G_i} \in \{0,1\}^{N \times M}$ to specify which decision steps are eligible for rectification. This mask is constructed as follows (directly for TSP and ATSP)[3]:

$$\mathcal{M}_{G_i} = \text{Concat}\left(\mathbf{1}^{N \times k}; \mathbf{0}^{N \times (M-k)}\right), \tag{9}$$

where $\mathbf{1}$ and $\mathbf{0}$ are all-one and all-zero matrices, $\text{Concat}(\cdot;\cdot)$ denotes concatenation, and $k = \lfloor \gamma M \rfloor$ is the segment length with $\gamma \sim \text{Uniform}(\alpha, \beta)$. The mask is randomly rolled along the step dimension so rectification can occur at diverse positions. Thus, each reference solution τ^* can provide many sub-trajectory guidance signals, improving the data efficiency of reference supervision.

Subsequently, for each trajectory $\tau_i^j \in \widehat{\mathcal{T}}_i$ and action step $t \in \{2, \ldots, M\}$, the masked rectification operates iteratively following the three steps below:

- **Recommended Action Retrieval**: Given the last executed action $a_{i,j,t-1} \in \tau_i^j$, retrieve the direct successor of $a_{i,j,t-1}$ from reference trajectory $\tau_{G_i}^*$ as the next action $a_{i,t}^*$.
- **Constraint Check**: Verify the corresponding state of $a_{i,t}^*$, i.e., $s_{i,j,t}$, is feasible (e.g., unvisited node in TSP). Let $\text{Feas}(a_{i,t}^*, s_{i,j,t-1}) = 1$ if feasible, 0 otherwise.
- **Action Replacement**: If $\mathcal{M}_{G_i}[j,t] = 1$ (mask allows rectification) and $\text{Feas}(a_{i,t}^*, s_{i,j,t-1}) = 1$, replace the policy-selected action $a_{i,j,t} \sim \pi_\theta(\cdot \mid s_{i,j,t-1}; G_i)$ with $a_{i,t}^*$. Formally,

$$\tau_i'^j[t] \leftarrow \begin{cases} a_{i,t}^* & \text{if } \mathcal{M}_{G_i}[j,t] \wedge \text{Feas}(a_{i,t}^*, s_{i,j,t-1}), \\ a_{i,j,t} & \text{otherwise.} \end{cases} \tag{10}$$

Feasibility is preserved by construction: each executed action is either allowed by the policy action mask or is a reference action that passes $\text{Feas}(a_{i,t}^*, s_{i,j,t}) = 1$. Hence, by induction over autoregressive steps, the resulting trajectory satisfies $\tau_i'^j \in \Omega_{G_i}$.

RRL Loss and Policy Update. After rectification, each instance G_i has N trajectories $\tilde{T}_i = \{\tilde{\tau}_{i,j}\}_{j=1}^N$, where $\tilde{\tau}_{i,j}$ is either an original rollout or a rectified one. We compute $R_{i,j} = R(\tilde{\tau}_{i,j}; G_i)$ and $A_{i,j} = R_{i,j} - \frac{1}{N}\sum_{j'=1}^N R_{i,j'}$, using the within-instance mean as the baseline. Let $\bar{\theta}$ be the frozen behavior policy for rollout collection and rectification, and let $\tilde{\Pi}_{\bar{\theta}}(\cdot \mid G)$ denote the induced rectified rollout distribution. We optimize the surrogate:

$$\mathcal{L}_{\text{RRL}}(\theta; \bar{\theta}) = -\mathbb{E}_{G \sim \mathcal{D}_\mathcal{G},\, \tilde{T} \sim \tilde{\Pi}_{\bar{\theta}}(\cdot|G)} \left[\frac{1}{N} \sum_{j=1}^N \hat{A}_j(\tilde{T}; G) \log \pi_\theta(\tilde{\tau}_j \mid G) \right]. \tag{11}$$

Here $\hat{A}_j$ is treated as constant w.r.t. θ. The mini-batch gradient is estimated by

$$\nabla_\theta \mathcal{L}_{\text{RRL}} \approx -\frac{1}{BN} \sum_{i,j} A_{i,j} \nabla_\theta \log \pi_\theta(\tilde{\tau}_{i,j} \mid G_i), \tag{12}$$

[3] Similar operations for PCTSP, CVRP, and the scheduling tasks are detailed in Appendix C.3.

with $\log \pi_\theta(\tilde{\tau}_{i,j} \mid G_i) = \sum_t \log \pi_\theta(\tilde{a}_{i,j,t} \mid \tilde{s}_{i,j,t-1}; G_i)$ evaluated on rectified-prefix states. In practice, $\bar{\theta}$ is set to the current policy before rollout collection, and gradients are stopped through sampling and rectification.

Theoretical justification of the RRL update. Eq. 11 is a surrogate over rectified rollouts generated by the frozen behavior policy, not the exact policy gradient of the high-level rectified-return objective. Under fixed $\bar{\theta}$, its mini-batch gradient estimator is unbiased for the surrogate gradient. Appendix G.2 formalizes this distribution and connects the update to vanilla RL when rectification is disabled and to advantage-weighted teacher forcing when guided trajectories dominate.

3 Related Work

Neural Heatmap-Guided Methods. They leverage neural networks to globally *predict* the selection likelihood of each decision variable, then *decode* these probability heatmaps to generate solutions for specific COPs. Pioneering works employ graph neural networks or relevant backbones [36–38], e.g., GNN4CO [10], Att-GCN [12], and GNNGLS [11]. Recent methods adopt diffusion models and generative learning to enhance performance, e.g., DIFUSCO [13], T2TCO [14, 15], and DIFUCO [39]. Beyond the common supervised learning paradigm, UTSP [40], DIMES [33], Meta-EGN [41], and M^{2}GenCO [42] have explored unsupervised or meta-learning for heatmap prediction.

Neural Sequential-Decision Methods. These methods formulate COPs as Markov Decision Processes (MDPs), where neural networks are trained to predict the optimal next-step action given the current problem state. Among these, PointerNet [32], BQ-NCO [43] and GOAL [24] equip sequential-decision solvers with supervised learning (SL). However, most methods of this type foster a vast field of RL4CO [22]. Earlier works [18, 19, 21] mainly focus on small-scale symmetric instances. Later, MatNet [20], GLOP [44], and UniCO [45], etc., extended the advantages of RL to the more complex ATSP task and beyond. More recent approaches, e.g., BOPO [25] and PO4CO [26], have started to tackle the inherent drawbacks of RL by proposing strategies like Preference Optimization (PO) to improve sample efficiency from a perspective orthogonal to our method. **Note.** A positioning declaration of our method within recent NCO/RL4CO research, including the major method types, applicable tasks, and common evaluation protocols, is provided in Appendix H.

Relation to Expert-guided Sequential Learning. Principles for teacher forcing, dataset aggregation, self-imitation, and advantage-weighted policy learning [46–48] have been established in the general ML context. Related NCO developments such as EAM [49] and LUFFY [50] also improve policy learning by reusing externally improved or off-policy trajectories. CORectifier can be viewed as an NCO-specific operationalization through hierarchical stochastic control, online feasibility checks, and coherent successor-following segments, without online expert queries or iterative dataset aggregation.

4 Experiments

Hardware. All models are trained and tested on a single NVIDIA 3090 24GB GPU with AMD Ryzen Threadripper 3970X 32-Core CPU to ensure consistency and reproducibility.

Datasets. Following the prevailing evaluation norms in prior NCO research, we assess all models on uniform instances across 7 COPs for in-distribution evaluation and supplement generalization results on real-world benchmarks TSPLIB and CVRPLIB. Dataset details are in Appendix A.2.

Metrics. 1) **OBJ.**: average objective value on a certain task. 2) **DROP**: relative drop of the objective w.r.t. the reference solutions. 3) **TIME**: per-instance average solving time.

Compared ML-based Methods. Unlike previous literature, which primarily limited comparisons of RL-based approaches to similar constructive methods on small-scale instances, we expand the evaluation by incorporating both **1) classic sequential-decision methods:** AM [18], POMO [19], Sym-NCO [21], MatNet [20], UniCO [45], GOAL [24], BQ-NCO [43], PO4CO [26], Omni-POMO [58], LCP [59], etc.; and **2) recent powerful heatmap-guided methods:** GNN4CO [10], DIMES [33], GNNGLS [11], UTSP [40], Fast-T2T [15], and COExpander [16].

Model Settings. During training, we set initially $p_{\text{batch}} = p_{\text{inst}} = 0.1$, $\alpha = 0.1, \beta = 0.2$ for TSP/PCTSP/CVRP, and set them as $0.5, 0.8, 1.0$ for ATSP. At testing, the batch-size is set as 1 with greedy decoding for all baselines. N_A denotes data augmentation following [20]. Implementation details, backbones, and unexhausted settings are detailed in Appendices C, D, and E, respectively.

Table 1: Main results on TSP. G: Greedy decoding. RL: Reinforcement Learning. SL: Supervised Learning. UL: Unsupervised Learning. Among the few RL-based method compared with heatmap solvers and evaluated on TSP ≥ 500, we hereinafter **bold** ours that top within the RL/Sequential family, and highlight those surpassing SL-based SOTAs on corresponding settings (w/ or w/o 2Opt).

METHOD	TYPE	**TSP-50** (1,280 inst.)			**TSP-100** (1,280 inst.)			**TSP-500** (128 inst.)		
		OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓
Concorde [51]	Exact	5.688*	0.000%	0.059s	7.756*	0.000%	0.238s	16.546*	0.000%	18.672s
LKH-3 (500) [52]	Heuristics	5.688	0.001%	0.058s	7.756	0.001%	0.176s	16.546	0.002%	1.848s
Heatmap Guided Methods										
GNN4CO [10]	SL+G	5.718	0.520%	0.006s	7.937	2.334%	0.007s	18.148	9.683%	0.047s
DIMES [33]	RL+G	6.325	11.233%	0.003s	8.757	12.918%	0.005s	18.888	14.155%	0.273s
GNNGLS [11]	SL+G	6.179	8.651%	0.019s	8.966	15.597%	0.021s	–	–	–
UTSP [40]	UL+G	7.097	24.808%	0.007s	9.377	20.909%	0.009s	19.761	19.435%	0.059s
Fast-T2T [15]	SL+G	5.723	0.612%	0.007s	7.941	2.381%	0.009s	18.176	9.840%	0.050s
COExpander [16]	SL+G	5.702	0.244%	0.018s	7.846	1.164%	0.020s	17.634	6.574%	0.062s
GNN4CO [10]	SL+G+2Opt	5.691	0.066%	0.007s	7.775	0.243%	0.009s	16.769	1.348%	0.063s
DIMES [33]	RL+G+2Opt	5.891	3.578%	0.007s	8.108	4.543%	0.012s	17.655	6.707%	0.314s
GNNGLS [11]	SL+G+2Opt	5.707	0.333%	0.020s	7.857	1.295%	0.129s	–	–	–
UTSP [40]	UL+G+2Opt	5.976	5.080%	0.012s	8.161	5.215%	0.017s	17.710	7.033%	0.090s
Fast-T2T [15]	SL+G+2Opt	5.691	0.065%	0.007s	7.773	0.223%	0.009s	16.750	1.233%	0.055s
COExpander [16]	SL+G+2Opt	5.689	0.029%	0.018s	7.765	0.124%	0.021s	16.684	0.837%	0.070s
Sequential Decision Methods										
AM [18]	RL+G	5.747	1.040%	0.008s	7.951	2.520%	0.033s	–	–	–
POMO [19]	RL+G	5.698	0.179%	0.142s	7.883	1.641%	0.151s	–	–	–
Sym-NCO [53]	RL+G	5.702	0.258%	0.050s	7.872	1.491%	0.098s	–	–	–
GOAL [24]	SL+G	5.742	0.965%	0.393s	7.905	1.920%	0.762s	18.006	8.825%	3.964s
CORectifier	RRL+G	**5.697**	**0.157%**	0.027s	**7.818**	**0.795%**	0.051s	**17.359**	**4.915%**	0.239s
CORectifier (N_A=16)	RRL+G	**5.689**	**0.017%**	0.029s	**7.782**	**0.338%**	0.056s	**17.247**	**4.241%**	1.652s
CORectifier (N_A=128)	RRL+G	**5.688**	**0.009%**	0.068s	**7.775**	**0.249%**	0.232s	**17.216**	**4.024%**	12.138s
AM [18]	RL+G+2Opt	5.733	0.799%	0.079s	7.902	1.887%	0.184s	–	–	–
POMO [19]	RL+G+2Opt	5.693	0.103%	0.182s	7.854	1.261%	0.225s	–	–	–
Sym-NCO [53]	RL+G+2Opt	5.699	0.197%	0.051s	7.840	1.087%	0.100s	–	–	–
CORectifier	RRL+G+2Opt	5.695	0.132%	0.027s	**7.807**	**0.664%**	0.052s	**17.140**	**3.593%**	0.389s
CORectifier (N_A=16)	RRL+G+2Opt	**5.688**	**0.011%**	0.030s	**7.777**	**0.276%**	0.055s	**17.068**	**3.154%**	1.668s
CORectifier (N_A=128)	RRL+G+2Opt	**5.688**	**0.007%**	0.069s	**7.771**	**0.198%**	0.249s	**17.039**	**2.978%**	12.912s

Table 2: Main results on PCTSP. Results are partially quoted† from [54]. f-t: finetune.

METHOD	TYPE	**PCTSP-20** (10,000 inst.)			**PCTSP-50** (10,000 inst.)			**PCTSP-100** (10,000 inst.)		
		OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓
OR-Tools (60s) [55]	Heuristics	3.134*	0.000%	60.005s	4.477*	0.000%	60.019s	5.994	0.231%	60.041s
ILS (C++) [56]	Heuristics	3.159	0.783%	0.148s	4.499	0.491%	1.256s	5.981*	0.000%	6.860s
AM [18]	RL+G	3.233	3.176%	0.053s	4.633	3.511%	0.095s	6.429	7.506%	0.167s
MCOMAB† (f-t) [57]	RL+G	3.34	6.79%	–	4.56	1.83%	–	–	–	–
GCNCO† (f-t) [54]	RL+G	3.26	4.25%	–	4.54	1.64%	–	–	–	–
CORectifier	RRL+G	**3.150**	**0.504%**	0.465s	**4.541**	**1.445%**	0.865s	**6.206**	**3.775%**	1.638s

4.1 Main Results and Analysis

Main experimental results for routing tasks are in Tables 1, 2, 3, 6 for TSP, PCTSP, CVRP, and ATSP, respectively. The training overhead for both memory and time is reported in Appendix F.1.

Comparison within RL-based Methods. CORectifier outperforms compared RL-based solvers (greedy), achieving reduced optimality drops by **47.69%**, **32.05%**, **59.67%**, **37.22%** on average with 50–100-node TSP, ATSP, PCTSP, and CVRP (a range where RL-based solvers prevail) respectively. Beyond these, we further extend the scalability of RL-based methods whereby larger CO instances could also potentially enjoy the benefits from the constraint-friendly sequential RL learners.

Comparison with Broader ML-based Approaches. On TSP and ATSP, domains dominated by SL-based heatmap-guided solvers, CORectifier delivers superior solution quality on 6 out of 7 test benchmarks (highlighted boxes). When decoding greedily without any augmentations, CORectifier reduces the optimality drop by an overall **26.49%** (TSP) and **8.98%** (ATSP) across 50–500-node instances, respectively. Admittedly, heatmap-based solvers are still powerful for large-scale TSP/ATSP, however, they can hardly model CO tasks with more complex constraints (e.g., PCTSP, CVRP, KP, JSSP, etc.), which highlights the necessity of progressively advancing RL-driven solvers.

Scalability Highlight. Poor scalability has long hindered the progress of RL-based NCO methods. In Table 9, we summarize the results of commonly compared RL-based solvers on TSP-500. Notably, CORectifier with greedy decoding substantially reduces the performance degradation relative to RL baselines, by up to **89.8%**, demonstrating improved scalability on graph-based problems with more than 100 nodes, a setting that remains challenging for many existing RL4CO approaches.

Table 3: Main results on CVRP. Baselines are either quoted† or re-implemented upon the RL4CO [53] and ML4CO-Bench-101 [34] benchmarks. LS: classic Local Search adopted in [16].

METHOD	TYPE	**CVRP-50** (10000 inst.)			**CVRP-100** (10000 inst.)		
		OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓
HGS [60]	Heuristics	10.366*	0.000%	1.005s	15.563*	0.000%	20.027s
Heatmap Guided Methods							
Fast-T2T [15]	SL+G	12.640	21.835%	0.009s	19.202	23.333%	0.010s
COExpander [16]	SL+G	11.979	15.407%	0.033s	17.497	12.343%	0.047s
Fast-T2T [15]	SL+G+LS	10.871	4.836%	0.013s	16.294	4.698%	0.018s
COExpander [16]	SL+G+LS	10.773	3.903%	0.037s	16.224	4.253%	0.055s
Sequential Decision Methods							
AM† [18]	RL+G	10.98	5.86%	–	16.80	7.34%	–
POMO† [19]	RL+G	10.74	3.52%	–	16.15	3.00%	–
Sym-NCO [21]	RL+G	10.769	3.891%	0.087s	16.220	4.241%	0.166s
GOAL [24]	SL+G	10.906	5.193%	0.504s	16.342	5.005%	0.962s
CORectifier	RRL+G	**10.540**	**1.668%**	0.042s	**15.939**	**2.425%**	0.079s
CORectifier (N_A=16)	RRL+G	**10.447**	**0.768%**	0.049s	**15.799**	**1.521%**	0.108s
Sym-NCO [21]	RL+G+LS	10.505	1.910%	0.168s	15.933	2.379%	0.173s
GOAL [24]	SL+G+LS	10.628	2.519%	0.507s	15.959	2.548%	0.969s
CORectifier	RRL+G+LS	**10.469**	**0.984%**	0.045s	**15.796**	**1.496%**	0.085s
CORectifier (N_A=16)	RRL+G+LS	**10.412**	**0.437%**	0.052s	**15.706**	**0.919%**	0.113s
METHOD	TYPE	**CVRP-200** (100 inst.)			**CVRP-500** (100 inst.)		
		OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓
HGS [60]	Heuristics	19.630*	0.000%	60.024s	37.154*	0.000%	360.376s
Heatmap Guided Methods							
Fast-T2T [15]	SL+G	25.064	27.616%	0.059s	47.749	28.509%	0.091s
COExpander [16]	SL+G	22.402	13.977%	0.145s	43.901	18.199%	0.554s
Fast-T2T [15]	SL+G+LS	20.662	5.290%	0.063s	39.195	5.530%	0.215s
COExpander [16]	SL+G+LS	20.587	4.893%	0.153s	39.121	5.337%	0.605s
Sequential Decision Methods							
Sym-NCO [21]	RL+G	20.662	5.274%	0.320s	40.382	8.723%	0.769s
CORectifier	RRL+G	**20.252**	**3.166%**	0.159s	**38.892**	**4.687%**	0.368s
CORectifier (N_A=16)	RRL+G	**20.105**	**2.420%**	0.219s	**38.675**	**4.103%**	1.210s
Sym-NCO [21]	RL+G+LS	20.193	2.880%	0.341s	38.700	4.173%	0.883s
CORectifier	RRL+G+LS	**20.010**	**1.929%**	0.171s	**38.222**	**2.891%**	0.461s
CORectifier (N_A=16)	RRL+G+LS	**19.929**	**1.518%**	0.237s	**38.067**	**2.462%**	1.295s
CORectifier (N_A=128)	RRL+G+LS	**19.914**	**1.445%**	0.990s	**38.050**	**2.414%**	9.231s

4.2 Extended Evaluations and Discussions

I) Generalization Results. By convention, we first evaluate our models (trained on uniform TSP/CVRP-100) on real-world **1) TSPLIB** with 50-200 nodes (Table 7) and **2) CVRPLIB**-Set-X with 100-1,000 clients (Table 8). Note CORectifier outperforms all compared baselines, showing remarkably reduced optimality gap over popular RL solvers (**ours: 1.05%** v.s. AM: 16.77%/POMO: 3.20% for TSP; **ours: 5.47%** v.s. Sym-NCO: 23.32%/POMO: 10.37% for CVRP). We leave the per-instance results and extended analyses on TSPLIB and CVRPLIB in Appendices F.2.1 and F.2.2. They exhibit robust generalization of CORectifier over (unseen) varying scales and shifted distributions. We further train CORectifier on CVRP-100-500 and evaluate it without further tuning on **3) CVRP-1K/2K/5K**. Table 20 in Appendix F.2.3 shows its competitive performance (drop: 3.58%, 6.37%, and 7.02%, respectively) against large-scale neural solvers, e.g., L2C [61], INViT [62], etc.

