RL4CO: An Extensive Reinforcement Learning for Combinatorial Optimization Benchmark

Federico Berto*
KAIST, Omelet
Daejeon, Republic of Korea
fberto@kaist.ac.kr

Chuanbo Hua*
KAIST, Omelet
Daejeon, Republic of Korea
chua@kaist.ac.kr

Junyoung Park*
KAIST
Daejeon, Republic of Korea
junyoungpark@kaist.ac.kr

Laurin Luttmann*
Lüneburg Leuphana University
Germany
lluttmann@leuphana.de

Yining Ma
MIT
Cambridge, MA, United States
yiningma@mit.edu

Fanchen Bu
KAIST
Daejeon, Republic of Korea
boqvezen97@kaist.ac.kr

Jiarui Wang
Southeast University
Nanjing, China
jiarui_wang@seu.edu.cn

Haoran Ye
Peking University
Beijing, China
haoran.ye@pku.edu.cn

Minsu Kim
Mila, KAIST
Montreal, Canada
minsu.kim@mila.quebec

Sanghyeok Choi
KAIST
Daejeon, Republic of Korea
sanghyeok.choi@kaist.ac.kr

Nayeli Gast Zepeda
Bielefeld University
Bielefeld, Germany
nayeli.gast@uni-bielefeld.de

André Hottung
Bielefeld University
Bielefeld, Germany
andre.hottung@uni-bielefeld.de

Jianan Zhou
Nanyang Technological University
Singapore, Singapore
jianan004@e.ntu.edu.sg

Jieyi Bi
Nanyang Technological University
Singapore, Singapore
jieyi001@e.ntu.edu.sg

Yu Hu
Soochow University
Suzhou, China
yhutycbony@stu.suda.edu.cn

Fei Liu
City University of Hong Kong
Hong Kong, China
fliu36-c@my.cityu.edu.hk

Hyeonah Kim
Mila, Université de Montréal
Montreal, Canada
hyeonah.kim@mila.quebec

Jiwoo Son
Omelet
Daejeon, Republic of Korea
jiwoo.son@omelet.ai

Haeyeon Kim
KAIST
Daejeon, Republic of Korea
haeyeonkim@kaist.ac.kr

Davide Angioni
University of Brescia
Brescia, Italy
d.angioni@unibs.it

Wouter Kool
ORTEC
Rotterdam, Netherlands
wouter.kool@ortec.com

Zhiguang Cao
Singapore Management University
Singapore, Singapore
zgcao@smu.edu.sg

Qingfu Zhang
City University of Hong Kong
Hong Kong, China
qingfu.zhang@cityu.edu.hk

Joungho Kim
KAIST
Daejeon, Republic of Korea
joungho@kaist.ac.kr

Jie Zhang
Nanyang Technological University
Singapore, Singapore
zhangj@ntu.edu.sg

Kijung Shin
KAIST
Daejeon, Republic of Korea
kijungs@kaist.ac.kr

Cathy Wu
MIT
Cambridge, MA, United States
cathywu@mit.edu

Sungsoo Ahn
KAIST
Daejeon, Republic of Korea
sungsoo.ahn@kaist.ac.kr

Guojie Song
Peking University
Beijing, China
gjsong@pku.edu.cn

Changhyun Kwon
KAIST, Omelet
Daejeon, Republic of Korea
chkwon@kaist.ac.kr

Kevin Tierney
Bielefeld University
Bielefeld, Germany
kevin.tierney@uni-bielefeld.de

Lin Xie
Brandenburg University of Technology
Cottbus and Senftenberg, Germany
lin.xie@b-tu.de

Jinkyoo Park
KAIST, Omelet
Daejeon, Republic of Korea
jinkyoo.park@kaist.ac.kr

Abstract

Combinatorial optimization (CO) is fundamental to several real-world applications, from logistics and scheduling to hardware design and resource allocation. Deep reinforcement learning (RL) has recently shown significant benefits in solving CO problems, reducing reliance on domain expertise and improving computational efficiency. However, the absence of a unified benchmarking framework leads to inconsistent evaluations, limits reproducibility, and increases engineering overhead, raising barriers to adoption for new researchers. To address these challenges, we introduce RL4CO, a unified and extensive benchmark with in-depth library coverage of 27 CO problem environments and 23 state-of-the-art baselines. Built on efficient software libraries and best practices in implementation, RL4CO features modularized implementation and flexible configurations of diverse environments, policy architectures, RL algorithms, and utilities with extensive documentation. RL4CO helps researchers build on existing successes while exploring and developing their own designs, facilitating the entire research process by decoupling science from heavy engineering. We finally provide extensive benchmark studies to inspire new insights and future work. RL4CO has already attracted numerous researchers in the community and is open-sourced at https://github.com/ai4co/rl4co[1].

CCS Concepts

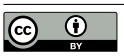
• **Computing methodologies** → **Reinforcement learning**; • **Mathematics of computing** → **Combinatorial optimization**; • **Applied computing** → *Supply chain management*.

Keywords

Reinforcement Learning, Combinatorial Optimization, Neural Combinatorial Optimization, Benchmark, Open Research Community

ACM Reference Format:
Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Qingfu Zhang, Joungho Kim, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Kevin Tierney, Lin Xie, Jinkyoo Park. 2025. RL4CO: An Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3737433

[*] Equal contribution.
[1] Homepage: https://rl4co.ai4co.org



KDD '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1454-2/2025/08
https://doi.org/10.1145/3711896.3737433

1 Introduction

Combinatorial optimization (CO) focuses on finding solutions for problems with discrete variables and has broad applications, including vehicle routing [42, 65], scheduling [91], and hardware device placement [37]. Given that the combinatorial space expands exponentially and exhibits NP-hard characteristics, the operations research (OR) community has traditionally tackled these challenges through the development of mathematical programming algorithms [27] and handcrafted heuristics [62]. Despite their success, these methods still face significant limitations: mathematical programming struggles with scaling, while handcrafted heuristics require significant domain-specific adjustments for different CO problems.

Recently, to address these limitations, neural combinatorial optimization (NCO) has emerged [4, 56]. NCO employs deep neural networks to automate the problem-solving process, significantly reducing computation demands and domain expertise requirements. One option to train NCO solvers is to use supervised learning [23, 58, 83] by training on large datasets of labeled optimal solutions, akin to large-language models [81]. However, this paradigm becomes impractical in NCO for large numbers of constraints, larger scales, and (new) problems where optimal solutions are unavailable. Conversely, reinforcement learning (RL) alleviates these issues by not necessitating labeled data and efficiently enabling automated learning of possibly better solutions than human experts in complex problems [21, 75]. RL has demonstrated its effectiveness in several research works and has made significant strides in several areas, including improving exploration efficiency [39, 44], relaxing the needs of obtaining optimal solutions [32], and extending to various CO tasks [8, 37, 42, 65, 91].

Despite the growing popularity and advancements in using RL for solving CO problems, there remains a lack of a unified benchmark for analyzing different approaches under consistent implementations and conditions. The absence of such standardized benchmarks hinders NCO researchers' efforts in leveraging existing successes to make impactful advancements, and it becomes challenging to determine the superiority of one method over another.

Table 1.1: Comparison of software libraries in reinforcement learning for combinatorial optimization.

Library	Environments #	Baselines† #	Hardware Acceleration	Availability	Modular Baselines	Open Community
ORL [1]	3	1	×	×	×	×
OR-Gym [33]	9	-	×	✓	×	×
Graph-Env [9]	2	-	×	✓	×	×
RLOR [84]	2	2	×	✓	✓	×
RoutingArena [80]	1	8	✓	×	×	×
Jumanji [12]	22	3	✓	✓	×	×
RL4CO (ours)	27	23	✓	✓	✓	✓

† We consider as *baselines* ad-hoc policies (i.e., network architectures) and RL algorithms from the literature.