Table 4: Main results (DROP↓) on KP (1,280 inst.).

METHOD	**KP-200**	**KP-500**
OR-Tools [55]	0.00%	0.00%
AM [18]	0.33%	–
POMO [19]	0.50%	6.41%
BQ-NCO [43]	0.14%	0.74%
GOAL [24]	1.63%	2.40%
CORectifier	**0.01%**	**0.01%**

II) Ablation Studies. 1) On the Three Rectification Levels. Results in Table 10 (additional in Table 23, Appendix F.3.1) show removing the gating mechanism at any rectification level (by loosening the respective probability to 1) degrades performance, confirming the complementary roles of the tri-level rectifiers. **2) On Learning Paradigms.** Fig. 2, along Fig. 6 and Table 24 in Appendix F.3.2 offer intuitive views comparing the RRL learning paradigm and vanilla RL or IL. We further provide a concise numerical summary over the four paradigms in Table 5 (scale: 50). **3) On Matched-Budget Mechanism Isolation.** Appendix F.3.3 compares

Table 5: Direct ablation comparison (DROP↓) on the training paradigms.

TYPE	**TSP**	**ATSP**	**PCTSP**
RL	0.20%	1.64%	3.75%
IL	1.82%	2.37%	4.02%
RRL	0.19%	0.85%	**3.41%**
IL+RRL	**0.16%**	**0.65%**	3.59%

Table 6: Main results on ATSP.

METHOD	TYPE	ATSP-50 (2500 inst.) OBJ.↓	DROP↓	TIME↓	ATSP-100 (2500 inst.) OBJ.↓	DROP↓	TIME↓
LKH-3 (1000) [52]	Heuristics	1.554*	0.000%	0.097s	1.566*	0.000%	0.238s
Heatmap Guided Methods							
GNN4CO [10]	SL+G	1.667	7.261%	0.007s	1.666	6.408%	0.008s
Fast-T2T [15]	SL+G	1.662	6.948%	0.006s	1.665	6.329%	0.008s
COExpander [16]	SL+G	1.637	5.304%	0.019s	1.646	5.086%	0.023s
GNN4CO [10]	SL+G+2Opt	1.613	3.795%	0.008s	1.629	4.004%	0.011s
Fast-T2T [15]	SL+G+2Opt	1.614	3.805%	0.007s	1.630	4.104%	0.009s
COExpander [16]	SL+G+2Opt	1.601	3.029%	0.019s	1.619	3.361%	0.024s
Sequential Decision Methods							
MatNet [20]	RL+G	1.576	1.373%	0.032s	1.620	3.456%	0.053s
UniCO [45]	RL+G	1.579	1.601%	0.044s	1.625	3.768%	0.060s
GOAL [24]	SL+G	1.595	2.567%	0.395s	1.644	5.010%	0.723s
CORectifier	RRL+G	**1.565**	**0.649%**	0.036s	**1.607**	**2.633%**	0.079s
CORectifier (N_A=16)	RRL+G	**1.556**	**0.115%**	0.047s	**1.585**	**1.192%**	0.120s
MatNet [20]	RL+G+2Opt	1.574	1.256%	0.032s	1.616	3.199%	0.057s
GOAL [24]	SL+G+2Opt	1.587	2.079%	0.396s	1.632	4.195%	0.724s
CORectifier	RRL+G+2Opt	**1.564**	**0.585%**	0.036s	**1.604**	**2.394%**	0.081s
CORectifier (N_A=16)	RRL+G+2Opt	**1.556**	**0.113%**	0.048s	**1.584**	**1.127%**	0.121s
CORectifier (N_A=128)	RRL+G+2Opt	**1.555**	**0.032%**	0.186s	**1.576**	**0.603%**	0.557s

METHOD	TYPE	ATSP-200 (100 inst.) OBJ.↓	DROP↓	TIME↓	ATSP-500 (100 inst.) OBJ.↓	DROP↓	TIME↓
LKH-3 (1000) [52]	Heuristics	1.565*	0.000%	0.724s	1.573*	0.000%	4.376s
Heatmap Guided Methods							
GNN4CO [10]	SL+G	1.671	6.777%	0.053s	1.729	9.880%	0.157s
Fast-T2T [15]	SL+G	1.678	7.280%	0.037s	1.675	6.443%	0.127s
COExpander [16]	SL+G	1.628	4.078%	0.099s	1.646	4.634%	0.403s
GNN4CO [10]	SL+G+2Opt	1.634	4.391%	0.070s	1.696	7.774%	0.313s
Fast-T2T [15]	SL+G+2Opt	1.652	5.608%	0.046s	1.653	5.088%	0.231s
COExpander [16]	SL+G+2Opt	1.616	3.293%	0.111s	1.630	3.592%	0.478s
Sequential Decision Methods							
MatNet [20]	RL+G	3.831	145.083%	0.114s	–	–	–
CORectifier	RRL+G	**1.649**	**5.407%**	0.145s	**1.710**	**8.699%**	0.377s
CORectifier	RRL+G+2Opt	**1.637**	**4.638%**	0.149s	**1.692**	**7.510%**	0.552s
CORectifier (N_A=128/64)	RRL+G+2Opt	**1.614**	**3.123%**	2.351s	**1.670**	**6.689%**	9.351s

Table 7: Generalization performance on TSPLIB (50-200).

METHOD	DROP↓
GNN4CO [10]	40.04%
AM [18]	16.77%
POMO [19]	3.20%
ICAM [63]	2.38%
Learn2OPT [64]	1.73%
GNNGLS [11]	1.53%
CORectifier	**1.05%**

Table 8: Generalization performance on CVRPLIB (100-1000).

METHOD	DROP↓
Sym-NCO [21]	23.32%
POMO [19]	10.37%
LEHD [65]	12.25%
BQ-NCO [43]	10.89%
Omni-POMO [58]	6.52%
ELG (RF) [66]	6.03%
ELG (PO) [26]	5.94%
CORectifier	**5.47%**

Table 9: Main *RL4CO* methods on TSP-500.

METHOD	DROP↓
AM [18]	20.99%
POMO+EAS-Emb [67]	16.25%
POMO+EAS-Tab [67]	48.22%
Sym-NCO [21]	27.56%
LCP [59]	25.80%
EAN+2Opt [68]	43.57%
CORectifier	**4.92%**

RRL with i) a *single global gate*, ii) *independent action-wise interventions*, and iii) *arbitrary feasible reference actions* under matched expected guidance budgets. RRL achieves the lowest gap on TSP, ATSP, and CVRP without the most proposed or realized interventions, showing that its gains do not arise merely from greater guidance quantity. **4) On Alternative Uses of References.** Appendix F.3.4 compares RRL with i) *joint behavior cloning (BC) and RL*, ii) *full-trajectory reward-weighted imitation*, and iii) *matched post-IL RL/RRL*. RRL performs best on all three controlled tasks. **Takeaway**: IL is only an optional warm start. The controlled results support selective, state-consistent rollout rectification over greater guidance quantity or generic imitation.

III) Hyper-parameter Studies. We empirically analyze the impact of three core hyper-parameters governing the tri-level hierarchical rectification mechanism: **Batch-wise (p_{batch})** and **Instance-wise Rectification Probability (p_{inst})**, and **Length Range of Rectified Segments ($[\alpha, \beta]$)**. Results are presented in Appendix F.4. The model's performance remains relatively stable as p_{batch} and p_{inst} vary across different values, with a slight degradation observed as these probabilities increase. For the rectified segment length range $[\alpha, \beta]$, Tables 27 and 28 show similarly stable trends that validate our choice of settings. Note we avoid setting p_{batch}, p_{inst}, α, or $\beta < 0.1$ in case the model degenerates to vanilla RL, as shown in Table 29.

Table 10: Ablation on the rectification levels (100 epochs, TSP-100).

SETTING	OBJ.↓	DROP↓
w/o batch level	8.080	4.183%
w/o instance level	8.061	3.937%
w/o intra-instance level	8.077	4.134%
Full model	**8.039**	**3.655%**

IV) Stability Studies. 1) On Sparse/Sub-optimal Supervision. Table 11 evaluates i) fewer reference solutions (1.28K/12.8K), ii) sub-optimal references generated by one LKH trial or HGS-1s, and iii) noisy references obtained by shuffling the nodes within a randomly selected 10% subtour. CORectifier remains stable under all three settings, with only modest degradation even when using 1/100 of the reference data. **2) On Instance-level Solving Variance.** Following common practice [23, 34], Appendix F.5.1 reports the mean and standard deviation of per-instance optimality drops across all evaluated task-scale pairs, character-

Table 11: Effect (Obj↓) of expert data quantity (Qnty) and quality (Qlty).

Qnty / Qlty.	TSP-100	ATSP-100	CVRP-100
128K Opt.	**7.956**	**1.692**	**16.312**
12.8K Opt.	7.956	1.693	16.313
1.28K Opt.	7.974	1.697	16.313
128K Sub-Opt.	7.986	1.701	16.347
128K Noisy	7.964	1.813	16.353

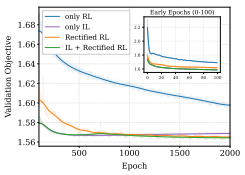


Figure 2: Training efficiency (on ATSP-50).

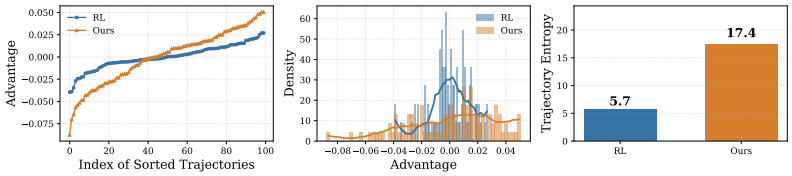


Figure 3: Training diversity on TSP-100 (following plots in [26]). L: Advantage separation. M: Advantage scale distribution. R: Trajectory entropy.

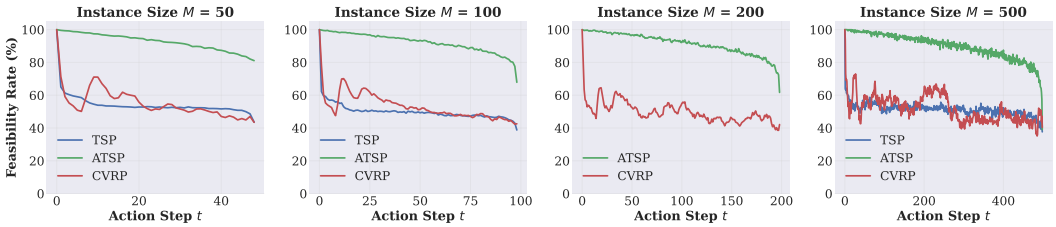


Figure 4: Per-step feasibility rate: the proportion of samples where the expert-suggested next action remains valid under the current partial solution at action step t (during training).

izing dispersion over test instances rather than training randomness. **3) On Seed-level Training Stability.** Appendix F.5.2 further reports results from three independent 200-epoch training runs on four 100-node tasks. RRL achieves lower mean gaps and smaller across-seed standard deviations than vanilla RL on four tasks, indicating the observed gains are not driven by a single favorable seed.

V) Applicability Discussion. 1) Multiple Backbones. Following [21], we show the adaptability of our learning paradigm to some best-known RL4CO methods: POMO, MatNet, and AM by employing them as the backbone (detailed in Appendix D) in main experiments for TSP/CVRP, ATSP, and PCTSP, respectively. **2) Beyond Routing Tasks**. Results on KP are in Table 4 (full in Appendix F.6.2), showing overwhelming advantages over neural baselines. Further, we adapt CORectifier to more complex-constrained scheduling tasks, giving results in Table 12 (for SMTWTP) and Appendix F.6.1 (for JSSP, scales: $6 * 6 \sim 20 * 20$), exhibiting solution superiority on non-graph-based combinatorial problems in most settings.

Table 12: Main results (DROP↓) on SMTWTP.

METHOD	SMTWTP-20	SMTWTP-50	SMTWTP-100
OR-Tools (10s)	0.00%	0.00%	0.00%
POMO (RL)	3.61%	1.62%	2.35%
POMO (IL)	3.24%	0.64%	1.49%
CORectifier (RRL)	**1.47%**	**-2.82%**	**-4.37%**

VI) Exploration Diversity. Fig. 3 shows CORectifier effectively provides **1)** a clearer separation of trajectory advantages for informative sampling, **2)** a smooth and scattered distribution for advantage scales, and **3)** substantially higher trajectory entropy. So, it encourages more diverse exploration that prior RL4CO models lack. See full plots in Appendix F.7. Notably, our rectification mechanism strengthens exploration, stabilizes training, and enhances overall scalability of RL-based solvers.

VII) Rectification Feasibility. Table 13 summarizes the per-step feasibility rate averaged over decision steps across tasks and problem sizes. Fig. 4 shows the complete trend of the feasibility rate with the decision steps grow. Note that feasibility remains stable across instance sizes: around 50% for TSP/CVRP and above 90% for ATSP. This indicates that rectification opportunities do not diminish with longer horizons. Importantly, these feasibility levels are higher than the rectification probabilities used for training, confirming the rectifier operates in a well-controlled regime where infeasible expert actions rarely limit the effective rectification rate.

Table 13: Average per-step feasibility rate of expert rectification across tasks/scales.

TASK	50	100	200	500
TSP	54.33%	50.10%	–	51.36%
ATSP	92.62%	92.21%	91.47%	90.05%
CVRP	51.31%	50.33%	48.32%	50.62%

5 Conclusion and Outlook

In this paper, we propose CORectifier, a rectified reinforcement learning framework for CO with hierarchical expert guidance. As empirically validated, the RRL paradigm improves sample efficiency and introduces optimality-aware supervision while preserving the sequential decision-making nature of RL solvers. Limitations and broader impacts are discussed in Appendices I and J. Future work will extend the evaluation to more COPs and explore integration with stronger backbones.

Acknowledgements

This work was partly supported by NSFC (92370201) and Shanghai Municipal Universities Industry-Academia-Research Joint Innovation Program.

References

[1] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[2] L. Baty, K. Jungel, P. S. Klein, A. Parmentier, and M. Schiffer, "Combinatorial optimization-enriched machine learning to solve the dynamic vehicle routing problem with time windows," *Transportation Science*, 2024.

[3] J. Ma, W. Pan, B. Cai, X. Zhang, and J. Yan, "NLD4CO: Neural langevin dynamics for combinatorial optimization," in *The Fortieth Annual Conference on Neural Information Processing Systems*, 2026.

[4] J. Yan, J. Liu, Y. Tu, F. Zhou, W. Pan, Z. Gui, and L. Shi, "Design linear constrained neural layers with implicit convex optimization," in *Forty-third International Conference on Machine Learning*, 2026.

[5] R. Wang, Z. Guo, W. Pan, J. Ma, Y. Zhang, N. Yang, Q. Liu, L. Wei, H. Zhang, C. Liu, Z. Jiang, X. Yang, and J. Yan, "Pygmtools: A python graph matching toolkit," *Journal of Machine Learning Research*, vol. 25, no. 33, pp. 1–7, 2024.

[6] Y. Li, J. Ma, Y. Yang, Q. Wu, H. Zha, and J. Yan, "Generative modeling reinvents supervised learning: Label repurposing with predictive consistency learning." in *ICML*, 2025.

[7] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[8] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *arXiv preprint arXiv:2102.09544*, 2021.

[9] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *International conference on machine learning*. PMLR, 2016, pp. 2702–2711.

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

[11] B. Hudson, Q. Li, M. Malencia, and A. Prorok, "Graph neural network guided local search for the traveling salesperson problem," in *International Conference on Learning Representations*, 2022.

[12] Z.-H. Fu, K.-B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large tsp instances," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 7474–7482.

[13] Z. Sun and Y. Yang, "DIFUSCO: Graph-based diffusion solvers for combinatorial optimization," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[14] Y. Li, J. Guo, R. Wang, and J. Yan, "From distribution learning in training to gradient search in testing for combinatorial optimization," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[15] Y. Li, J. Guo, R. Wang, H. Zha, and J. Yan, "Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[16] J. Ma, W. Pan, Y. Li, and J. Yan, "COExpander: Adaptive solution expansion for combinatorial optimization," in *Forty-second International Conference on Machine Learning*, 2025.

[17] Y. Wang, Y. Li, J. Ma, J. Yan, and Y. Chang, "Nexco: Native solution expansion for diffusion-based combinatorial optimization," in *The Fourteenth International Conference on Learning Representations*, 2026.

[18] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *arXiv preprint arXiv:1803.08475*, 2018.

[19] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 188–21 198, 2020.

[20] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5138–5149, 2021.

[21] M. Kim, J. Park, and J. Park, "Sym-nco: Leveraging symmetricity for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1936–1949, 2022.

[22] F. Berto, C. Hua, J. Park, L. Luttmann, Y. Ma, F. Bu, J. Wang, H. Ye, M. Kim, S. Choi, N. G. Zepeda, A. Hottung, J. Zhou, J. Bi, Y. Hu, F. Liu, H. Kim, J. Son, H. Kim, D. Angioni, W. Kool, Z. Cao, J. Zhang, K. Shin, C. Wu, S. Ahn, G. Song, C. Kwon, L. Xie, and J. Park, "RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark," *arXiv preprint arXiv:2306.17100*, 2024.

[23] Y. Li, J. Ma, W. Pan, R. Wang, H. Geng, N. Yang, and J. Yan, "Unify ml4tsp: Drawing methodological principles for tsp and beyond from streamlined design space of learning and search," in *The Thirteenth International Conference on Learning Representations*, 2025.

[24] D. Drakulic, S. Michel, and J.-M. Andreoli, "GOAL: A generalist combinatorial optimization agent learner," in *The Thirteenth International Conference on Learning Representations*, 2025.

[25] Z. Liao, J. Chen, D. Wang, Z. Zhang, and J. Wang, "BOPO: Neural combinatorial optimization via best-anchored and objective-guided preference optimization," in *Forty-second International Conference on Machine Learning*, 2025.

[26] M. Pan, G. Lin, Y.-W. Luo, B. Zhu, Z. Dai, L. Sun, and C. Yuan, "Preference optimization for combinatorial optimization problems," in *Forty-second International Conference on Machine Learning*, 2025.

[27] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine, "Efficient online reinforcement learning with offline data," in *International Conference on Machine Learning*. PMLR, 2023, pp. 1577–1594.

[28] D. Gao, H. Wang, H. Zhou, N. Ammar, S. Mishra, A. Moradipari, I. Soltani, and J. Zhang, "In-ril: Interleaved reinforcement and imitation learning for policy fine-tuning," 2025.

[29] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[30] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," *Advances in neural information processing systems*, vol. 28, 2015.

[31] A. M. Lamb, A. G. ALIAS PARTH GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," *Advances in neural information processing systems*, vol. 29, 2016.

[32] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[33] R. Qiu, Z. Sun, and Y. Yang, "DIMES: A differentiable meta solver for combinatorial optimization problems," in *Advances in Neural Information Processing Systems 35*, 2022.

[34] J. Ma, W. Pan, Y. Li, and J. Yan, "Ml4co-bench-101: Benchmark machine learning for classic combinatorial problems on graphs," in *Advances in Neural Information Processing Systems*, 2025.

[35] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[36] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[37] D. Meng, Z. Cao, J. Gao, Y. Wu, and Y. Hou, "UniteFormer: Unifying node and edge modalities in transformers for vehicle routing problems," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[38] W. Pan, S. Yan, N. Chen, Y. Qu, J. Ma, and J. Yan, "UECO: A unified encoder for neural combinatorial optimization via iterative edge-node reasoning," in *The Fortieth Annual Conference on Neural Information Processing Systems*, 2026.

[39] S. Sanokowski, S. Hochreiter, and S. Lehner, "A diffusion model framework for unsupervised neural combinatorial optimization," in *Forty-first International Conference on Machine Learning*, 2024.

[40] Y. Min, Y. Bai, and C. P. Gomes, "Unsupervised learning for solving the travelling salesman problem," in *Advances in Neural Information Processing Systems*, 2023.

[41] H. P. Wang and P. Li, "Unsupervised learning for combinatorial optimization needs meta learning," in *The Eleventh International Conference on Learning Representations*, 2023.

[42] W. Pan, J. Ma, N. Chen, Y. Li, and J. Yan, "Problem distributions as tasks: Repurposing meta learning for generative combinatorial optimization towards multi-task pretraining and adaptation," in *Forty-third International Conference on Machine Learning*, 2026.

[43] D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli, "Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization," in *Advances in Neural Information Processing Systems*, 2023.

[44] H. Ye, J. Wang, H. Liang, Z. Cao, Y. Li, and F. Li, "Glop: Learning global partition and local construction for solving large-scale routing problems in real-time," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

[45] W. Pan, H. Xiong, J. Ma, W. Zhao, Y. Li, and J. Yan, "UniCO: On unified combinatorial optimization via problem reduction to matrix-encoded general TSP," in *The Thirteenth International Conference on Learning Representations*, 2025.

[46] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 627–635.

[47] J. Oh, Y. Guo, S. Singh, and H. Lee, "Self-imitation learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 10–15 Jul 2018, pp. 3878–3887.

[48] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning," *arXiv preprint arXiv:1910.00177*, 2019.

[49] S. Gu, K. Li, J. Xing, Y. Zhang, and J. Cheng, "Synergizing reinforcement learning and genetic algorithms for neural combinatorial optimization," *arXiv preprint arXiv:2506.09404*, 2025.

[50] J. Yan, Y. Li, Z. Hu, Z. Wang, G. Cui, X. Qu, Y. Cheng, and Y. Zhang, "Learning to reason under off-policy guidance," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[51] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Concorde tsp solver," 2006.

[52] K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems," *Roskilde: Roskilde University*, vol. 12, 2017.

[53] F. Berto, C. Hua, J. Park, M. Kim, H. Kim, J. Son, H. Kim, J. Kim, and J. Park, "RL4CO: a unified reinforcement learning for combinatorial optimization library," in *Neural Information Processing Systems 2023, Workshop: New Frontiers in Graph Learning*, 2023.

[54] Y. Li, J. Liu, Q. Wu, X. Qin, and J. Yan, "Toward learning generalized cross-problem solving strategies for combinatorial optimization," 2025.

[55] T. Cuvelier, F. Didier, V. Furnon, S. Gay, S. Mohajeri, and L. Perron, "Or-tools' vehicle routing solver: A generic constraint-programming solver with heuristic search for routing problems," in *24e congrès annuel de la société française de recherche opérationnelle et d'aide à la décision*, 2023.

[56] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

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

[58] J. Zhou, Y. Wu, W. Song, Z. Cao, and J. Zhang, "Towards omni-generalizable neural methods for vehicle routing problems," in *International Conference on Machine Learning*, 2023.

[59] M. Kim, J. Park, and J. Kim, "Learning collaborative policies to solve np-hard routing problems," in *Advances in neural information processing systems*, 2021.

[60] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems," *Operations Research*, vol. 60, no. 3, pp. 611–624, 2012.

[61] F. Luo, X. Lin, M. Zhong, F. Liu, Z. Wang, J. Sun, and Q. Zhang, "Learning to insert for constructive neural vehicle routing solver," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[62] H. Fang, Z. Song, P. Weng, and Y. Ban, "INViT: A generalizable routing problem solver with invariant nested view transformer," in *Forty-first International Conference on Machine Learning*, 2024.

[63] C. Zhou, X. Lin, Z. Wang, X. Tong, M. Yuan, and Q. Zhang, "Instance-conditioned adaptation for large-scale generalization of neural combinatorial optimization," *CoRR*, 2024.

[64] P. R. d. O. da Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," in *Proceedings of The 12th Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. J. Pan and M. Sugiyama, Eds., vol. 129. PMLR, 18–20 Nov 2020, pp. 465–480.

[65] F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang, "Neural combinatorial optimization with heavy decoder: Toward large scale generalization," in *The 37th Anniversary Conference on Neural Information Processing Systems*, 2023.

[66] C. Gao, H. Shang, K. Xue, D. Li, and C. Qian, "Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, 2024.

[67] A. Hottung, Y.-D. Kwon, and K. Tierney, "Efficient active search for combinatorial optimization problems," *arXiv preprint arXiv:2106.05126*, 2021.

[68] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*. Springer, 2018, pp. 170–181.

[69] E. Balas, "The prize collecting traveling salesman problem," *Networks*, vol. 19, no. 6, pp. 621–636, 1989.

[70] P. Tassel, M. Gebser, and K. Schekotihin, "A reinforcement learning environment for job-shop scheduling," *CoRR*, vol. abs/2104.03760, 2021.

[71] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[72] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[73] L. Xin, W. Song, Z. Cao, and J. Zhang, "Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[74] Z. Zheng, C. Zhou, T. Xialiang, M. Yuan, and Z. Wang, "Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[75] C. Zhou, X. Lin, Z. Wang, and Q. Zhang, "Learning to reduce search space for generalizable neural routing solver," 2026.

[76] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *Advances in neural information processing systems*, vol. 33, pp. 1621–1632, 2020.

[77] J. Zhou, Z. Cao, Y. Wu, W. Song, Y. Ma, J. Zhang, and C. Xu, "Mvmoe: Multi-task vehicle routing solver with mixture-of-experts," *arXiv preprint arXiv:2405.01029*, 2024.

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

[79] X. Jiang, Y. Wu, Y. Wang, and Y. Zhang, "UNCO: Towards unifying neural combinatorial optimization through large language model," *arXiv preprint arXiv:2408.12214*, 2024.

[80] Z. Fang, D. Wang, J. Chen, J. Wang, and Z. Zhang, "UCPO: A universal constrained combinatorial optimization method via preference optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Appendix

Contents

A Formal Problem Definitions and Dataset Preparation **17**

A.1 Problem Definitions 17

A.2 Training and Testing Datasets 18

B Algorithm 1: Complete Pipeline of CORectifier **18**

C Supplementary Clarification of the Rectification Operator $\mathcal{R}$ **18**

C.1 Comparison of Mainstream Learning Objectives 18

C.2 Implementation Details with Pseudo-codes 18

C.3 Adapted Rectification Operation for Complex-constrained Problems 21

D Adapted Model Architectures **23**

D.1 Attention Model (AM) 23

D.2 Policy Optimization with Multiple Optima (POMO) 24

D.3 Matrix Encoding Network (MatNet) 25

E Detailed Experimental Settings **26**

F Supplementary Results **28**

F.1 Computational Overhead for Training 28

F.2 Supplementary Generalization Studies 28

F.2.1 Full Results on TSPLIB 28

F.2.2 Full Results on CVRPLIB 29

F.2.3 Results on CVRP-1K to CVRP-5K 29

F.3 Extended Ablation Studies 30

F.3.1 Results and Clarifications on the Rectification Levels 30

F.3.2 Results on Learning Paradigms 32

F.3.3 Mechanism Isolation under a Matched Guidance Budget 32

F.3.4 Comparison with Alternative Expert-Guidance Paradigms 33

F.4 Extended Results on the Effect of Hyper-parameters 34

F.5 Supplementary Statistical Studies 35

F.5.1 Instance-level Solving Stability 35

F.5.2 Seed-level Training Variability 36

F.6 Extended Results for Non-graph-based Combinatorial Problems 36

F.6.1 Extended Results and Analysis on SMTWTP and JSSP 36

F.6.2 Extended Results and Analysis on KP 37

F.7 Trajectory-distribution Diagnostics 37

G Further Theoretical Justifications **38**

G.1 A Sketch of Proof for Equation 4 38

G.2 A More Rigorous Interpretation of the RRL Update 39

H Declaration: Positioning within the NCO and RL4CO Landscape **41**

I Limitations **42**

J Broader Impact **42**

A Formal Problem Definitions and Dataset Preparation

A.1 Problem Definitions

- **Traveling Salesman Problem (TSP).** Given G and a cost matrix $\mathbf{C} \in \mathbb{R}^{N\times N}$, where $\mathbf{C}_{ij}$ denotes the cost of edge (i, j), the objective is to find a tour $\tau = (i_1, \cdots, i_N)$ that minimizes the total cost: $\sum_{k=1}^{N-1} \mathbf{C}_{i_k i_{k+1}} + \mathbf{C}_{i_N i_1}$.
- **Asymmetric Traveling Salesman Problem (ATSP).** ATSP is a variant of TSP where the cost matrix $\mathbf{C}$ is not necessarily symmetric, i.e., $\mathbf{C}_{ij} = \mathbf{C}_{ji}$ does not always hold for all $i, j \in \mathcal{V}$. In this paper, we follow [16, 20, 45] to study the metric ATSP, where the triangle inequality holds: $\mathbf{C}_{ij} + \mathbf{C}_{jk} \geq \mathbf{C}_{ik}$ for any distinct nodes i, j, and k.
- **Prize-Collecting Traveling Salesman Problem (PCTSP).** As defined in [69], PCTSP extends the classic TSP by associating each node $v \in \mathcal{V}$ with two values: a prize $p_v \in \mathbb{R}_{\geq 0}$ and a penalty $q_v \in \mathbb{R}_{\geq 0}$. Given a cost matrix $\mathbf{C} \in \mathbb{R}^{N\times N}$ (where $\mathbf{C}_{ij}$ denotes the travel cost between node i and node j), the objective is to find a tour $\tau = (i_1, i_2, \cdots, i_k)$ (visiting a subset of nodes, with $k \leq N$) that satisfies a minimum total prize constraint $\sum_{i\in\tau} p_i \geq P_{\text{min}}$ (where $P_{\text{min}} \in \mathbb{R}_{\geq 0}$ is a predefined threshold), while minimizing the total cost: $\sum_{m=1}^{k-1} \mathbf{C}_{i_m i_{m+1}} + \mathbf{C}_{i_k i_1} + \sum_{v\notin\tau} q_v$.
- **Capacitated Vehicle Routing Problem (CVRP).** Given G, a depot node $v_0 \in \mathcal{V}$, a cost matrix $\mathbf{C} \in \mathbb{R}^{N\times N}$, a demand vector $\mathbf{d} \in \mathbb{R}_+^N$, and a vehicle capacity $Q > 0$, the goal is to plan a set of routes $\mathcal{R}$, each route $r \in \mathcal{R}$ starting and ending at the depot v_0, such that each customer node is visited exactly once and the total demand on each route does not exceed Q, i.e., $\sum_{i\in r} \mathbf{d}_i \leq Q$. The objective is to minimize the total cost of all routes: $\min_{\mathcal{R}} \sum_{r\in\mathcal{R}} \sum_{(i,j)\in r} \mathbf{C}_{ij}$.
- **Knapsack Problem (KP).** Given a set of items $\mathcal{I} = \{1, 2, \cdots, M\}$, where each item $i \in \mathcal{I}$ is associated with two attributes: a weight $w_i \in \mathbb{R}_+$ and a value $v_i \in \mathbb{R}_+$, and a knapsack with a maximum weight capacity $W \in \mathbb{R}_+$, the objective is to select a subset $\mathcal{S} \subseteq \mathcal{I}$ of items such that the total weight of the selected items does not exceed the knapsack capacity, i.e., $\sum_{i\in\mathcal{S}} w_i \leq W$, while maximizing the total value of the selected items: $\max_{\mathcal{S}\subseteq\mathcal{I}} \sum_{i\in\mathcal{S}} v_i$. In this work, we focus exclusively on the **0-1 Knapsack Problem** that aligns with the settings in [19, 24, 43], where each item can be optionally included in the subset $\mathcal{S}$ only once.
- **SMTWTP (Single Machine Total Weighted Tardiness Problem).** Given a set of n jobs to be processed on a single machine, the machine can handle only one job at a time, and processing is non-preemptive (i.e., once started, a job cannot be interrupted). Each job i (where $i = 1, \ldots, n$) becomes available for processing at time zero and is defined by an integer processing time p_i, a positive weight w_i, and a due date d_i. For any given processing sequence, let C_i denote the completion time of job i. The tardiness of job i is defined as $T_i = \max\{C_i - d_i, 0\}$. The objective of the SMTWTP is to determine a processing sequence that minimizes the total weighted tardiness, formulated as $\min \sum_{i=1}^{n} w_i \times T_i$.
- **JSSP (Job-Shop Scheduling Problem).** Consider a set of jobs $\mathcal{J}$ and machines $\mathcal{M}$. Each job $J_i \in \mathcal{J}$ consists of a sequence of operations $O_i = \{o_{i,j}\}_{j=1}^{n_i}$, where each operation $o_{i,j}$ requires a specific machine and a processing time $p_{i,j}$. The objective is to find a schedule that satisfies precedence and capacity constraints while minimizing the *makespan* $C_{\max}$, defined as: $\min C_{\max} = \min\left(\max_{i\in\{1,\ldots,|\mathcal{J}|\}} C_i\right)$, where C_i denotes the completion time of job J_i. Following

standard implementations [70], we formulate JSSP as a sequential decision process. At each step t, the agent assigns eligible jobs to available machines. The environment transitions to the next step $t+1$ only when a machine completes an operation and becomes idle.

A.2 Training and Testing Datasets

For **TSP**, we follow [10, 13, 18, 19, 33] to sample graph instances with $M \in \{50, 100, 500\}$ nodes from the unit square $[0, 1]^2$, and obtain reference tours by exact solver Concorde [51]. Beyond the in-distribution scenario, we additionally evaluate the models on the real-world benchmark TSPLIB[4].

For **ATSP**, adhering to [20, 24, 44, 45], we generate the conventional "t-mat"-class instances with $M \in \{50, 100, 200, 500\}$ nodes, where pairwise asymmetric distances satisfy the triangle inequality, and use LKH-3 [52] to compute reference tours.

For **PCTSP**, following [18, 21, 54], we generate instances with $M \in \{20, 50, 100\}$ nodes, and gain reference tours by OR-Tools [55] and ILS [56].

For **CVRP**, we follow [16] to generate instances with $M \in \{50, 100, 200, 500\}$ clients, the demands of which are integers randomly sampled from $[1, 10]$, and the vehicle capacity is set as 40/50/80/100, respectively. The labels are solved by HGS [60]. Beyond the in-distribution scenario, we additionally evaluate the models on the real-world benchmark CVRPLIB-Set-X[5].

For **KP**, we follow [19] to generate instances with $M \in \{50, 100, 200, 500\}$ items, the weights and values of which are uniformly sampled from $[0, 1]$, and the knapsack capacity is set as 12.5, 25, 25, 25, respectively. The reference solutions for KP are also computed by OR-Tools [55].

For **SMTWTP**, we follow [53] to generate instances with $n \in \{20, 50, 100\}$ jobs, where processing times and weights are uniformly sampled from $[0, 1]$, and due dates are uniformly sampled from $[0, n/2]$. The reference heuristic solutions are obtained by the CP-SAT solver in OR-Tools [55].

For **JSSP**, we follow [53] to generate random instances with varying problem sizes ($J \times M \in \{6 \times 6, 10 \times 10, 15 \times 15, 20 \times 20\}$), where operations follow random machine orders and uniform processing times. The reference solutions are computed using OR-Tools [55].

B Algorithm 1: Complete Pipeline of CORectifier

Algorithm 1 streamlines the entire workflow of our proposed Rectified Reinforcement Learning for solving the combinatorial problems.

C Supplementary Clarification of the Rectification Operator $\mathcal{R}$

C.1 Comparison of Mainstream Learning Objectives

Fig. 5 illustrates the differences on the three mainstream learning pipelines for NCO as well as our proposed RRL paradigm.

C.2 Implementation Details with Pseudo-codes

Definition and practical implementation. For mathematical clarity, the rectification operator $\mathcal{R}(\tau, \tau_G^*, \mathcal{M})$ is defined in the main text from a *global* perspective, i.e., as if the entire policy-sampled trajectory τ and expert trajectory τ_G^* were both available beforehand, and a masked replacement is applied afterward. This description is intended to make the concept of "replacing segments of a trajectory" transparent at a macroscopic level.

In practice, however, the rectification procedure is implemented *locally and autoregressively at each step*, rather than by first sampling a full trajectory and then post-hoc modifying it. More precisely, a rectified trajectory $\tau' = \mathcal{R}(\tau, \tau_G^*, \mathcal{M})$ is constructed on the fly as follows. At decision step t, given the current partial trajectory $\tau'_{1:t-1}$, the policy network produces a distribution over feasible next

[4] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[5] http://vrp.atd-lab.inf.puc-rio.br/media/com_vrp/instances

Algorithm 1 CORectifier for Neural Combinatorial Optimization (TSP/ATSP)

Input: Random training data $\mathcal{G}$ of certain distribution; a subset of which, i.e., $\mathcal{G}^* \subset \mathcal{G}$, labeled with $\{\tau_i^*\}_{G_i \in \mathcal{G}^*}$ as reference trajectories; problem-specific objective function $c(\cdot|\cdot)$ and constrained feasible region Ω; batch-wise rectification probability p_{batch}; instance-wise rectification probability p_{inst} (within a mini-batch); length range for rectified segments $[\alpha, \beta]$ (within an instance); problem size M, batch-size B, maximum training epochs N_{ep}, number of data samples per epoch N_s; learning rate η; and schedulers for $p_{\text{batch}}, p_{\text{inst}}, \alpha, \beta$.