Variations in implementation can make it difficult for new researchers to engage with the NCO community, and inconsistent comparisons obstruct reproducibility, as the significance of NCO lies in its potential for generalizability across multiple problems without extensive problem-specific knowledge. Finally, the lack of a unified and well-documented library with modular components hinders extensibility beyond relatively simple problems to more realistic and complex settings. These issues pose significant challenges and underscore the need for a comprehensive benchmark to streamline research and foster consistent progress.

Contributions. To bridge this gap, we introduce RL4CO, the first comprehensive benchmark with multiple baselines, environments, and boilerplate from the literature, all implemented in a *modular*, *flexible*, *accelerated*, and *unified* manner. We aim to facilitate the entire research process for the NCO community with the following key contributions: 1) *Simplifying development* through modularizing 27 environments and 23 existing baseline models, allowing for flexible and automated combinations for effortless testing, changing components via modularization, and achieving state-of-the-art performance; 2) *Enhancing experiment efficiency* through our modular unified pipeline tailored for the NCO community based on advanced libraries from data generation to configuration handling; 3) *Standardizing evaluation* to ensure fair and comprehensive comparisons, enabling researchers to automatically test a broader range of problems from diverse distributions and gather valuable insights using our testbed; 4) *Fostering community growth* through comprehensive documentation, tutorials, and an active community that can effectively utilize, extend, and contribute to NCO research. Overall, RL4CO provides a unified platform that reduces implementation overhead, enhances efficiency, standardizes evaluation, and catalyzes innovative research directions with a focus on community building.

2 Related Works

Neural Combinatorial Optimization. Neural combinatorial optimization (NCO) utilizes machine learning techniques to automatically develop novel heuristics for solving NP-hard CO problems. We classify the majority of NCO research from the following perspectives: 1) *Learning Paradigms*: researchers have employed supervised learning (SL) [23, 30, 51, 58, 78, 83] to approximate optimal solutions to CO instances as well as alternative approaches as reinforcement learning (RL) [3, 42, 44, 65] to eliminate the need for collecting (near-)optimal solutions. 2) *Models*: various deep learning architectures such as recurrent neural networks [18, 48, 83], graph neural networks [34, 63], Transformers [42, 44], diffusion models [78], and GFlowNets [38, 92] have been employed. 3) *Problems*: NCO has demonstrated great success in various problems, including vehicle routing problems (VRPs) (e.g., traveling salesman problem (TSP) and capacitated VRP) [42], scheduling problems (e.g., job shop scheduling problems [91]), hardware device placement [37], and graph-based CO [92]. 4) *Heuristic Types*: in general, the learned heuristics can be categorized as *constructive* – constructing solutions step-by-step – in an autoregressive [42] or non-autoregressive [34] way, and *improvement* heuristics – iteratively improving existing solutions – that incorporate traditional heuristics [59, 87] and meta-heuristics [76] in a learning-based framework. We refer to Bengio et al. [4] for a comprehensive survey.

This paper focuses on the reinforcement learning paradigm due to its effectiveness and flexibility. Notably, the proposed RL4CO is versatile to support most combinations of models, problems and heuristic types, making it an apt library and benchmark for future research in RL-based NCO.

Related Benchmark Libraries. Despite the variety of general-purpose RL software libraries [16, 53, 70, 85] there is a lack of a unified and extensive benchmark for CO problems. Balaji et al. [1] propose an RL benchmark for OR that comes only with a PPO baseline [74]. Also, Hubbs et al. [33] and Biagioni et al. [9] implement a collection of OR environments, but do not provide any baselines to solve them. Wan et al. [84] propose an RL for OR library that benchmarks the canonical TSP and CVRP environments using different configurations of the attention model [42]. Despite having a different focus, we also mention the recent work of Li et al. [52], who study the effect of different combinations of learning and search methods on the TSP, and Prouvost et al. [69], who develop an API to use RL for controlling traditional MILP solvers [55] instead of directly constructing or improving solutions. Besides their relatively narrow scopes, a major downside of most of the above libraries is that they cannot be massively parallelized due to their reliance on the OpenAI Gym API [16], which can only run on CPU. In contrast, RL4CO is based on TorchRL [13], a recent official PyTorch [67] library for RL that enables hardware-accelerated execution of both environments and algorithms. In contrast, Routing Arena [80] provides a library of multiple parallelized neural as well as classical baselines, but benchmarking focused on CVRP. Most related to

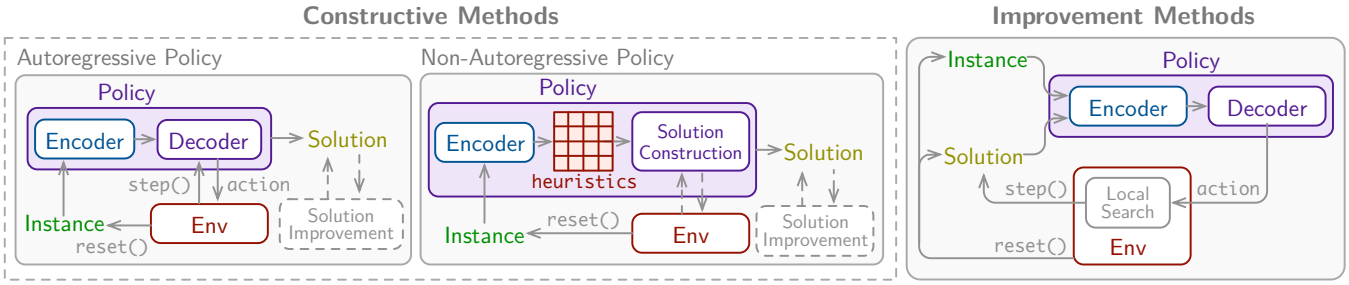


Figure 2.1: Overview of different types of policies and their modularization in RL4CO.

our work is the recent Jumanji [12], a library that provides a collection of CO problem environments written in JAX [14] that can be hardware-accelerated with an actor-critic constructive baseline. However, while Jumanji is an RL environment suite, RL4CO is a full-stack library that integrates environments, policies (i.e., neural network architectures), and RL algorithms under a unified PyTorch-based framework; thus, while policies in Jumanji are tailored to specific environments while baselines in RL4CO are modular and applicable to all suitable CO problems.

Table 1.1 compares software libraries in reinforcement learning for CO under several criteria, showing our proposed RL4CO benchmark provides the most comprehensive suite.

3 RL4CO: Taxonomy

RL has emerged as a powerful paradigm for solving combinatorial optimization (CO) problems. A key requirement for effectively applying RL to CO is structured *environments* where agents can interact with problem instances and learn through fast interactions. Therefore, a core part of our taxonomy is a diverse set of standardized CO environments that enable learning heuristics for different problems. Moreover, RL4CO integrates a comprehensive set of baseline *policies*. In our work, policies refer to specific models or neural network architectures that generate solutions for CO problems, defining how agents interact with the environment to construct or improve solutions. The final pillar of RL4CO is a set of widely adopted *RL algorithms* that optimize these policies to maximize expected solution quality.

The following chapter introduces these core components of RL4CO – Environments (Section 3.1), Policies (Section 3.2), and RL Algorithms (Section 3.3) – and their role in enabling effective RL-driven combinatorial optimization. Then, we translate the taxonomy to its unified implementation in Section 4.

3.1 Environments

Given a CO problem instance $\boldsymbol{x}$, we formulate the solution-generating procedure as a Markov Decision Process (MDP) characterized by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ as follows. *State* $\mathcal{S}$ is the space of states that represent the given problem $\boldsymbol{x}$ and the current partial solution being updated in the MDP. *Action* $\mathcal{A}$ is the action space, which includes all feasible actions a_t that can be taken at each step t. *State Transition* $\mathcal{T}$ is the deterministic state transition function $s_{t+1} = \mathcal{T}(s_t, a_t)$ that updates a state s_t to the next state s_{t+1}. *Reward* $\mathcal{R}$ is the reward function $\mathcal{R}(s_t, a_t)$ representing the immediate reward received after taking action a_t in state s_t. Finally, $\gamma \in [0, 1]$ is a discount factor that determines the importance of future rewards. Since the state transition is deterministic, we represent the solution for a problem $\boldsymbol{x}$ as a sequence of T actions $\boldsymbol{a} = (a_0, \dots, a_{T-1})$. Then, the cumulative reward $\sum_{t=0}^{T-1} \mathcal{R}(s_t, a_t)$ translates to the negative cost function of the CO problem.

3.2 Policies

Policies can be categorized into constructive policies, which generate a solution from scratch, and improvement policies, which refine an existing solution, as depicted in Fig. 2.1.

Constructive policies. A policy π is used to construct a solution from scratch for a given problem instance $\boldsymbol{x}$. We can further categorize policies into autoregressive (AR) and non-autoregressive (NAR) policies. An AR policy is composed of an encoder f that maps the instance $\boldsymbol{x}$ into an embedding space $\boldsymbol{h} = f(\boldsymbol{x})$ and a decoder g that iteratively determines a sequence of actions $\boldsymbol{a}$ based on the encoding $\boldsymbol{h}$ as well as the sequence of previous actions:

$$\begin{aligned} a_t &\sim g(a_t | a_{t-1}, ..., a_0, s_t, \boldsymbol{h}), \\ \pi(\boldsymbol{a}|\boldsymbol{x}) &\triangleq \prod_{t=1}^{T-1} g(a_t | a_{t-1}, \dots, a_0, s_t, \boldsymbol{h}). \end{aligned} \tag{1}$$

A NAR policy encodes a problem $\boldsymbol{x}$ into a heuristic $\mathcal{H} = f(\boldsymbol{x}) \in \mathbb{R}_+^N$, where N is the number of possible assignments across all decision variables. Each number in $\mathcal{H}$ represents an (unnormalized) probability of a particular assignment. To obtain a solution $\boldsymbol{a}$ from $\mathcal{H}$, one can sample a sequence of assignments from $\mathcal{H}$ while dynamically masking infeasible assignments to meet problem-specific constraints. It can also guide a search process, e.g., Ant Colony Optimization [22, 38, 89], or be incorporated into hybrid frameworks [90]. Here, the heuristic helps identify promising transitions and improve the efficiency of finding an optimal or near-optimal solution.

Improvement policies. A policy can be used for improving an initial solution $\boldsymbol{a}^0 = (a_0^0, \dots, a_{T-1}^0)$ into another one potentially with higher quality, which can be formulated as follows:

$$\begin{aligned} \boldsymbol{a}^k &\sim g(\boldsymbol{a}^0, \boldsymbol{h}), \\ \pi(\boldsymbol{a}^K | \boldsymbol{a}^0, \boldsymbol{x}) &\triangleq \prod_{k=1}^{K} g(\boldsymbol{a}^k | \boldsymbol{a}^{k-1}, ..., \boldsymbol{a}^0, \boldsymbol{h}), \end{aligned} \tag{2}$$

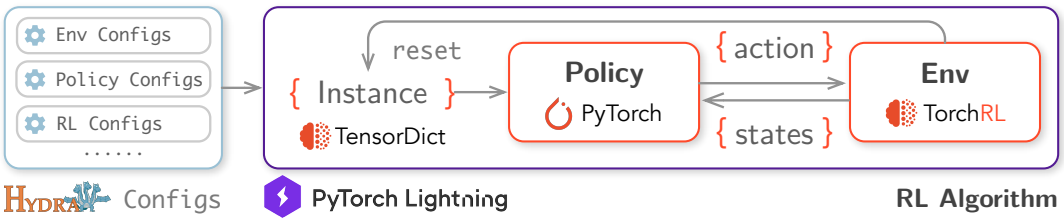


Figure 3.1: Overview of the RL4CO pipeline: from configurations to training a policy on an environment.

where $\boldsymbol{a}^k$ is the k-th updated solution and K number of improvement iterations. This process allows continuous refinement over time to improve the quality of the solution.

3.3 RL Algorithms

The RL objective is to learn a policy π that maximizes the expected cumulative reward (or equivalently minimizes the cost) over the distribution of problem instances:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})} \left[\mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{x})} \left[\sum_{t=0}^{T-1} \gamma^t \mathcal{R}(s_t, a_t) \right] \right], \quad (3)$$

where θ is the set of parameters of π and $P(\boldsymbol{x})$ is the distribution of problem instances. Eq. (3) can be solved using algorithms such as variations of REINFORCE [79], Advantage Actor-Critic (A2C) methods [41], or Proximal Policy Optimization (PPO) [74]. These algorithms are employed to train the policy network π, by transforming the maximization problem in Eq. (3) into a minimization problem involving a loss function, which is then optimized using gradient descent algorithms. For instance, the REINFORCE loss function gradient is given by:

$$\nabla_\theta \mathcal{L}_a(\theta|\boldsymbol{x}) = \mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{x})} \left[(R(\boldsymbol{a}, \boldsymbol{x}) - b(\boldsymbol{x})) \nabla_\theta \log \pi(\boldsymbol{a}|\boldsymbol{x}) \right], \quad (4)$$

where $b(\cdot)$ is a baseline function used to stabilize training and reduce gradient variance.

We further distinguish between two types of RL (pre-)training algorithms: 1) *inductive* and 2) *transductive* RL. In inductive RL, the focus is learning patterns from a training set/distribution to generalize directly to new instances. Conversely, transductive RL (i.e., test-time adaptation) optimizes parameters during testing on target instances. Typically, a policy π is trained using inductive RL, followed by transductive RL for test-time adaptation.

4 RL4CO: Library Structure

RL4CO is a unified reinforcement learning (RL) for combinatorial optimization (CO) library that aims to provide a *modular*, *flexible*, and *unified* code base for training and evaluating RL for CO methods with extensive benchmarking capabilities on various settings. As shown in Fig. 3.1, RL4CO decouples the major components of an RL pipeline, prioritizing their reusability in the implementation. Following also the taxonomy of Section 3, the main components are: Environments (Section 4.1), Policies (Section 4.2), RL algorithms (Section 4.3). We finally lay out our Baselines "Zoo" collection (Section 4.4) and Additional Features (Section 4.5).

4.1 Environments

Instance Generators. Problem instance data $\boldsymbol{x}$ is created through modular `generator` functions. Different generators can be employed to generate diverse data distributions, facilitating the generalization and testing of policies. We provide an interface to PyTorch's distributions, custom distributions such as clusters and Gaussian mixtures, as well as enabling researchers to use their own custom distributions. We also provide data-centric utilities for manipulation and loading/saving data from and to `TensorDict` [64], which acts as our advanced data carrier supporting tensor-like operations directly on GPU. RL4CO additionally allows users to load multiple datasets for training, validation and testing, which is useful for tracking generalization performance in multi-distribution and multi-task learning.