Initialize the policy network π_θ with random parameters θ
while not done **do**
  *// **Stage 1 (Optional)**: Pretrain the model with vanilla IL*
  Sample a batch of instances $\{G_i\}_{i=1}^B \sim \mathcal{G}^*$ with expert trajectories $\{\tau_i^*\}$
  Compute imitation loss $\mathcal{L}_{\text{IL}}(\theta) \leftarrow \frac{1}{B}\sum_{i=1}^B [-\log \pi_\theta(\tau^* \mid G_i)]$
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{IL}}(\theta)$ *// Mimick expert trajectories until a rough convergence*
end while

for epoch $\leftarrow 1$ to N_{ep} **do**
  *// **Stage 2 (Core)**: Learning with hierarchical RRL*
  for batch index $\leftarrow 1$ to $\lfloor N_s/B \rfloor$ **do**
    with probability p_{batch}: Sample batch $\{G_i\}_{i=1}^B \sim \mathcal{G}^*$ with expert trajectories $\{\tau_i^*\}$
    with probability $1 - p_{\text{batch}}$: Generate (on the fly) batch $\{G_i\}_{i=1}^B \sim \mathcal{G} \setminus \mathcal{G}^*$ without labels
    Sample $N = \min(M, 200)$ solutions $\mathcal{T}_i \leftarrow \{\tau_i^j\}_{j=1}^N \sim \pi_\theta(\tau \mid G_i)$ for each instance G_i
    if current batch enables expert guidance **then**
      Select a proportion of trajectories $\widehat{\mathcal{T}}_i \subseteq \mathcal{T}_i$ such that $|\widehat{\mathcal{T}}_i| / |\mathcal{T}_i| = p_{\text{inst}}$ for rectification
      Determine rectifier size $k \leftarrow \lfloor \gamma M \rfloor, \gamma \in [0, 1] \sim \text{Uniform}(\alpha, \beta)$
      Generate rectifier mask $\mathcal{M}_{G_i} \in \{0, 1\}^{N \times M} \leftarrow \text{Concat}([\mathbf{1}^{N \times k}; \mathbf{0}^{N \times (M-k)}])$ w.r.t. N solutions
      for $j \in \{1, \cdots N\} \cap \{n \mid \tau_i^n \in \widehat{\mathcal{T}}_i\}$ **do**
        // Start rectification in parallel
        for step $t \leftarrow 2$ to M (assuming $|\tau| = M$ e.g. for TSP) **do**
          Retrieve optimal next action a_t^* from τ_i^*, given last action taken $a_{t-1} \in \tau_i^j$
          if $\mathcal{M}_{G_i}^{j,t}$ is True and a_t^* does not violate problem constraints **then**
            $\tau_i^j \leftarrow \tau'^j_i$ (by replacing $a_t \in \tau_i^j$ with a_t^*) *// Always ensured that $\tau_i^j \in \Omega$*
          end if
        end for
      end for
    end if
    Reward $R_{i,j} \leftarrow -c(\tau_i^j; G_i)$; advantage $A_{i,j} \leftarrow R_{i,j} - \frac{1}{N}\sum_{j'=1}^N R_{i,j'}$;
    Loss $\mathcal{L}_{\text{RL}}(\theta) \leftarrow -A \cdot \mathbb{E}_{\tau \sim \pi_\theta} \log \pi_\theta(\tau \mid G)$
    Compute RL policy gradient $\nabla_\theta \mathcal{L}_{\text{RL}}(\theta) \leftarrow -\frac{1}{BN}\sum_{i=1}^B \sum_{j=1}^N A_{i,j} \nabla_\theta \log \pi_\theta(\tau_i^j \mid G_i)$
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{RL}}(\theta)$
  end for
  $p_{\text{batch}}, p_{\text{inst}}, \alpha, \beta \leftarrow \text{Schedulers}(p_{\text{batch}}, p_{\text{inst}}, \alpha, \beta).\text{step}()$ *// Dynamically scheduling the rectifiers*
end for

actions:

$$a_t \sim \pi_\theta(\cdot \mid s_t), \quad \text{where } s_t = s(\tau'_{1:t-1}). \tag{13}$$

If the rectification mask satisfies $\mathcal{M}[j, t] = 1$, and the expert successor a_t^* in τ_G^* is feasible under the current state (i.e., not violating task constraints), then the expert action is taken as in Eq. 10:

$$a_t \leftarrow a_t^*. \tag{14}$$

Otherwise, the model prediction a_t sampled from π_θ is adopted. The chosen action is appended to the partial trajectory and the environment transitions to the next state. The above procedure repeats until the trajectory is completed, resulting in a fully valid and feasible auto-regressive rollout.

Why does the suffix $\tau_{t+k+1:M}$ remain feasible? The implementation naturally resolves this question as rectification is performed *before* the next state is computed, the environment always observes the rectified prefix $\tau'_{1:t+k}$. Consequently, the subsequent actions $\tau'_{t+k+1:M}$ are generated:

$$a_{t+k+1} \sim \pi_\theta(\cdot \mid s_{t+k+1}), \tag{15}$$

where s_{t+k+1} is the correct state resulting from the rectified actions. Thus, the suffix is *not* inherited from a trajectory generated under a different state; instead, it is recomputed autoregressively under

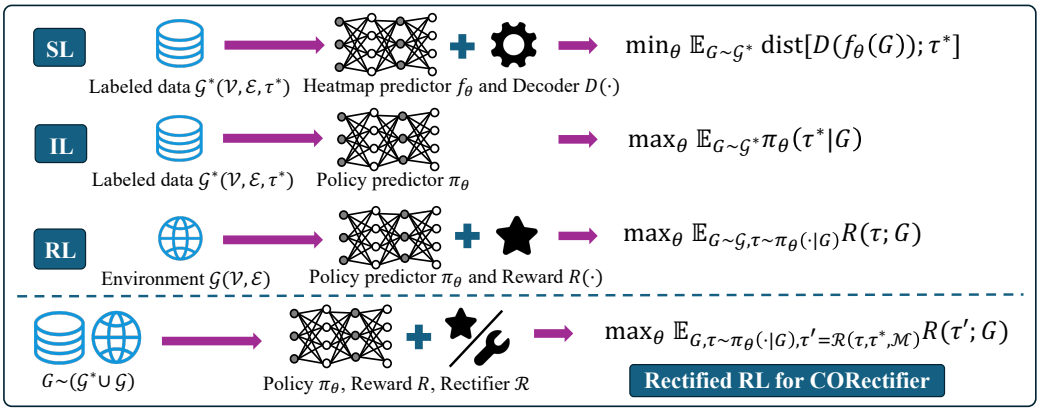


Figure 5: Mainstream NCO learning paradigms. SL minimizes the distance between model-predicted heatmaps and ground-truth matrices; IL and RL maximize the likelihood of expert trajectories or expected reward, respectively; Rectified RL aims to regularize exploration in RL via rectifiers $\mathcal{R}$.

consistent feasibility masks (e.g., visited-node masks in TSP, capacity-feasibility masks in CVRP, etc.). This guarantees that the entire rectified trajectory remains feasible.

An example code for TSP. Below we show a simplified PyTorch-style implementation for the TSP case, consistent with the notation in the paper, where B is the batch size, N is the number of parallel samples per instance, and M is the problem size (number of nodes):

```
class TSPTrainer:
    """
    Skipping other properties and methods ...
    """
    def _train_one_batch(self):
        # data preparations ... skipped

        # randomly decide the number of rectified actions
        rectify_level = int(M * random.uniform(alpha, beta))
        # construct global mask as in Eq. (9): shape (B * N, M)
        global_action_mask = torch.zeros(B * N, M).bool()
        global_action_mask[:, :rectify_level] = True
        global_action_mask.roll_(random.randint(0, M - 1), dims=1)
        # construct instance mask with prob. p_inst: shape (B * N,)
        instance_mask = torch.rand(B * N) < p_inst if enable_rectify else
    None

        while not done:
            # extract intra-instance level mask at current step
            action_mask = global_action_mask[:,
    self.env.selected_count].reshape(-1)
            # forward pass (rectification happens inside the model)
            action, prob = self.model.forward(state, instance_mask,
    action_mask)
            # environment transition and reward computation
            state, reward, done = self.env.step(action)
            # record probabilities for policy gradient
            prob_list = torch.cat((prob_list, prob[:, :, None]), dim=2)

        # backpropagation ... skipped
```

Inside the model, the rectification behavior is implemented as follows:

```
class TSPModel(nn.Module):
    """
    Skipping other properties and methods ...
    """
    def forward(self, state, instance_mask=None, action_mask=None):
        # load reference expert trajectories
```

```
        ref_tours = state.ref_tours
        if state.current_node is None:
            # first step: same as POMO
            selected = torch.arange(N)[None, :].expand(B, N)
            prob = torch.ones(size=(B, N))
        else:
            # decode probabilities over feasible next nodes: (B, N, M)
            node_embed = _get_encoding(state.current_node)
            probs = self.decoder(
                node_embed, ninf_mask=state.ninf_mask
            )
            if self.training:
                # sample initial next action from current policy
                init_action = probs.reshape(B * N, -1).multinomial(1)
                init_action = init_action.squeeze(1).reshape(B, N)
                if instance_mask is not None:
                    # prepare tensors for rectification
                    init_action = init_action.reshape(B * N)
                    cur_node = state.current_node.reshape(B * N)
                    unvisited = (state.ninf_mask==0).reshape(B*N, -1)
                    ref_tours = ref_tours.expand(B,N,M).reshape(B*N,M)
                    # rectified action at current step
                    rectified_action = _rectify_action(
                        cur_node, init_action,
                        instance_mask, action_mask,
                        ref_tours, unvisited
                    ).reshape(B, N)
                    prob = _get_action_prob(probs, rectified_action)
                else:
                    rectified_action = init_action
        return rectified_action, prob

def _rectify_action(
    cur_node,       # shape: (B * N,)
    init_action,    # shape: (B * N,)
    instance_mask,  # shape: (B * N,)
    action_mask,    # shape: (B * N,)
    ref_tour,       # shape: (B * N, M)
    unvisited       # shape: (B * N, M)
):
    # position of current node in the reference tour
    cur_node_idx = _get_idx_by_node(ref_tour, cur_node)
    # next position and next node in the reference tour
    ref_next_idx = (cur_node_idx + 1) % M
    ref_next_node = _get_node_by_idx(ref_tour, ref_next_idx)
    # feasibility check: reference next node must be unvisited
    unvisited_mask = unvisited[torch.arange(B * N), ref_next_node]
    # candidate rectified action (respecting feasibility)
    rectified_action = torch.where(unvisited_mask, ref_next_node,
    init_action)
    # apply instance- and action-level rectification masks
    rectified_action = torch.where(action_mask & instance_mask,
                                   rectified_action, init_action)
    return rectified_action
```

This code clarifies that feasibility is preserved automatically, since every action (rectified or not) is validated against the current environment state and all subsequent decisions are sampled conditioned on this updated state.

C.3 Adapted Rectification Operation for Complex-constrained Problems

In Sec. 2.2, we have delineated the process of our proposed tri-level rectification mechanism, with TSP (and its variants) used for illustration. Here, we elaborate on how this scheme is adapted to tasks with more complex constraints, i.e., CVRP, KP, and SMTWTP. Notably, the first two levels of rectifiers, i.e., batch-level and instance-level, shall remain identical to those employed in training TSP models. However, slight differences emerge in the third step (intra-instance-level rectification).

CVRP. For CVRP, the reference solution is represented as a single sequence with repeated depot tokens,

$$\tau_G^\star = (0, r_1^1, \ldots, r_{m_1}^1, 0, \ldots, 0, r_1^Q, \ldots, r_{m_Q}^Q, 0, \perp, \ldots, \perp), \tag{16}$$

where each segment between two consecutive depot tokens corresponds to one vehicle sub-tour and $\perp$ denotes padding. In our implementation, sub-tour selection is performed by a randomized depot anchoring rule. For each sampled rollout, we first choose one depot occurrence uniformly at random from the non-padded part of $\tau_G^\star$, and circularly roll the reference sequence so that this depot becomes the first token. Consequently, the sub-tour immediately following the chosen depot is used as the initial candidate reference sub-tour. We do not dynamically match a reference sub-tour to the remaining unvisited customers.

During decoding, rectification is applied locally through successor following. Let $\bar{\tau}_{G,j}^\star$ denote the rolled reference sequence for rollout j. Given the current node $a'_{j,t-1}$, the candidate reference action is the immediate successor of $a'_{j,t-1}$ in $\bar{\tau}_{G,j}^\star$. The action-level rectification gate is updated only when the current node is the depot: at a depot visit, a Bernoulli gate decides whether to enter a reference sub-tour; while the agent is inside a route, the previous gate value is kept, so the selected reference sub-tour is followed consecutively until the next depot unless feasibility fails. When a depot successor is rectified, the matched depot occurrence in $\bar{\tau}_{G,j}^\star$ is marked as consumed, so subsequent depot visits advance to later depot occurrences in the rolled reference sequence rather than repeatedly starting from the same sub-tour.

Conflicts with previously visited customers or capacity constraints are resolved by the environment feasibility mask. Specifically, the reference successor is executed only if it is currently unmasked, which requires the customer to be unvisited and its demand not to exceed the remaining vehicle capacity. Otherwise, rectification is bypassed at that step and the policy-sampled feasible action is used:

$$a'_{j,t} = \begin{cases} \mathrm{Succ}_{\bar{\tau}_{G,j}^\star}(a'_{j,t-1}), & M_G[j,t] = 1,\ I_G[j] = 1,\ \mathrm{Feas}_{\mathrm{CVRP}}\Big(\mathrm{Succ}_{\bar{\tau}_{G,j}^\star}(a'_{j,t-1}), s_{j,t-1}\Big) = 1, \\ a_{j,t}, & \text{otherwise.} \end{cases} \tag{17}$$

Thus, the CVRP rectifier uses randomized depot-anchored reference sub-tours, while infeasible successors caused by earlier policy decisions are handled by a step-wise fallback to the policy action, preserving autoregressive feasibility.

PCTSP. For PCTSP, the reference trajectory generally contains only a subset of customer nodes, since a feasible solution is allowed to skip nodes and pay the corresponding penalties. Therefore, the intra-instance rectifier is implemented with an explicit membership-and-feasibility fallback. Let the padded reference tour be $\tau_G^\star = (v_1^\star, \ldots, v_L^\star, \perp, \ldots, \perp)$, where $L \le M$ and $\perp$ denotes padding. At step t, after the policy proposes a feasible action $a_t \sim \pi_\theta(\cdot \mid s_{t-1}; G)$, we first locate the current node a'_{t-1} in the non-padded part of $\tau_G^\star$. If $a'_{t-1} = v_\ell^\star$ for some $\ell \le L$, the candidate reference action is defined as

$$\hat{a}_t^\star = \begin{cases} v_{\ell+1}^\star, & \ell < L, \\ 0, & \ell = L, \end{cases} \tag{18}$$

where node 0 is the depot. If the current node is absent from the reference subset, no reference successor is queried and we set $\hat{a}_t^\star = a_t$. The executed action is then

$$a'_t = \begin{cases} \hat{a}_t^\star, & M_G[j,t] = 1,\ I_G[j] = 1,\ a'_{t-1} \in \tau_G^\star,\ \mathrm{Feas}_{\mathrm{PCTSP}}(\hat{a}_t^\star, s_{t-1}) = 1, \\ a_t, & \text{otherwise.} \end{cases} \tag{19}$$

Here $\mathrm{Feas}_{\mathrm{PCTSP}}$ is exactly the environment action mask: a customer node must be unvisited, while returning to the depot is allowed only after the minimum prize requirement is satisfied, or when no further customer can be visited. Consequently, when the policy reaches a valid node that is not contained in the expert subset, rectification is dynamically bypassed and the rollout continues with the policy-selected feasible action. This preserves the autoregressive feasibility guarantee while still exploiting reference successors whenever the current partial solution intersects the reference tour.

KP. In contrast to routing problems, item selection in KP is unordered. Since the intra-instance rectifier was originally designed to better capture sequential relationships within local regions for routing problems (which is a feature irrelevant to KP), we disable the intra-instance rectification operation when training KP models. Instead, when a sampled trajectory is masked as `True` for

instance-level rectification, the entire trajectory is replaced with a randomly permuted trajectory that includes all items from the optimal reference solution.

SMTWTP. Structurally, SMTWTP is a permutation problem analogous to TSP, where any ordering of unvisited jobs constitutes a feasible solution. Consequently, we directly employ the standard intra-instance rectification strategy. This allows the model to effectively assimilate optimal sub-sequences from the reference solutions without modification.

JSSP. In contrast, JSSP imposes rigorous precedence and resource constraints; a deviation from the optimal trajectory can render subsequent expert actions infeasible. To address this, we introduce a *feasibility-guarded rectification* mechanism. Specifically, the reference action is enforced only when it satisfies the current environmental validity constraints (i.e., it is not masked). If an expert action becomes invalid due to prior state transitions, the rectification is dynamically bypassed, permitting the agent to revert to its own policy.

D Adapted Model Architectures

As described in Sec. 4.2, we employ AM [18], POMO [19], and MatNet [20] as the neural backbones for instantiating CORectifier paradigm to learn PCTSP, TSP, and ATSP, respectively, which simultaneously validates the solving performance as well as the wide applicability of our approach by adapting it to popular existing RL-based learning frameworks. Beyond the TSP family, we further develop models upon POMO to conduct the rectified RL learning on more complex tasks, i.e., CVRP, KP, SMTWTP, and JSSP. In this section, we provide a concise introduction to the layer-level neural architecture for the corresponding backbone models for your convenient reference.

D.1 Attention Model (AM)

Encoder. The encoder converts node features into embedding vectors containing global information. It adopts a multi-layer attention structure similar to the classical Transformer [71], with no positional encoding to ensure input order invariance.

Specifically, the initial embedding $h_i^{(0)}$ of node i is obtained via linear projection:

$$h_i^{(0)} = W^x x_i + b^x, \tag{20}$$

where $W^x \in \mathbb{R}^{d_h \times d_x}$ and $b^x \in \mathbb{R}^{d_h}$ are learnable parameters, and $d_h = 128$ (embedding dimension).

The encoder consists of L attention layers. Each layer comprises a Multi-Head Attention (MHA) sublayer and a node-wise Feed-Forward (FF) sublayer, with residual connections and Batch Normalization (BN) added to each sublayer. The update formula for the ℓ-th layer is:

$$\hat{h}_i = \mathrm{BN}^{\ell}\left(h_i^{(\ell-1)} + \mathrm{MHA}_i^{\ell}\left(h_1^{(\ell-1)}, ..., h_n^{(\ell-1)}\right)\right), \tag{21}$$

$$h_i^{(\ell)} = \mathrm{BN}^{\ell}\left(\hat{h}_i + \mathrm{FF}^{\ell}(\hat{h}_i)\right). \tag{22}$$

The MHA uses N_h attention heads, each with dimension $d_k = d_h/N_h = 16$. Compatibility between nodes is calculated via dot-product, and information is aggregated by weighted summation. The FF block is a single hidden layer with ReLU activation, formulated as $\mathrm{FF}(\hat{h}_i) = W^{ff,1} \cdot \mathrm{ReLU}(W^{ff,0}\hat{h}_i + b^{ff,0}) + b^{ff,1}$. The encoder ultimately outputs two types of embeddings: 1) node embeddings: $h_i^{(N)}$, and 2) graph embedding: $\overline{h}^{(N)}$ (mean of all node embeddings, containing global information):

$$\overline{h}^{(N)} = \frac{1}{n}\sum_{i=1}^{n} h_i^{(N)}. \tag{23}$$

Decoder. The decoder generates the visitation sequence step-by-step. It introduces a "context node" (c) to efficiently aggregate encoder information and partial sequence states, with two core attention computations (glimpse layer and output probability layer). The embedding $h_{(c)}^{(N)}$ of the context node integrates the graph embedding, and information about the "first node" and "last node" of the partial

sequence (to adapt to tasks like TSP that require returning to the starting point). Its formulation is:

$$h_{(c)}^{(N)} = \begin{cases} [\overline{h}^{(N)}, h_{\pi_{t-1}}^{(N)}, h_{\pi_1}^{(N)}] & t > 1 \\ [\overline{h}^{(N)}, v^l, v^f] & t = 1 \end{cases}. \tag{24}$$

For $t = 1$ (initial step), learnable placeholders $v^l, v^f \in \mathbb{R}^{d_h}$ are used; π_{t-1} is the node visited in the previous step; and $[\cdot, \cdot, \cdot]$ denotes horizontal concatenation (resulting in a dimension of $3d_h$).

The context node embedding $h_{(c)}^{(N+1)}$ is updated via MHA, with attention only computed between the context node and all other nodes (to improve efficiency):

First, compute queries (from the context node), keys, and values (from encoder node embeddings):

$$q_{(c)} = W^Q h_{(c)}, \quad k_i = W^K h_i^{(N)}, \quad v_i = W^V h_i^{(N)}, \tag{25}$$

where $W^Q, W^K \in \mathbb{R}^{Md_k \times 3d_h}$ and $W^V \in \mathbb{R}^{Md_v \times d_h}$ are parameters.

Second, compatibility calculation and masking (to block already visited nodes):

$$u_{(c)j} = \begin{cases} \frac{q_{(c)}^T k_j}{\sqrt{d_k}} & j \notin \{\pi_1, ..., \pi_{t-1}\} \\ -\infty & \text{otherwise} \end{cases}. \tag{26}$$

Third, weighted aggregation to obtain $h_{(c)}^{(N+1)}$ (following the multi-head fusion logic of MHA). A single-head attention layer converts $h_{(c)}^{(N+1)}$ into the probability of the next node to visit. First, a compatibility clipping is used to avoid numerical overflow:

$$u_{(c)j} = \begin{cases} C \cdot \tanh\left(\frac{q_{(c)}^T k_j}{\sqrt{d_k}}\right) & j \notin \{\pi_1, ..., \pi_{t-1}\} \\ -\infty & \text{otherwise} \end{cases}, \tag{27}$$

where $C = 10$ is the clipping threshold. Finally, softmax normalization is computed to get probabilities:

$$p_\theta(\pi_t = i | s, \pi_{1:t-1}) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}}. \tag{28}$$

D.2 Policy Optimization with Multiple Optima (POMO)

POMO [19] adopts the core encoder architecture from AM [18], while extending AM's training paradigm by introducing symmetric parallel trajectories to improve baseline estimation and solution space exploration.