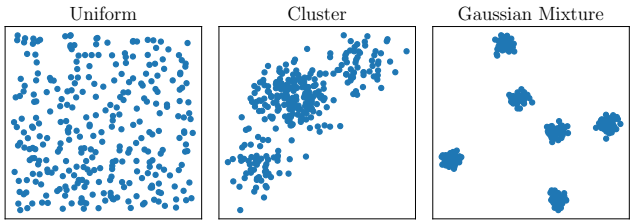


Figure 4.2: Generated instances with different distributions.

Environment Interface. Environments in RL4CO fully specify the CO problems and their logic in a stateless manner via `TensorDict`: all static and dynamic information about the environment, like the current state s_t, actions a_t, and rewards r_t are passed to and retrieved from the environment's `reset` and `step` functions. This enables modular interactions between environments and policies and allows for seamless integration with various components without the need for environment-specific adaptations. Further, a key advantage compared to other libraries is that environments in RL4CO can take batches of instances and process them in parallel on a GPU. RL4CO's environments are based on the `RL4COEnvBase` class that extends the `EnvBase` of TorchRL [13] with additional utilities and efficiency improvements. For instance, RL4CO's `step` method brings a decrease of up to 50% in latency and halves the memory impact by keeping only required transitions in the stateless `TensorDict`.

Finally, our environment API contains several methods, including vectorized `get_reward` for obtaining sparse reward from a

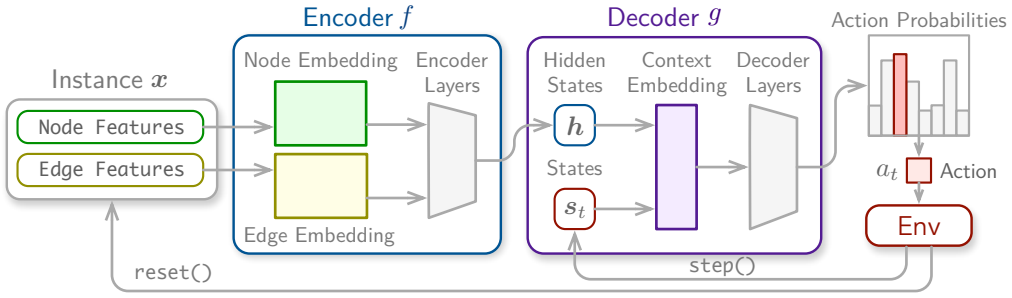


Figure 4.1: Overview of modularized RL4CO policies. Any component such as the encoder/decoder structure and feature embeddings can be replaced and thus the model is adaptable to various new environments.

given solution without stepping through the environment, methods for verifying solution validity, optional local search for integrating with heuristic solvers, and a `render` method for visualization. Fig. 4.2 shows renders of different data distributions for the Euclidean TSP generated efficiently on the fly with our instance generators.

Environments Collection. We include benchmarking for 27 environments divided into four problem areas. 1) *Routing*: Traveling Salesman Problem (TSP) [47], Capacitated Vehicle Routing Problem (CVRP) [11], Orienteering Problem (OP) [17, 46], Prize Collecting TSP (PCTSP) [2], Pickup and Delivery Problem (PDP) [35, 73] and Multi-Task VRP (MTVRP), which includes 16 problem variants, namely the basic VRPTW, OVRP, VRPB, VRPL and VRPs with the respective constraint combinations [6, 57, 93]; 2) *Scheduling*: Flexible Job Shop Scheduling Problem (FJSSP) [15], Job Shop Scheduling Problem (JSSP) [71] and Flexible Flow Shop Problem (FFSP); 3) *Electronic Design Automation*: multiple Decap Placement Problem (mDPP) [37]; 4) *Graph*: Facility Location Problem (FLP) [24] and Max Cover Problem (MCP) [36]. A detailed description of the environment implementations for these problems can be found in Appendix B.

4.2 Policies

Policies in RL4CO are based on PyTorch `nn.Module` classes and contain the encoding-decoding logic with parameters θ. Drawing on our taxonomy in Section 3, RL4CO provides policy metaclasses such as `AutoregressivePolicy`, `NonAutoregressivePolicy` and `ImprovementPolicy` that the different policies in the RL4CO "zoo" collection can inherit from. An important issue is that, unlike in natural language processing in which words are processed through pre-defined dictionaries and relatively simple embeddings, NCO needs processing possibly continuous features of discrete objects into the latent space $\boldsymbol{h}$ via specialized embeddings of different sizes depending on their function. We modularize such components to process environment-specific features into the embedding space via parametrized *embedding* functions. First, *node embeddings* $\phi_n : \mathbb{R}^{N \times m_n} \to \mathbb{R}^{N \times h}$ transform m_n raw features for the N nodes of problem $\boldsymbol{x}$ from the feature space to the embedding space $\boldsymbol{h}$. Further, *edge embeddings* $\phi_e : \mathbb{R}^{E \times m_e} \to \mathbb{R}^{E \times h}$ process m_e edge features of instance $\boldsymbol{x}$, where E is the number of edges. Lastly, *context embeddings* $\phi_c : \mathbb{R}^{m_c} \to \mathbb{R}^h$ capture contextual information not related to a specific node or edge by processing m_c context features in the current decoding step s_t.

Fig. 4.1 illustrates a general constructive autoregressive policy in RL4CO, where the feature embeddings are applied similarly to other policies. Feature embeddings can be automatically selected by RL4CO at runtime by simply passing the environment to the policy. Additionally, we allow for granular control of any higher-level policy components, such as encoders and decoders, allowing quick experimentation.

4.3 RL Algorithms

RL algorithms in RL4CO are used to learn the parameters θ of the policy by interacting with the environment and its problem instances. The parent class of algorithms is the `RL4COLitModule`, inheriting from PyTorch Lightning's `pl.LightningModule` [25]. This allows for granular support of various methods, including train, validation, and test steps, automatic logging with several services such as Wandb via automatic optimizer configuration, and several useful callbacks for RL methods such as `on_train_epoch_end` to perform additional actions such as updating a rollout baseline [42]. RL algorithms are additionally attached to an `RL4COTrainer`, a wrapper we made with additional optimizations around `pl.Trainer`. This module seamlessly supports features of modern training pipelines, including logging, checkpoint management, mixed-precision training, various hardware acceleration supports (e.g., CPU, GPU, TPU, and Apple Silicon), and multi-device distribution [50]. For instance, using mixed-precision training significantly decreases training time without sacrificing much convergence, enabling us to leverage recent routines, such as FlashAttention for transformer layers [19, 20] (see Appendix B).

4.4 Baselines Zoo

We categorize as *baselines* ad-hoc policies (i.e., network architectures) and RL algorithms from the NCO literature. RL4CO entails a collection of 23 baselines for CO from the literature, which are implemented in a modular and flexible way using metaclasses introduced in Section 4.2 and Section 4.3. We organize these into five categories: autoregressive (AR) and non-autoregressive (NAR) constructive methods, improvement methods, and general-purpose RL algorithms and transductive RL methods introduced in. Table 4.1

provides the list of baselines currently available in RL4CO. We refer the reader to Appendix B for further details.

Table 4.1: RL4CO baselines zoo.

Category	Methods
Constr. (AR)	AM [42], PtrNet [83], POMO [44], HAM [49], SymNCO [40], PolyNet [32], MTPOMO [57], MVMoE [93], L2D [91] HGNN [77], MatNet [45], DF [37]
Constr. (NAR)	DeepACO [89], GFACS [38], GLOP [90]
Improvement	DACT [61], N2S [60], NeuOpt [59]
RL Algorithms	REINFORCE [79] A2C [41], PPO [74]
Transductive RL	Active search [3], EAS [31]

4.5 Additional Features

Configuration Management. RL4CO uses Hydra[88], a framework that enables hierarchical configuration management via Yaml files. Hydra makes it easy to define complex configurations, manage lots of experiments with different hyperparameter settings, and facilitate the reproducibility of experiments by freezing experimental setups in configuration files. We showcase an example experiment Yaml file with Hydra in Appendix B of the Appendix.

Decoding Schemes. Decoding schemes define the logic of translating from the model's logits (i.e., unnormalized log probabilities) to actions. Specifically, they handle the transformation from logits to probabilities $P(\mathcal{A})$ by masking infeasible actions and optionally applying tanh clipping [3] prior to softmax normalization. Subsequently, different decoding strategies can be employed to determine the action based on the probability distribution: 1) *Greedy*, which selects the action with the highest probability; 2) *Sampling*, which samples from the masked probability distribution of the policy, where different sampling strategies like softmax temperature scaling τ, top-k sampling [43], and top-p (or Nucleus) sampling [29] can be used; 3) *Multistart*, which enforces diverse starting actions as demonstrated in POMO [44], such as starting from different cities in the TSP; 4) *Augmentation*, which applies transformations to instances, such as random rotations in Euclidean problems [40], to create an augmented set of problems. We describe these strategies in detail in Appendix B.

Testing. Our comprehensive testing infrastructure includes continuous integration (CI) pipelines that automatically validate the codebase across multiple Python versions and operating systems. Each commit and pull request undergoes automated testing to ensure compatibility and maintain code quality. We enforce code standards with pre-commit hooks for linting, while automated coverage reports help maintain high test coverage across the codebase.

Documentation & Tutorials. We release extensive documentation[2] to make RL4CO as accessible as possible for both newcomers and expert researchers. Several tutorials and examples are also available under the `examples` folder of our publicly available code[3] that can also be deployed on Google Colab [10]. Finally, we showcase the low entry barrier for RL4CO with the following code snippet, which can train a model in under 20 lines of code:

```python
from rl4co.envs.routing import TSPEnv, TSPGenerator
from rl4co.models import AttentionModelPolicy, POMO
from rl4co.utils import RL4COTrainer
# Instantiate generator and environment
generator = TSPGenerator(num_loc=100, loc_distribution="uniform")
env = TSPEnv(generator)
# Create policy and RL model
policy = AttentionModelPolicy(env_name=env.name)
model = POMO(env, policy, batch_size=64)
# Instantiate Trainer and fit
trainer = RL4COTrainer(epochs=10, precision="16-mixed")
trainer.fit(model)
```

5 Benchmarking Study

We showcase our unified RL4CO library by performing several benchmarking studies. Due to the extent of our benchmark, we report highlights of the results in the following with experimental setup and further details available in Appendix B. We refer the reader to Appendix B for additional experiments.

Flexibility and Modularity. The integration of many state-of-the-art (SOTA) methods in RL4CO from NCO in a modular framework makes it easy to implement and improve upon SOTA neural solvers for complex CO problems with only few lines of code[4].

Table 5.1: Solutions obtained with RL4CO for the FJSSP with different model configurations.

Encoder + Decoder	FJSSP 10×5		FJSSP 20×5	
	Obj.	Gap	Obj.	Gap
HGNN + MLP (g.) [77]	111.82	15.8%	211.21	12.1%
MatNet + MLP (g.)	103.91	7.6%	197.92	5.0%
MatNet + Pointer (g.)	101.17	4.8%	196.30	4.2%
MatNet + Pointer (s. x128)	**98.31**	**1.8%**	**192.02**	**1.9%**

We demonstrate this in Table 5.1 for the FJSSP for different configurations by replacing or adding elements to the original SOTA policy [77]. Replacing the HGNN encoder with the more expressive MatNet encoder [45] already improves the average makespan by around 7%. Further, replacing the MLP decoder with the pointer mechanism from the AM model [42] reduces the optimality gap to roughly one-third of Song et al. [77], even when using a greedy (g.) decoding strategy, with sampling (s.) 128 solutions further improving this result.

Mind Your Baseline. In on-policy RL, which is often employed in RL for CO due to fast reward function evaluations, several different REINFORCE baselines – which help reduce variance in gradient estimates by subtracting an expected reward estimate from the actual reward – have been proposed to improve the performance [40, 42, 44]. We benchmark several RL algorithms training constructive policies for routing problems of node size 50, whose underlying architecture is based on the encoder-decoder Attention Model [42]

[2] https://rl4co.ai4co.org
[3] https://github.com/ai4co/rl4co/tree/main/examples
[4] The different model configurations shown here can be obtained by simply changing the Hydra configuration file like the one shown in Appendix B.

and whose main difference lies in how the REINFORCE baseline is calculated. For a fair comparison, we run all baselines in controlled settings with the same number of optimization steps and report results in Table 5.2. The performance of A2C, i.e., with a learned critic baseline, is generally inferior to other baselines. This can be explained by the inherent challenge of estimating the value of a problem instance $\boldsymbol{x}$ based on the sparse reward, which is only observed after solving an entire instance in routing problems. We found similar trends regarding actor-critic methods as A2C and PPO in the EDA mDPP problem [37], which we report in Appendix B. Namely, a greedy rollout baseline [42] can do better than value-based methods due to the challenging task of instance value estimation, i.e., to know the value of an instance solution, the neural network should implicitly "solve" the problem in one shot, which is arguably cumbersome.

Table 5.2: Gaps obtain with different RL algorithms. Best in bold, second best underlined.

Method	TSP	CVRP	OP	PCTSP	PDP
A2C	2.22	7.09	8.64	14.96	10.02
AM (Rollout)	1.41	5.30	<u>4.40</u>	<u>2.46</u>	<u>9.88</u>
POMO	<u>0.89</u>	**3.99**	14.26	11.61	10.64
Sym-NCO	**0.47**	<u>4.61</u>	**3.09**	**2.12**	**7.73**

Interestingly, while POMO [44], which takes as a baseline the average reward over all routes forcing each starting node to be different, may work well as baselines for problems in which near-optimal solutions can be constructed from any node (e.g., TSP), this may not be true for other problems such as the Orienteering Problem (OP): the reason is that in OP only a subset of nodes should be selected in an optimal solution, while several states will be discarded. Hence, forcing the policy to select all of them makes up for a poor baseline. We remark that while SymNCO (whose shared baseline involves symmetric rotations and flips) [40] may perform well in Euclidean problems, it is not applicable in non-Euclidean CO, including asymmetric routing problems and scheduling.

Decoding Schemes. The solution quality of different solvers often shows significant improvements in performance with different decoding schemes. We evaluate the pre-trained solvers with different strategies and settings as shown in Fig. 5.1. Interestingly, we note that augmentations are generally more sample-efficient than default sampling. Sampling can, however, be further improved by studying the sensitivity of advanced techniques such as temperature conditioning and top-p sampling.

Generalization. Using RL4CO, we can easily evaluate the generalization performance of existing baselines by employing supported environments that incorporate various VRP variant tasks [57] and data distributions [7] (termed MTPOMO and MDPOMO, respectively). Empirical results on CVRPLib [54], shown in Table 5.3, reveal that training on different tasks significantly enhances generalization performance. This finding underscores the promise of building foundation models across diverse CO domains, which could open up the possibility of cross-task and cross-distribution generalization with real-world applications.