Specifically, POMO abandons AM's trainable start token and instead generates N parallel trajectories (one per node as the initial starting point, e.g., 100 trajectories for TSP-100). This enables symmetric exploration of the solution space by covering all possible initial nodes for each instance. For trajectory i (starting at node a_1^i), the initial decoder state is directly set to the encoder's embedding of a_1^i:

$$h_{\pi_1^i} = h_{a_1^i}, \tag{29}$$

eliminating the need for AM's trainable start token parameters v^l, v^f.

For $t > 1$, the decoder's context embedding for trajectory i follows AM's design, fusing three components: the global graph embedding $\bar{h}^{(N)}$, the embedding of the last visited node $h_{\pi_{t-1}^i}$, and the embedding of the trajectory's initial node $h_{\pi_1^i}$:

$$h_{(c)}^i = \left[\bar{h}^{(N)}, h_{\pi_{t-1}^i}, h_{\pi_1^i}\right]. \tag{30}$$

To reduce computational overhead, N context embeddings are stacked into a single matrix, and AM's attention mechanism is leveraged to compute all trajectories in parallel, avoiding the inefficiency of sequential trajectory generation.

Adaptive Modification for SMTWTP and JSSP. While preserving the core encoder-decoder structure of POMO, we tailor the architecture to accommodate the complex constraints and dynamic state transitions inherent in scheduling tasks. The key modifications are twofold:

1) Augmented Context Embeddings. Unlike the static context in TSP, decision-making in scheduling relies heavily on global status and remaining workload. We enhance the decoder's query formulation by integrating three additional context vectors into the standard query q: (i) **Graph Context** ($\bar{h}^{(N)}$): The mean of all node embeddings, providing a global summary of the instance. (ii) **Pool Context**: The mean embedding of all currently *unvisited* nodes, dynamically updating at each step to reflect the remaining job pool. (iii) **Step Embedding**: A learnable embedding encoding the current time step index, aiding the model in sensing the temporal progression of the schedule. The final query is computed as:

$$q_{final} = q + W^G \bar{h}^{(N)} + W^P h_{pool} + h_{step}. \tag{31}$$

2) Bias-Enhanced Attention. To explicitly incorporate critical numerical features (e.g., due dates, processing times) that might be diluted in high-dimensional embeddings, we introduce a scalar bias mechanism. We employ MLPs to project specific features into a scalar bias, which is directly added to the attention scores (logits) before the softmax operation.

- **For SMTWTP**, we compute dynamic urgency features at step t to prioritize high-risk jobs: (1) *Slack time* ($d_i - t - p_i$), representing the remaining buffer; (2) *Potential lateness* ($\max(0, t + p_i - d_i)$), estimating immediate tardiness; (3) *Weighted lateness*, which scales the potential lateness by weight w_i.
- **For JSSP**, to capture resource bottlenecks, we extract: (1) *Processing time*; (2) *Earliest Start Time (EST)*, defined as $\max(\text{job_avail}, \text{machine_avail})$; and (3) *Wait time* ($EST -$ job_avail), quantifying delays caused specifically by machine occupancy. These features help distinguish critical path operations from those with slack.

This bias term acts as a soft constraint, steering the policy towards urgency-based or resource-efficient decisions.

D.3 Matrix Encoding Network (MatNet)

The highlight of MatNet [20] lies in the fact that it pioneers the applicability of RL-based models to learn the more complex asymmetric TSP (ATSP) without 2D-coordinates as input node features, which designs a mixed-score attention module to incorporate information from the distance matrix. As for training, MatNet follows POMO to utilize an average reward from N different samples as the REINFORCE baseline. Note that for narrative simplicity, we allow a slight overload of the notation N in this local scope to represent the number of nodes for an ATSP instance.

Encoder. The encoder models node relationships using two parallel embedding matrices ($\mathbf{A}^l, \mathbf{B}^l \in \mathbb{R}^{d \times N}$ for the l-th layer, where d denotes embedding dimension and N denotes node count) and a mixed-score attention mechanism that fuses cost matrix information:

Embedding Initialization: $\mathbf{A}^0$ is initialized as a zero embedding matrix, while $\mathbf{B}^0$ is initialized as a one-hot embedding matrix. To ensure $\mathbf{B}^0$ matches dimension d, MatNet enforces $d \geq N$ and pads a square one-hot matrix $\mathbf{OE} \in \mathbb{R}^{N \times N}$ with zeros, i.e., $\mathbf{B}^0 = [\mathbf{OE}, \mathbf{0}]^\top$ (where $\mathbf{0} \in \mathbb{R}^{N \times (d-N)}$). For MatNetPOE, $\mathbf{B}^0$ uses a position-aware one-hot embedding ($\mathbf{POE}$) to avoid padding.

Mixed-Score Attention: Taking $\mathbf{A}$-embedding update as an example, query, key, and value matrices are computed as:

$$\mathbf{Q}_a^l = \mathbf{W}_a^Q \mathbf{A}^l, \quad \mathbf{K}_a^l = \mathbf{W}_a^K \mathbf{B}^l, \quad \mathbf{V}_a^l = \mathbf{W}_a^V \mathbf{B}^l, \tag{32}$$

where $\mathbf{W}_a^Q, \mathbf{W}_a^K, \mathbf{W}_a^V \in \mathbb{R}^{d_{qkv} \times d}$ are learnable attention parameters. A $2 \to 1$-dimensional MLP (MLP_1) concatenates and mixes the cost matrix $\mathbf{C}$ with scaled dot-product attention scores ($\mathbf{Q}_a^{l\top} \mathbf{K}_a^l / \sqrt{d_{qkv}}$), generating the mixed attention score matrix:

$$\mathbf{MixedScoreAtt}_a^l = \text{softmax}\left(\text{MLP}_1\left(\left[\mathbf{C}; \frac{\mathbf{Q}_a^{l\top} \mathbf{K}_a^l}{\sqrt{d_{qkv}}}\right]\right)\right) \in \mathbb{R}^{N \times N}. \tag{33}$$

A d-dimensional MLP (MLP_2, input/output dimension d) then updates $\mathbf{A}^{l+1}$:

$$\mathbf{A}^{l+1} = \text{MLP}_2\left(\mathbf{MixedScoreAtt}_a^l \mathbf{V}_a^{l^\top}\right)^\top. \tag{34}$$

Symmetric Update for $\mathbf{B}$: $\mathbf{B}^{l+1}$ is computed symmetrically by swapping $\mathbf{A}^l$ and $\mathbf{B}^l$ and using separate parameters ($\mathbf{W}_b^Q, \mathbf{W}_b^K, \mathbf{W}_b^V$). The encoder extends this mechanism to multi-head attention (following the original Transformer [71]) for richer node representations.

Decoder. The decoder performs "rollout" to generate a tour $\tau = \{i_1, i_2, \cdots, i_N\}$ using the encoder's final $\mathbf{A}, \mathbf{B}$ embeddings:

Key/Value Initialization: Decoder key and value matrices are precomputed as $\mathbf{K}_{dec} = \mathbf{W}_{dec}^K \mathbf{B}$ and $\mathbf{V}_{dec} = \mathbf{W}_{dec}^V \mathbf{B}$, where $\mathbf{W}_{dec}^K, \mathbf{W}_{dec}^V \in \mathbb{R}^{d_{qkv} \times d}$ are trainable parameters.

Query Update: The query vector accumulates information of previously selected nodes. For the first node i_1, $\mathbf{q}_{dec}^1 = \mathbf{W}_{dec}^{Q^1} \mathbf{a}_{i_1}$; for subsequent nodes, it is updated as:

$$\mathbf{q}_{dec}^{n+1} = \mathbf{W}_{dec}^{Q^0} \mathbf{a}_{i_n} + \mathbf{q}_{dec}^1, \tag{35}$$

where $\mathbf{W}_{dec}^{Q^1}, \mathbf{W}_{dec}^{Q^0} \in \mathbb{R}^{d_{qkv} \times d}$ are learnable parameters.

Next Node Selection: A masking vector $\mathbf{Inf}_{i_{n' \le n}}$ (elements corresponding to visited nodes $i_{n'}$ set to $-\infty$, others 0) is used to avoid revisiting nodes. The output embedding of the n-th rollout iteration is:

$$\mathbf{o}_{dec}^{n+1} = \text{Linear}\left(\text{softmax}\left(\mathbf{Inf}_{i_{n' \le n}} + \frac{\mathbf{q}_{dec}^{n^\top} \mathbf{K}_{dec}}{\sqrt{d_{qkv}}}\right) \mathbf{V}_{dec}\right), \tag{36}$$

where $\text{Linear}(\cdot)$ denotes a linear layer (input dimension d_{qkv}, output dimension d). The probability of selecting the next node i_{n+1} is:

$$\mathbf{p}_\theta(i_{n+1} | i_{n' \le n}) = \text{softmax}\left(\mathbf{Inf}_{i_{n' \le n}} + \tanh\left(\frac{\mathbf{B}^\top \mathbf{o}_{dec}^{n+1}}{\sqrt{d}}\right)\right). \tag{37}$$

Multi-head attention enhances this process in practice; a complete tour is obtained after N rollout iterations.

E Detailed Experimental Settings

Training. We provide the hyper-parameters and detailed training and solving settings of our method in Table 14 and Table 15. The hyper-parameters w.r.t. the backbones are in consistency with established implementations from [19, 20, 45, 71]. Additionally, we use Adam [72] optimizer with a unified weight decay of 1e-6 for all tasks. When training POMO [19] for TSP and CVRP, we adopt a parallel sampling size (`pomo_size`) as $\min(200, M)$ with M being the number of nodes of the graph instances. Note that we have empirically justified the selection of some core parameters in Sec. 4.2 and Appendix F.4.

Table 14: Hyper-parameters and training settings of CORectifier on TSP/ATSP/PCTSP.

	TSP			ATSP				PCTSP		
	50	100	500	50	100	200	500	20	50	100
Number of pre-training epochs (IL)	200	100	100	200	100	100	100	200	200	200
Number of training epochs (RRL)	2000	10000	1000	8000	3000	3000	2000	10000	10000	10000
Samples per epoch	10000	10000	10000	10000	10000	10000	10000	6400	6400	6400
(Initial) Learning rate η	2e-4	1e-4	1e-4	4e-4	2e-4	2e-4	5e-5	1e-4	1e-4	1e-4
Training batch-size B	200	64	2	200	64	8	2	64	64	64
Max Batch-wise rectify probability p_{batch}	0.1	0.1	0.1	0.5	0.5	0.5	0.5	0.1	0.1	0.1
Max Instance-wise rectify probability p_{inst}	0.1	0.1	0.1	0.5	0.5	0.5	0.5	0.1	0.1	0.1
Max Intra-instance rectify length ratio α	0.1	0.1	0.1	0.8	0.8	0.8	0.8	0.1	0.1	0.1
Max Intra-instance rectify length ratio β	0.2	0.2	0.2	1.0	1.0	1.0	1.0	0.2	0.2	0.2

Additionally, to dynamically adjust the intensity of expert guidance during training, we adopt a separate cosine-annealing scheduler (borrowing its successful use in adjusting learning rate, etc.) for each parameter $\mathtt{p} \in \{p_{\text{batch}}, p_{\text{inst}}, \alpha, \beta\}$ with distinct cycle lengths ($T_{\max}$):

$$\mathtt{p}(t) = \mathtt{p}_{\min} + 0.5\,(\mathtt{p}_0 - \mathtt{p}_{\min})\left[1 + \cos\left(\pi t / T_{\max}\right)\right], \tag{38}$$

Table 15: Hyper-parameters and training settings of CORectifier on CVRP, KP.

	CVRP				KP			
	50	100	200	500	50	100	200	500
Number of pre-training epochs (pretrain)	1000	1000	500	100	1000	1000	500	100
Number of training epochs (RRL)	12000	12000	8000	2000	5000	5000	1000	1000
Samples per epoch	10000	10000	10000	10000	10000	10000	10000	10000
(Initial) Learning rate η	2e-4	2e-4	1e-4	1e-4	1e-4	1e-4	5e-5	5e-5
Training batch-size B	200	128	16	8	200	200	64	16
Max Batch-wise probability p_{batch}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Max Instance-wise probability p_{inst}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Max Intra-instance sub-tour probability p_{sub}	0.1	0.1	0.1	0.1	N/A	N/A	N/A	N/A

Table 16: Hyper-parameters and training settings of CORectifier on SMTWTP, JSSP.

	SMTWTP			JSSP			
	20	50	100	6×6	10×10	15×15	20×20
Number of pre-training epochs (pretrain)	200	200	200	300	300	300	300
Number of training epochs (RRL)	1000	1000	1000	2000	2000	2000	2000
Samples per epoch	10000	10000	5000	10000	10000	5000	5000
(Initial) Learning rate η	2e-4	2e-4	1e-4	2e-4	2e-4	1e-4	1e-4
Training batch-size B	200	200	100	100	200	200	100
Max Batch-wise probability p_{batch}	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Max Instance-wise probability p_{inst}	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Max Intra-instance sub-tour probability p_{sub}	0.1	0.1	0.1	0.2	0.2	0.2	0.2

where p_{min} is the lower bound for the scheduled parameter, p_0 is the initial value, and t is the training epoch. The independent cosine-periodic updates generate diverse combinations of expert guidance modes across training, exposing the model to varying degrees of reference reliance (e.g., high p_{batch} with low β, or low p_{inst} with high α) over epochs, enhancing the robustness of the learned policy. We set the period ($T_{\max}$) for the independent cosine schedulers for the rectifier-related probabilities as 10, 20, 40, 40, 40 for p_{batch}, p_{inst}, α, β, p_{sub} where applicable, respectively.

Evaluation. 1) The OBJ. and DROP Metrics. Our evaluation and reporting protocol follows a broad line of prior NCO literature [13, 16, 23, 33, 34], where DROP is computed by first measuring the per-instance relative gap and then averaging over all n test instances. Here, Obj_i denotes the objective value returned by a method on test instance i, and Obj_i^* denotes the objective value of the corresponding reference solution. Note that following standard practice in NCO, the reference solution is obtained by a strong heuristic or exact solver, depending on the task and benchmark, and therefore is not necessarily guaranteed to be globally optimal in every case. As a result, negative DROP values may occur when a neural method finds a solution better than the reference one.

Since the covered tasks in this paper include both *minimization* and *maximization* problems, the per-instance DROP is defined case by case. For minimization tasks (TSP, ATSP, PCTSP, CVRP, JSSP, SMTWTP), we compute

$$drop_i = \frac{Obj_i - Obj_i^*}{Obj_i^*} \times 100\%, \tag{39}$$

whereas for the maximization task (KP), we compute

$$drop_i = \frac{Obj_i^* - Obj_i}{Obj_i^*} \times 100\%. \tag{40}$$

We then report

$$\text{DROP} = \frac{1}{n}\sum_{i=1}^{n} drop_i. \tag{41}$$

This is different from first averaging OBJ. across instances and then computing a single relative gap against the averaged reference objective. We do not use an absolute-value formulation, since doing so would remove the sign information and fail to reflect cases where a method surpasses the reference solution. On the other, the reported OBJ. metric is the average objective value over the n test instances, i.e.,

$$\text{OBJ.} = \frac{1}{n}\sum_{i=1}^{n} Obj_i. \tag{42}$$

2) Decoding and Post-processing. We adopt the standard decoding and post-processing implementations, including greedy decoding and local search, provided by ML4CO-Kit[6] (version: 0.3.3), an open-source Python package for reproducible and fair NCO evaluation. In particular, the underlying 2OPT implementation is written in C, so its additional computational overhead is negligible at small problem scales, which is consistent with standard evaluation practice.

F Supplementary Results

F.1 Computational Overhead for Training

As shown in Table 17, CORectifier introduces extra computation for mask construction and rectification, but the overall training overhead remains moderate in most settings. For TSP, RRL has almost identical memory consumption to the RL baseline, while increasing the per-epoch training time by only 1.7%, 4.0%, and 2.2% at scales 50, 100, and 500, respectively. For CVRP, the overhead is also moderate, with per-epoch time increasing by 20.0% and 18.6% at scales 50 and 500, while being comparable at scale 100. For ATSP, the additional rectification cost is more visible, especially at scales 50 and 100, but the memory footprint remains nearly unchanged for small and medium scales and increases only at scale 500.

The convergence-time results further show that the additional wall-clock cost is generally within a reasonable range compared with the corresponding RL baselines. For example, TSP-RRL increases convergence time by 9.7% and 29.8% over TSP-RL at scales 50 and 100, respectively, while enabling training at scale 500 where the corresponding RL baseline is unavailable. Similarly, CVRP-RRL and ATSP-RRL incur additional convergence time due to rectification, but they provide a consistent mechanism for improving solution quality across problem types and scales. Overall, these results suggest that CORectifier trades a modest amount of additional computation for improved optimization performance, and the overhead is justified by the solution-quality gains reported in the main experiments.

Table 17: Training overhead and convergence-time comparison. Left: GPU memory consumption and average training time per epoch under identical batch size, number of episodes per epoch, and hardware settings. Right: wall-clock time to convergence measured from training logs.

(a) Per-epoch training overhead

Method	50	100	500
TSP-RL	5.67 GB / 7.46 s	8.85 GB / 35.33 s	18.28 GB / 263.86 s
TSP-RRL	5.67 GB / 7.59 s	8.85 GB / 36.74 s	18.28 GB / 269.77 s
CVRP-RL	3.64 GB / 7.35 s	11.82 GB / 24.17 s	10.62 GB / 3007.79 s
CVRP-RRL	4.30 GB / 8.82 s	11.35 GB / 23.99 s	10.58 GB / 3567.98 s
ATSP-RL	12.39 GB / 18.50 s	17.07 GB / 70.04 s	16.56 GB / 1138.83 s
ATSP-RRL	12.38 GB / 29.84 s	17.06 GB / 127.91 s	17.99 GB / 1363.03 s

(b) Time to convergence

Method	50	100	500
TSP-SL [15]	112.8 h	448.2 h	142.3 h
TSP-RL [19]	18.5 h	151.2 h	–
TSP-RRL (ours)	20.3 h	196.2 h	62.7 h
CVRP-RL [19]	113.5 h	169.7 h	–
CVRP-RRL (ours)	129.3 h	210.1 h	1165.0 h
ATSP-RL [20]	87.6 h	60.7 h	–
ATSP-RRL (ours)	111.2 h	107.1 h	489.5 h

Note. For panel (a), each entry reports GPU memory usage and average training time per epoch in the format "memory / time". All measurements are obtained under the same batch size, number of episodes per epoch, and hardware configuration as described in Appendix **E**. The notation "–" indicates that the corresponding baseline was not applicable at that scale. Also, the time to convergence has included durations for, e.g., data I/O and pre-processing, single batch-sized validation after every training epoch, logging, checkpoint saving, etc., in hope to provide more realistic statistics for the experiments, rather than accumulating only the raw training times.

F.2 Supplementary Generalization Studies

F.2.1 Full Results on TSPLIB

Following [11, 14, 16, 21], we present the per-instance solving result (optimality drop) on the real-world TSPLIB dataset (zero-shot generalization) in Table 18, comparing CORectifier with the most

[6] https://pypi.org/project/ml4co-kit (Licensed under Mulan PSL v2)

commonly quoted results as well as former RL-based approaches. The CORectifier model evaluated is trained on uniform TSP instances with 100 nodes.

Table 18: Per-instance optimality drop for methods trained on uniform TSP-100 and evaluated on TSPLIB instances with 50-200 nodes (29 instances). The comparison partially includes baselines in Table 3 in [11]. We normalize the 2D coordinates for consistent comparison following [16].

INSTANCES	AM [18]	POMO [19]	GCN [10]	Learn2OPT [64]	GNNGLS [11]	Ours
eil51	16.767%	0.719%	40.025%	1.725%	1.529%	0.057%
berlin52	4.169%	0.004%	33.225%	0.449%	0.142%	0.004%
st70	1.737%	0.013%	24.785%	0.040%	0.764%	0.000%
eil76	1.992%	0.004%	27.411%	0.096%	0.163%	0.613%
pr76	0.816%	0.000%	27.793%	1.228%	0.039%	0.000%
rat99	2.645%	6.469%	17.633%	0.123%	0.550%	1.581%
kroA100	4.017%	3.247%	28.828%	18.313%	0.728%	0.412%
kroB100	5.142%	2.817%	34.686%	1.119%	0.147%	0.912%
kroC100	0.972%	1.377%	35.506%	0.349%	1.571%	1.417%
kroD100	2.717%	1.995%	38.018%	0.866%	0.572%	1.361%
kroE100	1.470%	3.088%	26.589%	1.832%	1.216%	0.811%
rd100	3.407%	0.000%	50.432%	1.725%	0.003%	0.000%
eil101	2.994%	0.060%	26.701%	0.387%	1.529%	1.069%
lin105	1.739%	1.393%	34.902%	1.867%	0.606%	0.756%
pr107	3.933%	1.344%	80.564%	0.898%	0.439%	0.504%
pr124	3.677%	1.052%	70.146%	10.322%	0.755%	0.782%
bier127	5.908%	6.009%	45.561%	3.044%	1.948%	1.387%
ch130	3.182%	0.663%	39.090%	0.709%	3.519%	0.854%
pr136	5.064%	1.040%	58.673%	0.000%	3.387%	1.252%
pr144	7.641%	1.129%	55.837%	1.526%	3.581%	0.504%
ch150	4.584%	0.546%	49.743%	0.312%	2.113%	0.846%
kroA150	3.784%	3.034%	45.411%	0.724%	2.984%	1.118%
kroB150	2.437%	2.968%	56.745%	0.886%	3.258%	1.469%
pr152	7.494%	2.534%	33.925%	0.029%	3.119%	0.295%
u159	7.551%	1.245%	38.338%	0.054%	1.020%	1.081%
rat195	6.893%	9.623%	24.968%	0.743%	1.666%	4.923%
d198	373.020%	32.555%	62.351%	0.522%	4.772%	2.461%
kroA200	7.106%	3.563%	40.885%	1.441%	2.029%	1.806%
kroB200	8.541%	4.372%	43.643%	2.064%	2.589%	2.096%
Mean	16.767%	3.202%	40.025%	1.725%	1.529%	**1.049%**

F.2.2 Full Results on CVRPLIB

More Detailed Results and Analysis on CVRPLIB. Table 19 gives the detailed comparisons following PO [26], which reports the zero-shot generalization results on the entire CVRPLIB (Set-X) set, in split of 22 small instances and 78 large instances. Table 21 gives the per-instance solving quality on CVRPLIB (Set-X) with 100-512 client nodes. Note that the "ML4CO-Best" results are selected from the best performance of Sym-NCO models trained on uniform CVRP-{100, 200, 500} data, as reported by the authors of [34] with performance enhancements over the original implementation [21], while our model is merely trained on CVRP-100 data without such best-picking trick among candidate models. Settings: Sym-NCO: $N\times$ Sampling; CORectifier: $\mathrm{N_A} = 16$; both are with classic local search (LS) enabled.

F.2.3 Results on CVRP-1K to CVRP-5K

We train CORectifier on uniformly generated CVRP instances of sizes 100-500 and evaluate it without further training on CVRP-1K, CVRP-2K, and CVRP-5K. Table 20 includes representative scalable NCO baselines reported in recent work under comparable data distributions and inference budgets. A dagger denotes a result quoted from the corresponding paper; quoted runtimes are retained for context but are not treated as hardware-matched measurements. All entries use greedy decoding (G).

CORectifier obtains the lowest reported gap among the listed methods on CVRP-1K, remains competitive on CVRP-2K, and is not uniformly best on CVRP-5K, where UDC attains the lowest gap. These results provide evidence of meaningful zero-shot large-scale generalization using a basic autoregressive backbone, while also showing the limits of the current approach. In particular, the maximum practical scale is governed not only by the rectification operator but also by the computational and memory complexity of the underlying attention-based constructive policy.

Table 19: Detailed comparative results on CVRPLIB-Set-X benchmarks (100 instances in total, 22 within (100, 200] clients and 78 within (203, 1000] clients). Data compared are quoted from one of the latest works, i.e., PO [26], in the RL4CO field.

SOLVER	TYPE	CVRPLIB			
		(0, 200] DROP	(200, 1000] DROP	Total DROP	TIME
LKH3 [52]	Heuristic	0.36%	1.18%	1.00%	16m
HGS [60]	Heuristic	0.01%	0.13%	0.11%	16m
NeuroLKH [73]	Heuristic+SL	0.47%	1.16%	0.88%	16m
LEHD [65]	SL	11.11%	12.73%	12.25%	1.67s
BQ-NCO [43]		10.60%	10.97%	10.89%	3.36s
POMO [19]	RL	5.26%	11.82%	10.37%	0.80s
Sym-NCO [21]		9.99%	27.09%	23.32%	0.87s
Omni-POMO [58]		5.04%	6.95%	6.52%	0.75s
ELG (RF) [66]		4.51%	6.46%	6.03%	1.90s
ELG (PO) [26]		4.39%	6.37%	5.94%	1.90s
CORectifier	IL+RRL	**3.49%**	**5.98%**	**5.47%**	2.11s

Table 20: Zero-shot generalization to large-scale CVRP. Each entry reports gap/time; lower is better for both quantities. A dagger marks quoted baseline results, and "–" denotes an unavailable result.

METHOD	CVRP-1K	CVRP-2K	CVRP-5K
GLOP (LKH-3) [44], G†	11.40% / 1.1s	5.01% / 1.6s	11.40% / 4s
UDC ($\alpha = 50$) [74], G†	7.18% / 1.3s	8.75% / 3s	**2.99%** / 1m
ICAM [63], G†	5.07% / 0.1s	–	–
LEHD [65], G†	6.80% / 0.8s	–	9.51% / 1.4m
BQ-NCO [43], G†	7.28% / 1s	–	5.37% / 12m
INViT-3V [62], G†	17.23% / 1s	–	15.85% / 7s
L2C-Insert [61], G†	8.01% / 0.7s	–	–
L2R [75], G†	11.37% / 0.1s	–	7.72% / 0.5s
CORectifier, G	**3.58%** / 0.8s	**6.37%** / 2.8s	7.02% / 0.9m

F.3 Extended Ablation Studies

F.3.1 Results and Clarifications on the Rectification Levels

By design, the three rectification levels operate in a **cascaded and progressively finer-grained** manner, where each level gates how much rectification is applied. Specifically, rectification is first enabled at the batch level, then selectively applied to individual trajectories at the instance level, and finally localized along the trajectory at the intra-instance level. Because these levels form **a chain of gating mechanisms**, setting the probability of any level to zero effectively disables rectification entirely. This design intentionally prevents expert supervision from dominating training and helps preserve a healthy balance between guidance and free RL exploration.

To assess the contribution of each level, we conduct ablations by **fully loosening** the corresponding gate, i.e., setting $p_{\text{batch}} = 1$, $p_{\text{inst}} = 1$, or the intra-instance rectification probability to 1, while keeping all other hyperparameters fixed. This removes the filtering effect at the chosen level and allows us to analyze its role in training. Experiments are performed on TSP-100 and ATSP-100 for 100 epochs under identical settings.

Table 22: Results on TSP-100.

SETTING	OBJ.↓	DROP↓
w/o batch level	8.080	4.183%
w/o instance level	8.061	3.937%
w/o intra-instance level	8.077	4.134%
Full model	**8.039**	**3.655%**

Table 23: Results on ATSP-100.

SETTING	OBJ.↓	DROP↓
w/o batch level	1.714	9.422%
w/o instance level	1.751	11.804%
w/o intra-instance level	1.729	10.374%
Full model	**1.668**	**6.504%**

Table 21: Per-instance comparison with SOTA generalization results from the latest ML4CO Benchmark [34] on 70 CVRPLIB-Set-X instances with 100–512 client nodes.

INSTANCE	OPTIMAL	ML4CO-Bench-101 (Best)		CORectifier	
		OBJ.↓	DROP↓	OBJ.↓	DROP↓
X-n101-k25	27.905	30.563	9.524%	29.465	5.588%
X-n106-k14	26.655	27.105	1.668%	27.074	1.570%
X-n110-k13	14.979	15.097	0.784%	14.982	0.014%
X-n115-k10	12.933	13.574	4.957%	13.536	4.663%
X-n120-k6	13.437	13.611	1.296%	13.857	2.089%
X-n125-k30	56.164	59.041	5.122%	60.901	5.254%
X-n129-k18	29.005	29.356	1.211%	29.454	0.734%
X-n134-k13	11.351	11.633	2.479%	11.547	1.729%
X-n139-k10	13.678	13.804	0.920%	13.835	1.151%
X-n143-k7	15.776	16.533	4.798%	15.842	0.421%
X-n148-k46	43.671	45.190	3.478%	44.615	2.161%
X-n153-k22	21.572	28.809	33.547%	23.920	10.882%
X-n157-k13	17.572	17.708	0.775%	17.702	0.740%
X-n162-k11	14.412	14.705	2.029%	14.583	1.183%
X-n167-k10	20.766	21.244	2.303%	21.032	1.177%
X-n172-k51	46.974	59.677	27.041%	49.557	5.497%
X-n176-k26	47.816	51.999	8.748%	53.496	11.767%
X-n181-k23	26.809	27.067	0.963%	27.179	1.381%
X-n186-k15	24.251	24.720	1.933%	24.886	2.618%
X-n190-k8	17.037	17.418	2.238%	17.496	2.386%
X-n195-k51	44.274	45.772	3.385%	47.080	6.339%
X-n200-k36	60.084	61.834	2.914%	63.205	5.196%
X-n204-k19	19.588	19.967	1.936%	20.029	2.250%
X-n209-k16	30.691	31.107	1.355%	31.484	2.582%
X-n214-k11	10.879	11.278	3.665%	11.159	2.572%
X-n219-k73	117.601	124.696	6.033%	124.616	5.435%
X-n223-k34	40.478	41.308	2.051%	41.429	2.347%
X-n228-k23	25.747	26.867	4.350%	27.699	7.581%
X-n233-k16	19.297	19.731	2.249%	20.189	4.623%
X-n237-k14	27.078	27.397	1.178%	27.742	1.940%
X-n242-k48	82.757	84.427	2.018%	85.339	2.501%
X-n247-k50	37.966	55.808	46.994%	41.499	9.305%
X-n251-k28	38.885	39.798	2.348%	39.734	2.185%
X-n256-k16	18.942	19.379	2.306%	19.483	2.854%
X-n261-k13	26.591	27.586	3.741%	27.323	2.030%
X-n266-k58	75.489	78.007	3.336%	81.174	7.530%
X-n270-k35	35.432	36.133	1.978%	36.421	2.789%
X-n275-k28	22.924	23.488	2.464%	23.433	2.220%
X-n280-k17	33.503	34.597	3.265%	34.462	2.758%
X-n284-k15	20.421	21.252	4.071%	21.026	2.963%
X-n289-k60	95.35	103.478	8.523%	101.154	6.086%
X-n294-k50	47.315	48.783	3.102%	51.678	9.221%
X-n298-k31	34.247	35.464	3.553%	35.614	3.993%
X-n303-k21	21.961	22.659	3.179%	22.491	2.467%
X-n308-k13	25.980	26.723	2.859%	28.104	8.173%
X-n313-k71	94.437	103.575	9.676%	100.691	6.633%
X-n317-k53	78.456	86.456	10.198%	88.001	12.055%
X-n322-k28	29.846	30.590	2.492%	30.886	3.483%
X-n327-k20	27.681	28.546	3.127%	28.762	3.904%
X-n331-k15	31.155	31.993	2.690%	31.783	2.021%
X-n336-k84	139.283	157.244	12.896%	148.214	6.113%
X-n344-k43	42.194	43.225	2.442%	43.560	3.243%
X-n351-k40	25.948	27.200	4.822%	28.216	8.919%
X-n359-k29	51.512	52.817	2.535%	53.122	3.023%
X-n367-k17	24.741	25.711	3.920%	25.324	2.354%
X-n376-k94	147.881	148.786	0.612%	149.029	0.777%
X-n384-k52	66.086	67.689	2.426%	70.786	7.123%
X-n393-k38	38.386	39.702	3.427%	39.847	3.805%
X-n401-k29	66.204	67.696	2.253%	67.577	1.817%
X-n411-k19	19.741	20.966	6.210%	20.474	3.766%
X-n420-k130	108.137	134.019	23.935%	116.761	7.975%
X-n429-k61	65.777	67.992	3.368%	67.791	3.140%
X-n439-k37	36.484	38.941	6.735%	39.588	8.509%
X-n449-k29	55.282	56.635	2.447%	59.518	7.620%
X-n459-k26	25.279	26.603	5.236%	26.423	4.534%
X-n469-k138	222.364	254.665	14.526%	253.272	13.944%
X-n480-k70	89.842	92.585	3.053%	92.446	2.907%
X-n491-k59	66.591	68.596	3.010%	69.526	4.452%
X-n502-k39	69.262	74.836	8.049%	76.976	11.150%
X-n513-k21	24.273	25.259	4.061%	25.236	3.964%
mean	**45.183**	**48.263**	**5.469%**	**47.748**	**4.440%**

Across both benchmarks, removing the gating effect at any level leads to consistent performance degradation. These results confirm that the three rectification levels play *dependent and complementary* roles, and that the hierarchical design is essential for maintaining training stability, balancing expert guidance with exploration, and achieving optimal performance.

F.3.2 Results on Learning Paradigms

Fig. 6 shows the comparative training dynamics for the four ways for training PCTSP-100, which is supplementary to Fig. 2 for ATSP. Further, Table 24 provides numerical comparison for TSP, ATSP, and PCTSP results in this regard. Note we conduct these ablations with 200 epochs for each setting to avoid costly full-scale retrain, which aligns with mainstream ML practices and is sufficient to demonstrate the trend of the divergence among paradigms.

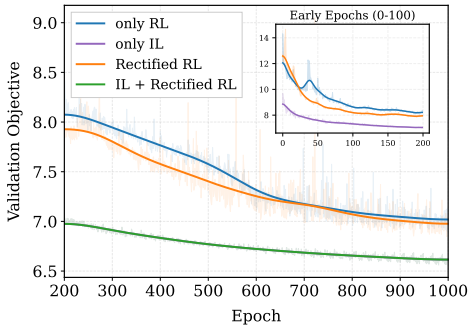


Figure 6: Train curves on PCTSP-100 (Greedy).

Table 24: Ablation results (Drop ↓) on 50-node instances.

TASK	TSP	ATSP	PCTSP
vanilla RL	0.197%	1.639%	3.754%
vanilla IL	1.818%	2.373%	4.020%
ours w/o IL	0.191%	0.853%	**3.407%**
ours w/ IL	**0.186%**	**0.649%**	3.593%

F.3.3 Mechanism Isolation under a Matched Guidance Budget

We further isolate the effect of the rectification design under a matched expected pre-feasibility expert-action budget. All variants use the same architecture, training instances, optimizer, seed, 200-epoch update budget, and decoding protocol. Models are trained on 100-node instances and evaluated by greedy decoding without augmentation.

Let p_b and p_i denote the scheduled batch- and instance-level probabilities, and let $K-1$ be the number of rectified action positions in a segment. For an n-node problem, RRL has expected proposal rate $p_b p_i (K-1)/(n-1)$. The *one-global-gate* variant applies a single $\mathrm{Bernoulli}(p_b p_i)$ gate to the same contiguous $K-1$ positions. The *independent-actions* variant replaces the contiguous mask with independent masks of probability $(K-1)/(n-1)$. The *arbitrary-reference-actions* variant retains the scheduled masks and feasibility checks but does not require the proposed reference action to be the successor of the currently executed action.

We additionally record how much scheduled guidance is actually executed. Each trajectory–step pair is treated as a potential intervention opportunity. A *proposal* is counted when the scheduled gates select that pair before the feasibility check; a *realized rectification* is counted only when the proposed reference action is feasible and is executed. Counts are accumulated over the complete training run and divided by 200 to obtain the average per epoch. The realization rate is

$$\text{Realization Rate} = \frac{N_{\text{realized}}}{N_{\text{proposal}}} \times 100\%. \tag{43}$$

This rate diagnoses proposal feasibility, rather than guidance quality, so a higher value need not imply a better final solution.

Gate factorization. On TSP, RRL and the one-global-gate variant have similar proposal counts, realization rates, and gaps (2.33% versus 2.41%); thus, gate factorization is not uniformly indispensable. On ATSP, however, the two variants again have similar intervention statistics, while the gap increases from 4.10% to 8.06%. This indicates that matching marginal intervention intensity and segment positions does not make the procedures equivalent: factorization also changes how interventions are grouped across minibatches and trajectories.

Contiguous versus independent actions. Independent replacements reduce the realization rate from 69.43% to 52.49% on TSP and from 64.95% to 46.84% on CVRP, because scattered interventions

Table 25: Mechanism isolation on 100-node instances under a matched expected pre-feasibility guidance budget. Proposal and realized counts are averages per training epoch. All models use greedy decoding without augmentation. Lower objective and gap are better; higher realization rate means that a larger fraction of proposed actions pass the feasibility check.

Task	Mechanism	Obj.↓	Gap (%)↓	Proposal/epoch	Realized/epoch	Rate (%)↑
TSP	RRL (ours)	**7.94**	**2.33**	18,042	12,526	69.43
TSP	One global gate	7.95	2.41	18,234	12,687	69.58
TSP	Independent actions	7.97	2.79	16,217	8,513	52.49
TSP	Arbitrary reference actions	7.95	2.56	16,169	14,987	92.69
ATSP	RRL (ours)	**1.63**	**4.10**	2,760,170	2,519,950	91.30
ATSP	One global gate	1.69	8.06	2,727,950	2,497,341	91.55
ATSP	Independent actions	1.70	8.30	2,667,695	2,435,999	91.31
ATSP	Arbitrary reference actions	2.19	34.35	2,730,508	1,586,653	58.11
CVRP	RRL (ours)	**16.26**	**4.46**	149,191	96,897	64.95
CVRP	One global gate	16.30	4.70	123,770	90,967	73.50
CVRP	Independent actions	16.27	4.55	181,485	85,012	46.84
CVRP	Arbitrary reference actions	16.38	5.24	128,961	73,489	56.99

can move a rollout away from the reference ordering and make later successors unavailable. More importantly, the ATSP independent-actions variant has almost the same realization rate as RRL (91.31% versus 91.30%), but its gap is 8.30% rather than 4.10%. Feasibility and intervention quantity alone therefore do not explain the difference; the result is consistent with contiguous successor segments better preserving directed edge structure.

Successor-following versus arbitrary reference actions. On TSP, arbitrary reference actions have the highest realization rate (92.69%) but a larger gap than RRL (2.56% versus 2.33%). The contrast is strongest on ATSP, where removing successor consistency increases the gap to 34.35%. CVRP exhibits the same but milder pattern (5.24% versus 4.46%). Thus, a feasible action drawn from a high-quality reference is not automatically useful outside its successor context. Overall, RRL obtains the lowest gap in all three controlled comparisons without maximizing either the proposal count or realization rate. Because several TSP and CVRP margins are modest, we interpret these results as evidence that guidance quality cannot be reduced to guidance quantity, rather than as universal dominance of every design choice.

F.3.4 Comparison with Alternative Expert-Guidance Paradigms

We next test whether RRL can be replaced by simpler uses of the same reference solutions. All controls use the same architecture, data, optimizer, 200-epoch update budget, and greedy decoding without augmentation.