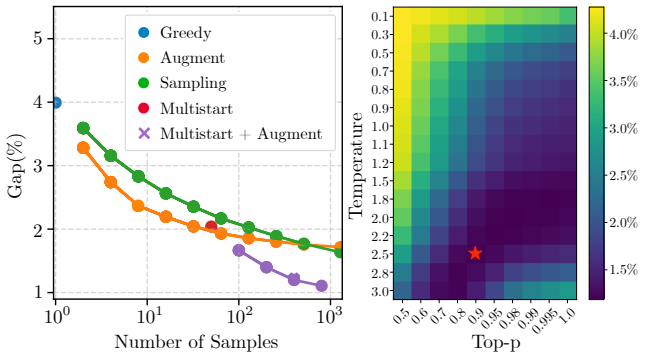


Figure 5.1: Study of decoding schemes using POMO on CVRP50. [Left]: Pareto front of decoding schemes by the number of samples; [Right]: sampling performance with different temperatures τ and p values for top-p sampling.

Table 5.3: Results on CVRPLIB with models trained on $N = 50$. Greedy multi-start decoding is used.

	POMO		MTPOMO		MDPOMO	
	Obj.	Gap	Obj.	Gap	Obj.	Gap
Set A	1075	3.13%	1076	3.20%	**1074**	**2.97%**
Set B	996	3.41%	1003	4.06%	**995**	**3.26%**
Set E	761	5.04%	**760**	**4.82%**	762	5.07%
Set F	813	13.52%	**798**	**12.09%**	825	13.66%
Set M	1259	16.37%	**1234**	**13.58%**	1263	16.03%
Set P	620	6.72%	**608**	**3.72%**	613	5.04%
Set X	73953	16.80%	**73763**	**16.69%**	81848	23.69%

Large-Scale Instances. We also evaluate large-scale CVRP instances of thousands of nodes (visualizations and more scaling in Appendix B). The last row of Table 5.4 illustrates the performance of the hybrid NAR/AR GLOP [90], while others refer to reproduced results from Ye et al. [90]. Our implementation in RL4CO improves the performance in not only speed but also solution quality.

Table 5.4: Performance on large-scale CVRP instances with thousands of nodes. Solution time reported in seconds.

	CVRP1K		CVRP2K		CVRP7K	
	Obj.	Time	Obj.	Time	Obj.	Time
LKH-3	46.4	6.2	64.9	20	245.0	501
AM	61.4	0.6	114.4	1.9	354.3	26
TAM (AM)	50.1	0.8	74.3	2.2	233.4	26
TAM (LKH-3)	46.3	1.8	64.8	5.6	196.9	33
GLOP-G (AM)†	47.1	0.4	63.5	1.2	191.7	2.4
GLOP-G (LKH-3)†	45.9	1.1	63.0	1.5	191.2	5.8
GLOP-G (AM)	46.9	**0.3**	64.7	**0.7**	190.9	**2.0**
GLOP-G (LKH-3)	**45.5**	0.5	**62.8**	0.8	**190.1**	3.9

Combining Construction and Improvement Methods. While constructive policies can build solutions in seconds, their performance is often limited, even with advanced decoding schemes

such as sampling or augmentations. On the other hand, improvement methods are more suitable for larger computing budgets.

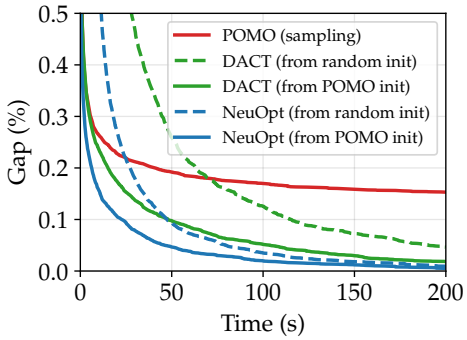


Figure 5.2: Bootstrapping improvement with constructive methods yields the best of both approaches.

We benchmark models on TSP with 50 nodes: the AR constructive method POMO [44] and the improvement methods DACT [61] and NeuOpt [59]. In the original implementation, DACT and NeuOpt started from a solution constructed randomly. To further demonstrate the flexibility of RL4CO, we show that bootstrapping improvement methods with constructive ones enhance convergence speed. Fig. 5.2 shows that bootstrapping with a pre-trained POMO policy significantly enhances the convergence speed. To further investigate the performance, we report the Primal Integral (PI) [5, 80, 82], which evaluates the evolution of solution quality over time. Improvement methods alone, such as DACT and NeuOpt, achieve 2.99 and 2.26, respectively, while sampling from POMO achieves 0.08. This shows that the "area under the curve" can be better even if the final solution is worse for constructive methods. Bootstrapping with POMO then improves DACT and NeuOpt to 0.08 and 0.04, respectively, showing the benefits of modularity and hybridization of different components thanks to RL4CO.

6 Discussion

Long-term Plans RL4CO is an actively maintained library that has gained significant traction in the research community, with over 500 GitHub stars. Our vision is to establish RL4CO as the primary benchmarking library for reinforcement learning approaches to combinatorial optimization. We maintain active engagement with the community through issue resolution and ongoing discussions. We anticipate that this work will catalyze new research directions and collaborations in the neural combinatorial optimization field. Further details are provided in Appendix A.

Figure 6.1: The RL4CO benchmark library logo.

Open License All RL4CO contents, including our logo shown in Fig. 6.1, are released under the MIT license, adhering to the principles of free and open-source software [26].

Open Community Through this project, we established the AI4CO community, a non-profit, cross-institutional, and inclusive research initiative. AI4CO initially originated as a Slack channel for RL4CO discussions, and it has evolved into a broader platform for neural combinatorial optimization research, which now hosts more than 250 researchers. We encourage interested researchers to join our community.

Limitations and Future Directions While RL4CO provides an efficient and modular framework for combinatorial optimization problems, its specialized optimizations may limit direct integration with general frameworks like OpenAI Gym without modifications. The current scope of the library primarily focuses on reinforcement learning approaches. Future work could involve developing a comprehensive "AI4CO" codebase that incorporates supervised learning methods to benefit the broader neural combinatorial optimization community. Additionally, we identify foundation models for combinatorial optimization and scalable architectures as promising research directions for addressing cross-task and distribution generalization challenges, as discussed in Appendix B.

7 Conclusion

This paper introduces RL4CO, a modular, flexible, and unified Reinforcement Learning (RL) for Combinatorial Optimization (CO) benchmark. We provide a comprehensive taxonomy from environments to policies and RL algorithms that translate from theory to practice to software level. Our benchmark library aims to fill the gap in unifying implementations in RL for CO by utilizing several best practices with the goal of providing researchers and practitioners with a flexible starting point for NCO research. We provide several experimental results with insights and discussions that can help identify promising research directions. We hope that our open-source library will provide a solid starting point for NCO researchers to explore new avenues and drive advancements. We warmly welcome researchers and practitioners to actively participate and contribute to RL4CO.

Acknowledgments

We thank those anonymous reviewers whose feedback significantly improved our paper.

We also thank contributors from the AI4CO community and the TorchRL team for their support. We welcome future collaborations and thank in advance for contributions to RL4CO through bug reports, feature requests, or research ideas.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00410082), the Institute of Information & Communications Technology Planning & Evaluation (IITP)-Innovative Human Resource Development for Local Intellectualization program grant funded by the Korea government(MSIT)(IITP-2025-RS-2024-00436765), and the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031). Minsu Kim was supported by KAIST Jang Yeong Sil Fellowship and the Canadian AI Safety Institute Research Program at CIFAR through a Catalyst award. Nayeli Gast Zepeda and André Hottung were supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - No. 521243122.

References

[1] Bharathan Balaji, Jordan Bell-Masterson, Enes Bilgin, Andreas Damianou, Pablo Moreno Garcia, Arpit Jain, Runfei Luo, Alvaro Maggiar, Balakrishnan Narayanaswamy, and Chun Ye. 2019. Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641* (2019).