For *Joint BC+RL*, each labeled minibatch contains an ordinary POMO rollout and a complete teacher-forced reference rollout, followed by the combined update

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{POMO-RL}} + \beta \mathcal{L}_{\text{BC}}, \qquad \mathcal{L}_{\text{BC}} = -\sum_t \log \pi_\theta(a_t^* \mid s_t^*, G), \tag{44}$$

where $\beta = 1$. This objective increases the likelihood of every reference action independently of its realized quality relative to the current policy.

For *full-trajectory reward-weighted imitation* (RWI), we remove the RL loss and estimate a stop-gradient within-instance baseline $\widehat{J}_\pi(G)$ from $P = 100$ POMO rollouts. The reference trajectory is assigned the detached signed advantage $\widehat{A}(\tau^*) = R(\tau^*) - \widehat{J}_\pi(G)$ and optimized using

$$\mathcal{L}_{\text{RWI}} = -\widehat{A}(\tau^*) \sum_t \log \pi_\theta(a_t^* \mid s_t^*, G). \tag{45}$$

RWI therefore applies one trajectory-level weight to the complete reference, whereas RRL updates on partially rectified behavior trajectories containing only state-feasible successor segments.

For the IL-pretrained comparison, both variants start from the same checkpoint obtained after 100 epochs of IL pretraining. We load only the model weights and use freshly initialized optimizer and

scheduler states for an identical 100-epoch post-IL phase. *IL-then-RL* applies ordinary POMO RL without further expert intervention, whereas *IL-then-RRL* enables rectification during the post-IL phase.

Table 26: Comparison with alternative expert-guidance paradigms. All entries report objective/gap (%) on 100-node instances under greedy decoding without augmentation. Lower is better.

Learning paradigm	TSP-100	ATSP-100	CVRP-100
Trained from scratch (200 epochs)			
Joint BC + RL	7.97 / 2.80	1.69 / 7.93	16.31 / 4.80
Full-trajectory RWI	7.96 / 2.57	1.66 / 6.01	16.47 / 5.81
RRL (ours)	**7.94 / 2.33**	**1.63 / 4.10**	**16.26 / 4.46**
Shared IL initialization (100 IL + 100 post-IL epochs)			
IL-then-RL	7.92 / 2.10	1.67 / 6.87	16.30 / 4.78
IL-then-RRL (ours)	**7.87 / 1.68**	**1.63 / 4.05**	**15.96 / 2.58**

Both non-rectifying from-scratch controls consume complete reference trajectories throughout training, whereas RRL intervenes only at a subset of action steps. Nevertheless, RRL gives the lowest gap on all three tasks. The TSP margin and the RRL-versus-Joint-BC+RL margin on CVRP are modest, showing that simpler imitation remains competitive in some settings; the clearer ATSP separation is consistent with coherent, state-feasible guidance being more useful under directional costs. Full-trajectory RWI is also competitive on TSP, supporting the connection between the implemented update and advantage-weighted likelihood optimization, while showing that action-level, state-dependent rectification provides additional flexibility.

The shared-IL comparison separates rectification from supervised initialization. Replacing ordinary RL with RRL during the matched post-IL phase reduces the gap from 2.10% to 1.68% on TSP, from 6.87% to 4.05% on ATSP, and from 4.78% to 2.58% on CVRP. These results do not establish universal superiority, but they show that the observed gains are not explained solely by access to reference labels, IL initialization, or generic full-trajectory advantage weighting.

F.4 Extended Results on the Effect of Hyper-parameters

The above results support our selection of the probability parameters for our framework. An observation is that ATSP is better suited with relatively larger rectification levels. We speculate this quantitative difference from those of TSP and PCTSP is that learning from solely distance matrices with inherent asymmetry is more difficult for current sequential models as they basically benefit from sampling multiple symmetric (equivalent) solution trajectories, thus demanding a deeper involvement of expert guidance.

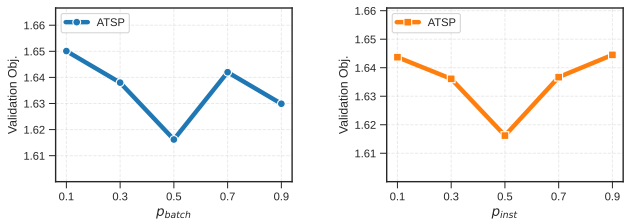


Figure 7: Effect of p_{batch} and p_{inst} on MatNet (for ATSP).

Table 27: Effect of the length range (α, β) of rectifiers on MatNet (for ATSP).

$[\alpha, \beta]$	OBJ. (ATSP-50)
$[0.0, 0.2]$	1.6703
$[0.2, 0.4]$	1.6491
$[0.4, 0.6]$	1.6448
$[0.6, 0.8]$	1.6379
$[0.8, 1.0]$	**1.6162**

Results on Smaller Rectification Probabilities. As clarified in the main paper, we explicitly avoid setting the rectification hyperparameters p_{batch}, p_{inst}, α, or β to values smaller than 0.1, as doing so would progressively degenerate the training process toward a vanilla RL paradigm. Moreover, by design, CORectifier is guaranteed to degrade gracefully and remain upper-bounded by vanilla RL performance as the amount or quality of expert guidance decreases.

To further validate this claim empirically, we conduct additional experiments on TSP-50 (consistent with the settings reported in the main paper), training all variants for 100 epochs while sweeping

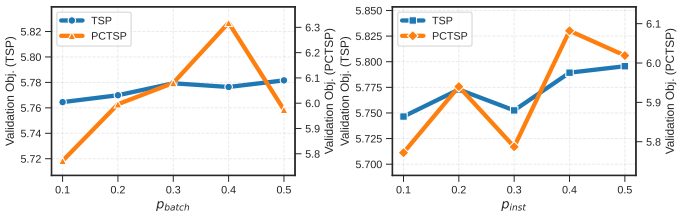


Figure 8: Effect of p_{batch} and p_{inst} on POMO (for TSP) and AM (for PCTSP).

Table 28: Effect of the length range (α, β) of rectifiers on POMO (for TSP) and AM (for PCTSP).

$[\alpha, \beta]$	TSP-50	PCTSP-50
$[0.1, 0.2]$	5.770	5.678
$[0.2, 0.3]$	5.778	5.671
$[0.3, 0.4]$	5.799	5.732
$[0.4, 0.5]$	5.822	5.722
$[0.5, 1.0]$	5.782	5.688

rectification probabilities below 0.1 at different hierarchical levels. Specifically, we vary the batch-level probability p_{batch}, the instance-level probability p_{inst}, and the intra-instance rectification interval $[\alpha, \beta]$ independently.

Table 29: Effect of small rectification probabilities on TSP-50.

p_{batch}	Obj. ↓	p_{inst}	Obj. ↓	$[\alpha, \beta]$	Obj. ↓
0.50	5.782	0.50	5.796	[0.1, 0.2]	**5.770**
0.40	5.776	0.40	5.789	[0.2, 0.3]	5.778
0.30	5.779	0.30	5.753	[0.3, 0.4]	5.799
0.20	5.770	0.20	5.773	[0.4, 0.5]	5.822
0.10	**5.765**	0.10	**5.747**	[0.5, 1.0]	5.782
0.08	5.769	0.08	5.750	[0.08, 0.10]	5.754
0.06	5.766	0.06	5.749	[0.06, 0.08]	5.753
0.04	5.769	0.04	5.750	[0.04, 0.06]	5.760
0.02	5.770	0.02	5.751	[0.02, 0.04]	5.755

The results in Table 29 confirm several consistent trends: (1) performance peaks when rectification probabilities are around 0.1, (2) further reducing the probabilities leads to only *minor and smooth degradation*, and (3) extremely small values (e.g., < 0.1) provide insufficient expert guidance to produce observable gains. These findings align well with our theoretical analysis and design principle, demonstrating that CORectifier behaves stably and degrades gracefully toward vanilla RL when rectification becomes sparse.

F.5 Supplementary Statistical Studies

F.5.1 Instance-level Solving Stability

Variance of the Optimality Drops. We offer the complete results (Table 30) of standard deviation of performance drops on the 18 evaluated benchmarks. We use "Greedy+2Opt" setting for TSP and ATSP, "Greedy+LS" for CVRP, and raw Greedy decoding for PCTSP and KP. The results validate the solving stability of CORectifier despite the inherent discrepancy among instance-wise difficulties.

Learning with Fewer Reference Solutions. As shown in Table 11, we evaluate representative benchmarks, including TSP-100, ATSP-100, and CVRP-100, under varying (much fewer than standard) amounts of expert data: **1.28k**, **12.8k**, and **128k** trajectories. All models are trained for 200 epochs with all other hyperparameters fixed to their default values.

Across all benchmarks, two consistent trends emerge. First, increasing the amount of expert data leads to modest performance improvements, mainly by stabilizing early training. Second, even with only **1/100** of the expert trajectories, performance degradation is minimal and no training collapse is observed. These results demonstrate that CORectifier is robust to the quantity of expert data and that its performance gains primarily stem from the rectified learning mechanism itself rather than from large-scale expert supervision.

Table 30: Full results of performance drops with respective standard deviations.

TASK	DROP±STD.	TASK	DROP±STD.
TSP-50	0.132±0.281%	CVRP-50	0.984±0.900%
TSP-100	0.664±0.604%	CVRP-100	1.496±0.724%
TSP-500	3.593±0.601%	CVRP-200	1.929±0.581%
ATSP-50	0.585±0.674%	CVRP-500	2.891±0.527%
ATSP-100	2.394±1.024%	KP-50	0.013±0.046%
ATSP-200	4.638±0.921%	KP-100	0.012±0.029%
ATSP-500	7.510±0.784%	KP-200	0.009±0.017%
PCTSP-20	0.504±1.108%	KP-500	0.007±0.011%
PCTSP-50	1.445±1.328%		
PCTSP-100	3.775±1.964%		

F.5.2 Seed-level Training Variability

To distinguish training variability from instance-level dispersion, we conduct a controlled 200-epoch study with three independent training seeds. Table 31 reports the mean and standard deviation across the three independently trained checkpoints.

Table 31: Seed-level results over three independent 200-epoch training runs. Values are mean ± standard deviation across training seeds. Lower is better.

	RRL		Vanilla RL	
Task	Obj.↓	Gap (%)↓	Obj.↓	Gap (%)↓
TSP-100	$\mathbf{7.937 \pm 0.007}$	$\mathbf{2.335 \pm 0.091}$	7.964 ± 0.024	2.690 ± 0.307
CVRP-100	$\mathbf{16.257 \pm 0.016}$	$\mathbf{4.462 \pm 0.104}$	16.284 ± 0.019	4.639 ± 0.120
PCTSP-100	$\mathbf{7.496 \pm 0.330}$	$\mathbf{25.421 \pm 5.534}$	7.998 ± 0.358	33.812 ± 5.995
ATSP-100	$\mathbf{1.630 \pm 0.004}$	$\mathbf{4.103 \pm 0.105}$	1.741 ± 0.005	11.196 ± 0.293

RRL yields a lower mean objective and gap than vanilla RL on all four tasks. The improvement persists across independently trained checkpoints, including the higher-variance PCTSP setting. Because this controlled study uses three seeds and a shortened 200-epoch budget, we interpret it as direct evidence that the observed trend is not attributable to a single favorable seed, rather than as a high-powered significance analysis of every full-scale experiment.

F.6 Extended Results for Non-graph-based Combinatorial Problems

F.6.1 Extended Results and Analysis on SMTWTP and JSSP

Table 32: Full results on SMTWTP.

METHOD	TYPE	**SMTWTP-20** (100k inst.)			**SMTWTP-50** (100k inst.)			**SMTWTP-100** (100k inst.)		
		OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓	OBJ.↓	DROP↓	TIME↓
OR-Tools (10s) [55]	Heuristics	**2.5458**	0.000%	0.051s	5.8857	0.000%	0.084s	12.3909	0.000%	0.103s
POMO [19]	RL+G	2.6377	3.61%	0.042s	5.9812	1.62%	0.065s	12.6816	2.35%	0.088s
POMO [19]	IL+G	2.6283	3.24%	0.041s	5.9234	0.64%	0.067s	12.5753	1.49%	0.085s
CORectifier (ours)	RRL+G	**2.5833**	**1.47%**	0.039s	**5.7195**	**-2.82%**	0.062s	**11.8495**	**-4.37%**	0.082s

Experimental results on SMTWTP and JSSP (Tables 32 and 33) demonstrate the superior scalability and efficiency of CORectifier. On SMTWTP, CORectifier significantly outperforms both OR-Tools and learning baselines on larger instances, achieving negative optimality gaps of **-2.82%** (Size-50) and **-4.37%** (Size-100) with faster inference. Similarly for JSSP, CORectifier consistently surpasses heuristics and standard RL baselines across all sizes; it outperforms the strong MatNet baseline on small-to-medium instances and remains highly competitive on 20×20 instances, demonstrating a robust balance between solution quality and computational efficiency.

Table 33: Full results on JSSP. Baseline results† are reported from RL4CO [53].

METHOD	TYPE	**JSSP** 6 × 6 (100k inst.)			**JSSP** 10 × 10 (100k inst.)			**JSSP** 15 × 15 (100k inst.)			**JSSP** 20 × 20 (100k inst.)		
		OBJ.↓	GAP↓	TIME	OBJ.↓	GAP↓	TIME	OBJ.↓	GAP↓	TIME	OBJ.↓	GAP↓	TIME
OR-Tools(10s) [55]	Heuristics	494.83	0.00%	0.001s	814.24	0.00%	0.027s	1249.83	0.00%	0.072s	1655.06	0.00%	0.161s
MWKR†	Heuristics	656.96	32.76%	-	1151.41	41.41%	-	1812.13	44.99%	-	2469.19	49.19%	-
MOR†	Heuristics	630.19	27.35%	-	1101.08	35.23%	-	1693.33	35.48%	-	2263.68	36.77%	-
L2D† [76]	RL+G	574.09	16.02%	-	988.58	21.41%	-	1504.79	20.40%	-	2007.76	21.31%	-
RL4CO+GCN† [53]	RL+G	569.53	15.10%	-	972.35	19.42%	-	1492.94	19.45%	-	1992.36	20.38%	-
RL4CO+MatNet† [53]	RL+G	515.11	4.10%	-	865.78	6.33%	-	1318.25	5.47%	-	1847.33	11.62%	-
CORectifier (ours)	RRL + G	**506.39**	**2.34%**	0.0009s	**856.67**	**5.21%**	0.021s	**1315.91**	**5.29%**	0.058s	**1870.16**	**13.00%**	0.126s

F.6.2 Extended Results and Analysis on KP

Table 34 presents the full results on KP, supplementing Table 4 with solving time reported.

Table 34: Full results on KP. Compared results are partially quoted from POMO [19] and GOAL [24]. The objective values of the baseline methods are omitted because these existing literature do not open-source their original test datasets, while we have generated test instances with the exact distribution and settings they described, which shall be made public.

METHOD	TYPE	KP-50 (1280 inst.)			KP-100 (1280 inst.)		
		OBJ.↑	DROP↓	TIME↓	OBJ.↑	DROP↓	TIME↓
OR-Tools [55]	Exact	20.021*	0.000%	0.001s	40.302*	0.000%	0.002s
Sequential Decision Methods							
AM [18]	RL+G	–	0.173%	–	–	0.211%	–
POMO [19]	RL+G	–	0.130%	–	–	0.19%	–
BQ-NCO [43]	SL+G	–	–	–	–	0.10%	–
GOAL [24]	SL+G	–	–	–	–	0.12%	–
CORectifier (ours)	RRL+G	**20.018**	**0.013%**	0.024s	**40.298**	**0.012%**	0.043s

METHOD	TYPE	KP-200 (1280 inst.)			KP-500 (1280 inst.)		
		OBJ.↑	DROP↓	TIME↓	OBJ.↑	DROP↓	TIME↓
OR-Tools [55]	Exact	57.402*	0.000%	0.002s	91.128*	0.000%	0.004s
Sequential Decision Methods							
AM [18]	RL+G	–	0.325%	–	–	–	–
POMO [19]	RL+G	–	0.50%	–	–	6.41%	–
BQ-NCO [43]	SL+G	–	0.14%	–	–	0.74%	–
GOAL [24]	SL+G	–	1.63%	–	–	2.40%	–
CORectifier (ours)	RRL+G	**57.397**	**0.009%**	0.058s	**91.121**	**0.007%**	0.110s

F.7 Trajectory-distribution Diagnostics

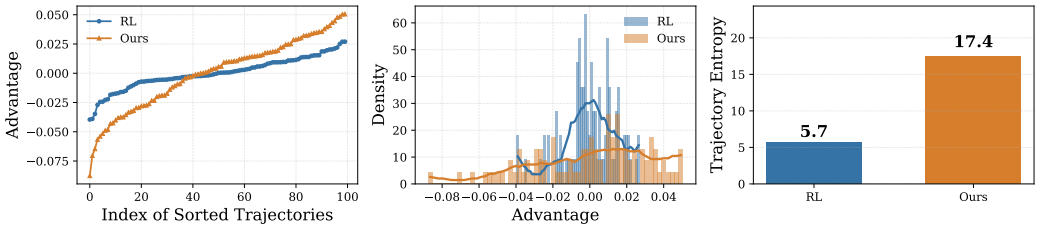


Figure 9: Exploration diversity on TSP-100. Left: Advantage separation. Mid: Advantage scale distribution. Right: Trajectory entropy.

Fig. 9, Fig. 10, and Fig. 11 provide additional visualization of the exploration behaviors on TSP-100, ATSP-100, and CVRP-100. Across all problems, three consistent phenomena can be observed.

Advantage separation. The left panels show the sorted trajectory advantages. Our method consistently yields a wider spread between high- and low-advantage trajectories compared to the RL baseline. This indicates that CORectifier promotes a more diverse exploration profile, generating

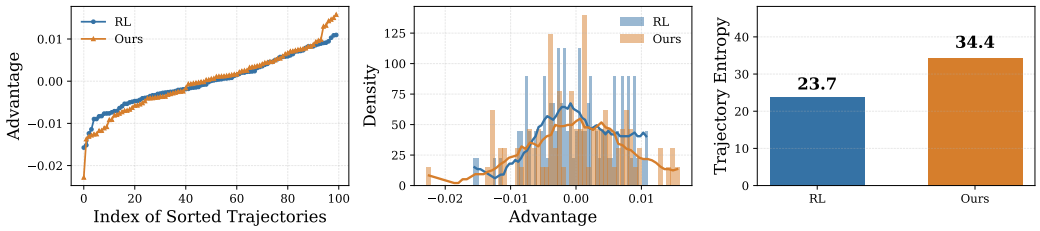


Figure 10: Exploration diversity on ATSP-100. Left: Advantage separation. Mid: Advantage scale distribution. Right: Trajectory entropy.

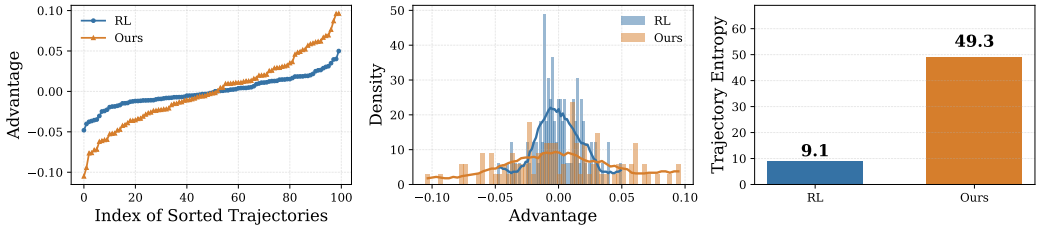


Figure 11: Exploration diversity on CVRP-100. Left: Advantage separation. Mid: Advantage scale distribution. Right: Trajectory entropy.

trajectories of both higher and lower advantage rather than collapsing toward a narrow band of behaviors.

Advantage scale distribution. The middle panels further confirm this effect. The RL baseline exhibits a sharper, more concentrated advantage distribution, whereas our method produces a noticeably broader density with heavier tails. This larger variance suggests that our agent explores more heterogeneous solution paths, covering regions that standard RL rarely visits.

Trajectory entropy. The right panels summarize trajectory-level diversity using a discrete entropy metric. Our method achieves substantially higher entropy across all tasks (e.g., 17.4 v.s. 5.7 on TSP-100, 34.4 v.s. 23.7 on ATSP-100, 49.3 v.s. 9.1 on CVRP-100), demonstrating that the generated solution trajectories differ more significantly from one another. This indicates a richer and less deterministic exploration process.

Summary. Taken together, these visualizations show that CORectifier consistently enhances exploration diversity from multiple perspectives: advantage ordering, advantage variance, and trajectory entropy. The effect is robust across symmetric, asymmetric, and capacitated routing problems. This provides further evidence that the rectified policy improves the quality of exploration rather than merely altering its magnitude.

G Further Theoretical Justifications

G.1 A Sketch of Proof for Equation 4

In this section, we provide a proof sketch for Eq. 4, i.e., for the existence of a positive *policy-dependent* margin for labeled instances in $\mathcal{G}^*$. Concretely, for a fixed policy parameter θ, we consider

$$\exists \epsilon_\theta > 0, \quad \mathbb{E}_{\tau \sim \pi_\theta(\cdot|G)} \left[c(\tau; G) - c(\tau_G^* \mid G) \right] \geq \epsilon_\theta, \ \forall G \in \mathcal{G}^*, \tag{46}$$

which serves as the conceptual motivation of the CORectifier framework. The equation asserts the existence of a positive quality margin ϵ_θ between the expected cost of trajectories generated by the policy π_θ and the cost of reference trajectories τ_G^* for all labeled instances $G \in \mathcal{G}^*$, under the regime where the references outperform the current policy. This margin explains why the reference trajectories can provide meaningful guidance during training. *Please note that this inequality, as well as the following proof, is better viewed as a conceptual illustration grounded in engineering practice rather than a uniform formal guarantee over the entire training process.*

Quality of Reference Trajectories. The reference trajectories τ_G^* are generated by established oracle solvers (e.g., Concorde for TSP, LKH-3 for ATSP, OR-Tools/ILS for PCTSP). These solvers are

designed to produce exact or near-optimal solutions for their respective COPs. Formally, for any instance G, the cost $c(\tau_G^* \mid G)$ is expected to be within a small factor of the (theoretical) optimal cost $c^*(G)$, often satisfying

$$c(\tau_G^* \mid G) \leq (1+\delta)c^*(G), \tag{47}$$

for some small $\delta \geq 0$ (depending on the solver's approximation ratio, or $\delta = 0$ for exact solvers). Therefore, since solutions generated by existing RL-based solvers usually possess strictly positive performance drops in the regimes considered in this paper, $c(\tau_G^* \mid G)$ is reasonably qualified to constitute high-quality guiding signals.

Policy-Generated Trajectories. At the initial stages of training, the policy π_θ is typically suboptimal. Since COPs involve vast combinatorial action spaces, random exploration or naive policy updates often yield trajectories τ with costs $c(\tau; G)$ that are significantly higher than $c(\tau_G^* \mid G)$. Even as training progresses, unless the policy converges to the global optimum or the references become insufficiently informative, the expectation $\mathbb{E}_{\tau\sim\pi_\theta(\cdot|G)}[c(\tau; G)]$ can remain above $c(\tau_G^* \mid G)$ due to the complexity of the problem and the limited sample efficiency of vanilla RL.

Existence of ϵ-Margin. For a fixed θ, let

$$\Delta_\theta(G) := \mathbb{E}_{\tau\sim\pi_\theta(\cdot|G)}[c(\tau; G)] - c(\tau_G^* \mid G), \tag{48}$$

which, by the above arguments, implies $\Delta_\theta(G) > 0$ for each $G \in \mathcal{G}^*$ in the regime where the reference trajectories outperform the current policy. Since $\mathcal{G}^*$ is a finite set (in practice, a collection of labeled instances), the finite collection $\{\Delta_\theta(G) : G \in \mathcal{G}^*\}$ attains a minimum over $\mathcal{G}^*$. Define $\epsilon_\theta := \min_{G\in\mathcal{G}^*} \Delta_\theta(G)$, and we have $\epsilon_\theta > 0$ because:

1) Each $\Delta_\theta(G) > 0$ (when the reference trajectories are strictly better than the policy's expected performance at the current θ);

2) The minimum over a finite set of positive values is positive.

Thus, for all $G \in \mathcal{G}^*$, there exists some positive margin ϵ_θ such that

$$\mathbb{E}_{\tau\sim\pi_\theta(\cdot|G)}[c(\tau; G) - c(\tau_G^* \mid G)] \geq \epsilon_\theta > 0 \tag{49}$$

holds, which satisfies Eq. 4 for the fixed policy π_θ.

Practical Justification. In practice, the value of ϵ_θ may vary with the policy, problem size, and instance distribution, and it may shrink as the policy improves, but it remains positive whenever the oracle solvers outperform the current policy on the labeled set. This margin justifies the hierarchical rectification mechanism in CORectifier. By replacing segments of policy-generated trajectories with reference segments, the rectified trajectories inherit useful low-cost structure from τ_G^*, thereby providing strong learning signals (via advantage estimates) to guide the policy toward better regions.

This proof sketch relies on the realistic assumption that strong heuristic solvers provide high-quality solutions, which is standard in NCO practices. Empirical results in Sec. 4.1 should be viewed as grounded validation that such a positive margin exists in the relevant training regime, as CORectifier consistently improves over RL baselines.

G.2 A More Rigorous Interpretation of the RRL Update

In this section, we formally justify the optimization actually carried out in Stage 2. Eq. 5 provides a high-level rectified-return view of CORectifier, while Eq. 11 and Eq. 12 define the tractable objective used in practice. The key point is that the rectified trajectory τ' is *not* obtained by post-hoc editing of a completed rollout. Instead, as specified by the iterative rule in Eq. 10, rectification is performed locally and autoregressively: once a rectified action is accepted at step t, the environment transitions on the corresponding updated partial trajectory, and all subsequent actions are sampled under the resulting valid state. Therefore, the final trajectory τ' remains feasible, and $\log \pi_\theta(\tau' \mid G)$ is well-defined on the actually visited states.

Fix a parameter vector $\bar{\theta}$, which is used only for trajectory generation. For an instance G_i, let

$$T_i' = \{\tau_i'^j\}_{j=1}^N \tag{50}$$

denote the N parallel trajectories produced by the exact Stage-2 procedure under $\pi_{\bar{\theta}}$: batch sampling follows Eq. 7, a proportion p_{inst} of the N trajectories is selected for rectification, the mask is sampled as in Eq. 9, and action replacement follows Eq. 10. The joint distribution of the N trajectories are

$$T_i' \sim \widetilde{\Pi}_{\bar{\theta}}(\cdot \mid G_i). \tag{51}$$

When rectification is inactive, this distribution reduces to the ordinary rollout distribution induced by $\pi_{\bar{\theta}}(\cdot \mid G_i)$. For each instance G_i, the advantage used in Stage 2 is

$$A_{i,j}(T_i') = R(\tau_i'^j; G_i) - \frac{1}{N}\sum_{j'=1}^{N} R(\tau_i'^{j'}; G_i), \tag{52}$$

which is exactly the within-instance baseline defined in Sec. 2.2. We define the Stage-2 surrogate:

$$\widetilde{\mathcal{L}}_{RRL}(\theta;\bar{\theta}) = -\mathbb{E}_{G_i,\, T_i' \sim \widetilde{\Pi}_{\bar{\theta}}(\cdot|G_i)} \left[\frac{1}{N}\sum_{j=1}^{N} A_{i,j}(T_i') \log \pi_\theta(\tau_i'^j \mid G_i) \right]. \tag{53}$$

Proposition 1. For fixed rollout parameter $\bar{\theta}$, the gradient of $\widetilde{\mathcal{L}}_{RRL}(\theta;\bar{\theta})$ is

$$\nabla_\theta \widetilde{\mathcal{L}}_{RRL}(\theta;\bar{\theta}) = -\mathbb{E}_{G_i,\, T_i' \sim \widetilde{\Pi}_{\bar{\theta}}(\cdot|G_i)} \left[\frac{1}{N}\sum_{j=1}^{N} A_{i,j}(T_i') \nabla_\theta \log \pi_\theta(\tau_i'^j \mid G_i) \right]. \tag{54}$$

Here $A_{i,j}(T_i')$ is treated as constant w.r.t. θ. The mini-batch gradient estimator is evaluated on rectified-prefix states. In practice, $\bar{\theta}$ is set to the current policy before rollout collection, and gradients are stopped through sampling and rectification.

Hence, when the expectation is approximated by a mini-batch of B instances, Eq. 12 is an unbiased Monte Carlo estimator of $\nabla_\theta \widetilde{\mathcal{L}}_{RRL}(\theta;\bar{\theta})$.

Proof. By construction, for each sampled instance G_i, the random rollout set T_i' is drawn from the induced rectified rollout distribution $\widetilde{\Pi}_{\bar{\theta}}(\cdot \mid G_i)$, which is determined by the frozen behavior policy $\pi_{\bar{\theta}}$ together with the rectification procedure. Therefore, the coefficients $A_{i,j}(T_i')$ are sampled first and are treated as fixed random coefficients during differentiation with respect to θ. By linearity of expectation,

$$\begin{aligned} \nabla_\theta \widetilde{\mathcal{L}}_{RRL}(\theta;\bar{\theta}) &= -\nabla_\theta \mathbb{E}_{G_i,\, T_i' \sim \widetilde{\Pi}_{\bar{\theta}}(\cdot|G_i)} \left[\frac{1}{N}\sum_{j=1}^{N} A_{i,j}(T_i') \log \pi_\theta(\tau_i'^j \mid G_i) \right] \\ &= -\mathbb{E}_{G_i,\, T_i' \sim \widetilde{\Pi}_{\bar{\theta}}(\cdot|G_i)} \left[\frac{1}{N}\sum_{j=1}^{N} A_{i,j}(T_i') \nabla_\theta \log \pi_\theta(\tau_i'^j \mid G_i) \right], \end{aligned} \tag{55}$$

which proves Eq. (54). Replacing the expectation by the empirical average over B sampled instances and N trajectories per instance yields

$$\nabla_\theta \widetilde{\mathcal{L}}_{RRL}(\theta;\bar{\theta}) \approx -\frac{1}{BN}\sum_{i=1}^{B}\sum_{j=1}^{N} A_{i,j}(T_i') \nabla_\theta \log \pi_\theta(\tau_i'^j \mid G_i), \tag{56}$$

which is exactly Eq. 12. This proves the claim.

Interpretation. For each realized trajectory $\tau_i'^j$, the sample-level contribution to the loss is

$$-A_{i,j}(T_i') \log \pi_\theta(\tau_i'^j \mid G_i). \tag{57}$$

Therefore, minimizing $\widetilde{\mathcal{L}}_{RRL}$ increases the likelihood of rectified trajectories with positive advantage and decreases the likelihood of those with negative advantage. In this sense, CORectifier does not blindly imitate expert-guided rollouts; it performs advantage-weighted teacher forcing on the realized rectified trajectories, so that only those rectified samples that outperform the within-instance baseline are reinforced.

Proposition 2. The surrogate in Eq. 53 recovers or connects to standard learning paradigms in two limiting cases.

i) Vanilla RL case. If rectification is disabled, namely when labeled batches are not selected, or $p_{inst} = 0$, or the sampled rectified segment length is zero, then

$$\tau_i'^j = \tau_i^j, \qquad \forall i, j, \tag{58}$$

and the rollout distribution becomes the ordinary policy-induced distribution. Accordingly,

$$\widetilde{\mathcal{L}}_{RRL}(\theta; \bar{\theta}) = -\mathbb{E}_{G_i,\, T_i \sim \Pi_{\bar{\theta}}(\cdot|G_i)} \left[\frac{1}{N} \sum_{j=1}^{N} A_{i,j}(T_i) \log \pi_\theta(\tau_i^j \mid G_i) \right], \tag{59}$$

which is precisely the standard advantage-weighted REINFORCE surrogate in Sec. 2.1.

ii) Fully guided case. At the opposite extreme, if every selected labeled rollout is rectified at every applicable step and the recommended action always passes the feasibility check, then each selected trajectory is completely generated by the expert-guided rule in Eq. 10. In this regime, the guided samples enter the same surrogate through

$$\widetilde{\mathcal{L}}_{RRL}(\theta; \bar{\theta}) = -\mathbb{E}_{G_i,\, T_i' \sim \widetilde{\Pi}_{\bar{\theta}}(\cdot|G_i)} \left[\frac{1}{N} \sum_{j=1}^{N} A_{i,j}(T_i') \log \pi_\theta(\tau_i'^j \mid G_i) \right]. \tag{60}$$

which is an advantage-weighted teacher-forcing form on the guided trajectories.

Under the within-instance mean baseline used in this paper, however, there is an important degenerate sub-case. If, for some instance G_i, all N trajectories are fully guided and collapse to the same guided trajectory, or to reward-equivalent symmetric variants, then

$$R(\tau_i'^1; G_i) = \cdots = R(\tau_i'^N; G_i), \tag{61}$$

so that the baseline equals this common reward and hence

$$A_{i,j}(T_i') = 0, \qquad \forall j. \tag{62}$$

In that case, the gradient contribution of this instance vanishes rather than yielding a functional teacher-forcing update. Therefore, the teacher-forcing interpretation is non-degenerate only when the within-instance trajectories retain reward diversity, e.g., by mixing guided and unguided rollouts or, more generally, by avoiding complete collapse of all trajectories to identical fully guided outcomes. This also clarifies the role of keeping $p_{inst} < 1$ in practice: it preserves reward variance within an instance and thus maintains informative advantage signals.

Together, Proposition 1 and Proposition 2 show that CORectifier does not introduce an unrelated optimization rule on top of RL. Instead, it preserves the same advantage-weighted log-likelihood form while changing the rollout distribution through the hierarchical rectification mechanism. The three controls p_{batch}, p_{inst}, and $[\alpha, \beta]$ determine how far the training distribution departs from vanilla RL and how strongly expert guidance participates in the update. Therefore, Eq. 5 should be understood as the high-level rectified-return perspective of the method, whereas Eq. 11 is the exact stochastic optimization rule for the Stage-2 surrogate induced by the autoregressive rectified rollout process.

H Declaration: Positioning within the NCO and RL4CO Landscape

To better contextualize our contribution, Table 35 summarizes representative recent NCO methods, with emphasis on sequential-decision approaches that are most relevant to RL4CO.

Takeaways. 1) In current NCO practice, the learning paradigm (SL/RL/UL) is still closely associated with the decision paradigm: heatmap-guided global prediction methods are predominantly developed under SL/UL settings, whereas local constructive methods are largely driven by RL and its recent variants. **2)** This distinction also correlates with task preference and empirical scalability. Global-prediction methods are typically easier to scale on routing benchmarks such as TSP, while sequential RL4CO methods remain more naturally suited to problems with richer constraints because feasibility can be enforced directly during autoregressive construction. **3)** By recent RL4CO standards, training scales around 100 nodes remain common among autoregressive constructive methods, although several methods, including BQ-NCO and GOAL, report test-time generalization to substantially larger instances. In this context, our study follows established community practice in its core settings, while also broadening the evaluation spectrum across both more constrained and larger-scale COPs.

Table 35: Representative recent NCO methods, grouped by decision paradigm. *Max Scale* denotes the largest reported routing scale for *training* in the corresponding paper. In particular, BQ-NCO and GOAL report test-time evaluation up to 1K nodes, while CORectifier evaluates zero-shot CVRP generalization up to 5K nodes.

Method	Main Evaluated Tasks	Paradigm	Max Scale	Year
Heatmap-guided Global Prediction (GP)				
GNN4CO [10]	TSP	SL	100	2019
GNNGLS [11]	TSP	SL	100	2022
DIFUSCO [13]	TSP, MIS	SL	1000	2023
T2TCO [14]	TSP, MIS	SL	1000	2023
UTSP [40]	TSP	UL	100	2023
Fast-T2T [15]	TSP, MIS	SL	1000	2024
UnifyML4TSP [23]	TSP	SL	500	2025
Sequential-decision Local Construction (LC)				
AM [18]	TSP, CVRP, SDVRP, OP, (S)PCTSP	RL	100	2019
POMO [19]	TSP, CVRP, KP	RL	100	2020
MatNet [20]	ATSP, FFSP	RL	100	2021
Sym-NCO [21]	TSP, CVRP, PCTSP, OP	RL	100	2022
BQ-NCO [43]	TSP, ATSP, CVRP, OP, KP	SL	100	2023
MAB-MTL [57]	TSP, CVRP, OP, KP	RL	100	2023
MVMoE [77]	VRP variants	RL	100	2024
MTNCO [78]	VRP variants	RL	100	2024
UNCO [79]	TSP, CVRP, KP, SMTWTP	RL	100	2024
UniCO [45]	TSP, ATSP	RL	100	2025
GOAL [24]	ATSP, CVRP, CVRPTW, OP, JSSP	SL	100	2025
UniteFormer [37]	TSP, ATSP, CVRP	RL	100	2025
BOPO [25]	TSP, JSSP, FJSP	improved RL	100	2026
PO4CO [26]	TSP, CVRP	improved RL	100	2026
UCPO [80]	TSPTW, TSPDL, CVRPTW, CVRPTWLV	improved RL	100	2026
CORectifier	TSP, ATSP, CVRP, PCTSP, KP, JSSP, SMTWTP	improved RL	500	2026

I Limitations

While CORectifier significantly mitigates core challenges in RL-based CO solving (e.g., reward sparsity, sample inefficiency, and scalability issues), not a single work could claim to fully resolve fundamental limitations inherent to broad RL frameworks, which may still manifest in specific scenarios. In some cases (e.g., novel CO variants with no established optimal solvers, or large-scale instances where expert trajectory generation is computationally intractable), acquiring references may be cost-prohibitive. That said, CORectifier demonstrates robust performance improvements over existing methods within the same RL4CO category when expert data is available. Critically, as stated in Sec. 4.2 (stability study), this paradigm exhibits a "degradation safety guarantee": as expert data quality declines or supervision is entirely absent, CORectifier naturally (in the worst case) degenerates to vanilla RL, eliminating the risk of performance falling below the baseline established by standalone RL solvers. We hold it firmly that advancing the ML4CO field requires synergistic efforts across multiple directions (e.g., expert data distillation, unsupervised RL for CO, and backbone architecture innovation), while this work represents one step toward addressing RL4CO's core limitations.

J Broader Impact

CORectifier adapts established expert-guided learning principles to autoregressive neural combinatorial optimization. Its potential benefit lies in improving the use of limited offline reference trajectories while preserving the feasibility interface of the underlying constructive environment. Its applicability depends on the availability and computational cost of reference solutions, and its trained policies remain specific to their tasks and training distributions.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction accurately reflect the paper's contributions and scope, which are further validated in the experiment section.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes. We discuss the limitations in Appendix **I**.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Necessary proofs are provided in Appendix **G.1** and **G.2**.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We describe in detail our proposed model architecture and algorithms, and provide sufficient hyper-parameter settings, hardware configurations, datasets used, and other model settings for reproducibility. Full codebase and datasets have also been made publicly available.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Full code base and datasets have been made publicly available upon acceptance (https://github.com/Thinklab-SJTU/CORectifier). We describe in detail our proposed model architecture and algorithms, and provide sufficient hyper-parameter settings, hardware configurations, datasets used, and other model settings for reproducibility. Also, we have provided the PyTorch-style pseudo-code for the core rectification parts in Appendix **C.2**.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We describe in detail our proposed model architecture and algorithms, and provide sufficient hyper-parameter settings, hardware configurations, datasets used, and other model settings for reproducibility. Please see Appendix **E**.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We have reported standard deviations of the optimality drops, following convention in the NCO literature [23, 34, 16], etc.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Please see Appendix **E** and **F.1**.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conform strictly with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Please see Appendix **J**.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All our datasets are from the open-source benchmark named ML4CO-Bench-101 [34], with a license of CC-BY 4.0.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Source code is available at `https://github.com/Thinklab-SJTU/CORectifier`, with documentation of the proposed models.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components. LLMs are only used for polishing the writing.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.