[2] Egon Balas. 1989. The prize collecting traveling salesman problem. *Networks* 19, 6 (1989), 621–636.

[3] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. arXiv:1611.09940 [cs.AI]

[4] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research* 290, 2 (2021), 405–421.

[5] Timo Berthold. 2013. Measuring the impact of primal heuristics. *Operations Research Letters* 41, 6 (2013), 611–614.

[6] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Kevin Tierney, and Jinkyoo Park. 2024. RouteFinder: Towards Foundation Models for Vehicle Routing Problems. *Arxiv* (2024).

[7] Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. 2022. Learning generalizable models for vehicle routing problems via knowledge distillation. *Advances in Neural Information Processing Systems* 35 (2022), 31226–31238.

[8] Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. 2024. Learning to handle complex constraints for vehicle routing problems. *Advances in Neural Information Processing Systems* (2024).

[9] David Biagioni, Charles Edison Tripp, Struan Clark, Dmitry Duplyakin, Jeffrey Law, and Peter C St John. 2022. graphenv: a Python library for reinforcement learning on graph search spaces. *Journal of Open Source Software* 7, 77 (2022), 4621.

[10] Ekaba Bisong and Ekaba Bisong. 2019. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners* (2019), 59–64.

[11] Lawrence Bodin. 1983. Routing and scheduling of vehicles and crews. *Computer & Operations Research* 10, 2 (1983), 69–211.

[12] Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. 2024. Jumanji: a Diverse Suite of Scalable Reinforcement Learning Environments in JAX. In *International Conference on Learning Representations*.

[13] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. 2024. TorchRL: A data-driven decision-making library for PyTorch. In *International conference on learning representations*. arXiv:2306.00577 [cs.LG]

[14] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs. http://github.com/google/jax

[15] Paolo Brandimarte. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research* 41, 3 (1993), 157–183.

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[17] I-Ming Chao, Bruce L Golden, and Edward A Wasil. 1996. A fast and effective heuristic for the orienteering problem. *European journal of operational research* 88, 3 (1996), 475–489.

[18] Xinyun Chen and Yuandong Tian. 2019. Learning to Perform Local Rewriting for Combinatorial Optimization. In *Advances in Neural Information Processing Systems*.

[19] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[20] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[21] DeepSeek-AI and DeepSeek-R1 Team. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025). doi:10.48550/arXiv.2501.12948

[22] Marco Dorigo and Thomas Stützle. 2019. *Ant colony optimization: overview and recent advances*. Springer.

[23] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. 2023. BQ-NCO: Bisimulation Quotienting for Generalizable Neural Combinatorial Optimization. *Advances in Neural Information Processing Systems* (2023).

[24] Zvi Drezner and Horst W Hamacher. 2004. *Facility location: applications and theory*. Springer Science & Business Media.

[25] William Falcon and The PyTorch Lightning team. 2019. PyTorch Lightning. doi:10.5281/zenodo.3828935

[26] Joseph Feller. 2005. *Perspectives on free and open source software*. MIT Press.

[27] LLC Gurobi Optimization. 2021. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[28] Keld Helsgaun. 2017. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. *Roskilde: Roskilde University* (12 2017). doi:10.13140/RG.2.2.25569.40807

[29] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751* (2019).

[30] André Hottung, Bhanu Bhandari, and Kevin Tierney. 2020. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*.

[31] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. 2022. Efficient active search for combinatorial optimization problems. *International conference on learning representations* (2022).

[32] André Hottung, Mridul Mahajan, and Kevin Tierney. 2025. PolyNet: Learning Diverse Solution Strategies for Neural Combinatorial Optimization. In *International Conference on Learning Representations*.

[33] Christian D Hubbs, Hector D Perez, Owais Sarwar, Nikolaos V Sahinidis, Ignacio E Grossmann, and John M Wassick. 2020. OR-Gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319* (2020).

[34] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227* (2019).

[35] Bahman Kalantari, Arthur V Hill, and Sant R Arora. 1985. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research* 22, 3 (1985), 377–386.

[36] Samir Khuller, Anna Moss, and Joseph Seffi Naor. 1999. The budgeted maximum coverage problem. *Information processing letters* 70, 1 (1999), 39–45.

[37] Haeyeon Kim, Minsu Kim, Federico Berto, Joungho Kim, and Jinkyoo Park. 2023. DevFormer: A Symmetric Transformer for Context-Aware Device Placement. *International Conference on Machine Learning* (2023). arXiv:2205.13225 [cs.LG]

[38] Minsu Kim, Sanghyeok Choi, Jiwoo Son, Hyeonah Kim, Jinkyoo Park, and Yoshua Bengio. 2025. Ant Colony Sampling with GFlowNets for Combinatorial Optimization. In *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, Vol. 258. 469–477.

[39] Minsu Kim, Jinkyoo Park, and Joungho Kim. 2021. Learning Collaborative Policies to Solve NP-hard Routing Problems. In *Advances in Neural Information Processing Systems*.

[40] Minsu Kim, Junyoung Park, and Jinkyoo Park. 2022. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems* (2022).

[41] Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. *Advances in neural information processing systems* 12 (1999).

[42] Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Attention, learn to solve routing problems! *International Conference on Learning Representations* (2019).

[43] Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*. PMLR, 3499–3508.

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

[45] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. 2021. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems* 34 (2021), 5138–5149.

[46] Gilbert Laporte and Silvano Martello. 1990. The selective travelling salesman problem. *Discrete applied mathematics* 26, 2-3 (1990), 193–207.

[47] EL Lawler, JK Lenstra, AHG Rinnooy Kan, and DB Shmoys. 1986. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. *The Journal of the Operational Research Society* 37, 5 (1986), 535.

[48] Jingwen Li, Yining Ma, Zhiguang Cao, Yaoxin Wu, Wen Song, Jie Zhang, and Yeow Meng Chee. 2023. Learning Feature Embedding Refiner for Solving Vehicle Routing Problems. *IEEE Transactions on Neural Network and Learning Systems* (2023).

[49] Jingwen Li, Liang Xin, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. 2021. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems* 23, 3 (2021), 2306–2315.

[50] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[51] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. 2024. From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems* 36 (2024).

[52] Yang Li, Jiale Ma, Wenzheng Pan, Runzhong Wang, Haoyu Geng, Nianzu Yang, and Junchi Yan. 2025. Streamlining the Design Space of ML4TSP Suggests Principles for Learning and Search. In *International Conference on Learning Representations*.
[53] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. 2017. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381* 85 (2017).
[54] Ivan Lima, Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, Anand Subramanian, Richard W, Daniel Oliveira, and Eduardo Queiroga. 2014. CVRPLIB: Capacitated Vehicle Routing Problem Library. http://vrp.galgos.inf.puc-rio.br/index.php/en/ Last checked on October 6, 2024.
[55] Jeffrey T Linderoth, Andrea Lodi, et al. 2010. MILP software. *Wiley encyclopedia of operations research and management science* 5 (2010), 3239–3248.
[56] Chang Liu, Runzhong Wang, Jiayi Zhang, Zelin Zhao, Haoyu Geng, Tianzhe Wang, Wenxuan Guo, Wenjie Wu, Nianzu Yang, Ziao Guo, Yang Li, Hao Xiong, and Junchi Yan. 2025. Awesome Machine Learning for Combinatorial Optimization. https://github.com/Thinklab-SJTU/awesome-ml4co Accessed: February 23, 2025.
[57] Fei Liu, Xi Lin, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. 2024. Multi-Task Learning for Routing Problem with Cross-Problem Zero-Shot Generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.
[58] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. 2024. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems* 36 (2024).
[59] Yining Ma, Zhiguang Cao, and Yeow Meng Chee. 2024. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *Advances in Neural Information Processing Systems* 36 (2024).
[60] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Hongliang Guo, Yuejiao Gong, and Yeow Meng Chee. 2022. Efficient Neural Neighborhood Search for Pickup and Delivery Problems. *arXiv preprint arXiv:2204.11399* (2022).
[61] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. 2021. Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer. *Advances in Neural Information Processing Systems* 34 (2021).
[62] Rafael Mart, Panos M Pardalos, and Mauricio GC Resende. 2018. *Handbook of heuristics*. Springer Publishing Company, Incorporated.
[63] Yimeng Min, Yiwei Bai, and Carla P Gomes. 2023. Unsupervised Learning for Solving the Travelling Salesman Problem. In *Neural Information Processing Systems*.
[64] Vincent Moens. 2023. TensorDict: your PyTorch universal data carrier. https://github.com/pytorch-labs/tensordict
[65] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. 2018. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems* 31 (2018).
[66] Hyunah Park, Haeyeon Kim, Hyunwoo Kim, Joonsang Park, Seonguk Choi, Jihun Kim, Keeyoung Son, Haeseok Suh, Taesoo Kim, Jungmin Ahn, et al. 2023. Versatile Genetic Algorithm-Bayesian Optimization (GA-BO) Bi-Level Optimization for Decoupling Capacitor Placement. In *2023 IEEE 32nd Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*. IEEE, 1–3.
[67] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
[68] Laurent Perron and Vincent Furnon. 2023. OR-Tools. https://developers.google.com/optimization/
[69] Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chételat, and Andrea Lodi. 2020. Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*. https://openreview.net/forum?id=IVc9hqgibyB
[70] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. http://jmlr.org/papers/v22/20-1364.html
[71] Graham K. Rand. 1982. Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop. *Journal of the Operational Research Society* 33 (1982), 862. https://api.semanticscholar.org/CorpusID:62592932
[72] Gerhard Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing* 3, 4 (1991), 376–384.
[73] Martin WP Savelsbergh and Marc Sol. 1995. The general pickup and delivery problem. *Transportation science* 29, 1 (1995), 17–29.
[74] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[75] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359.
[76] Jialin Song, Yisong Yue, Bistra Dilkina, et al. 2020. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems* 33 (2020), 20012–20023.
[77] Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. 2022. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics* 19, 2 (2022), 1600–1610.
[78] Zhiqing Sun and Yiming Yang. 2023. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, Vol. 36. 3706–3731.
[79] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12 (1999).
[80] Daniela Thyssens, Tim Dernedde, Jonas K Falkner, and Lars Schmidt-Thieme. 2023. Routing Arena: A Benchmark Suite for Neural Routing Solvers. *arXiv preprint arXiv:2310.04140* (2023).
[81] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[82] Thibaut Vidal. 2022. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research* 140 (2022), 105643.
[83] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems*, Vol. 28. 2692–2700.
[84] Ching Pui Wan, Tung Li, and Jason Min Wang. 2023. RLOR: A Flexible Framework of Deep Reinforcement Learning for Operation Research. *arXiv preprint arXiv:2303.13117* (2023).
[85] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. 2022. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research* 23, 267 (2022), 1–6.
[86] Niels A Wouda, Leon Lan, and Wouter Kool. 2024. PyVRP: A high-performance VRP solver package. *INFORMS Journal on Computing* (2024).
[87] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2021. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems* 33, 9 (2021), 5057–5069.
[88] Omry Yadan. 2019. Hydra - A framework for elegantly configuring complex applications. Github. https://github.com/facebookresearch/hydra
[89] Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. 2023. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. *arXiv preprint arXiv:2309.14032* (2023).
[90] Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. 2024. 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*, Vol. 38. 20284–20292.
[91] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. 2020. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems* (2020).
[92] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. 2023. Let the Flows Tell: Solving Graph Combinatorial Problems with GFlowNets. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 11952–11969.
[93] Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Chi Xu. 2024. MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts. In *International Conference on Machine Learning*.

APPENDIX

A RL4CO: Vision and Software

A.1 Why Choosing the RL4CO Library?

RL4CO is a *unified* and *extensive* benchmark for the RL-for-CO research domain, designed to be accessible and valuable to researchers and practitioners across all levels of expertise.

Availability and Future Support RL4CO can be installed through PyPI. We adhere to continuous integration, deployment, and testing to ensure reproducibility and accessibility.

Open License We adopt the open MIT license for all content contained in RL4CO. We ascribe to the principles of *libre software*[5]. Most reimplementations are from original authors and are re-licensed under the MIT license. Data and baseline-specific licenses are reported in Appendix A.3.

Figure A.1: Unofficial - but widely used - open MIT license logo.

Open Community Through our journey, we started the AI4CO community[6], which is a non-profit, cross-institution, inclusive, and open research community. AI4CO originally started out as a Slack channel for discussing the RL4CO but evolved into a broader-visioned and inclusive space to communicate with other researchers about general NCO. We warmly invite all interested people to join us.

A.2 On the Choice of the Software

During the development of RL4CO, we wanted to make it as simple as possible to integrate reproducible and standardized code adhering to the latest guidelines. As a main template for our codebase, we use Lightning-Hydra-Template [7] which we believe is a solid starting point for reproducible deep learning. We further discuss framework choices below.

PyTorch PyTorch [67] is a popular open-source deep-learning framework that has gained significant traction in the research community. We chose PyTorch as the primary framework for RL4CO due to its intuitive API, dynamic computational graphs, strong community support, and seamless integration with the Python ecosystem. These features make PyTorch well-suited for rapid prototyping and experimentation, which are essential in research settings. Moreover, most of the existing research in NCO has been implemented. It is currently being implemented using PyTorch, making it not only easier to build upon and compare with previous work but also easier for newcomers and experienced researchers.

TorchRL and TensorDict One of the software hindrances in RL is the bottleneck between CPU and GPU communication, majorly due to CPU-based operating environments. For this reason, we did not opt for OpenAI Gym [16] since, although it includes some level of parallelization, this does not happen on GPU and would thus greatly hinder performance. Kool et al. [42] creates *ad-hoc* environments in PyTorch to handle batched data efficiently. However, it could be cumbersome to integrate into standardized routines that include `step` and `reset` functions. As we searched for a better alternative, we found that TorchRL library [13], an official PyTorch project that allows for efficient batched implementations on (multiple) GPUs as well as functions akin to OpenAI Gym. We also employ the TensorDict [13] to handle tensors efficiently on multiple keys (i.e. in CVRP, we can directly operate transforms on multiple keys as locations, capacities, and more). This makes our environments compatible with the models in TorchRL, which we believe could further spread interest in the CO area.

PyTorch Lightning PyTorch Lightning [25] is a useful tool for abstracting away the boilerplate code, allowing researchers and practitioners to focus more on the core ideas and innovations. It features a standardized training loop and an extensive set of prebuilt components, including automated checkpointing, distributed training, and logging. PyTorch Lightning accelerates development time and facilitates scalability. We employ PyTorch Lightning in RL4CO to integrate with the PyTorch ecosystem – which includes TorchRL – enabling us to leverage the rich set of tools and libraries available.

Hydra Hydra [88] is a powerful open-source framework for managing complex configurations in machine-learning models and other software. Hydra facilitates creating hierarchical configurations, making it easy to manage even very large and intricate configurations. Moreover, it integrates with command-line interfaces, allowing the execution of different configurations directly from the command line, thereby enhancing reproducibility. We found Hydra to be effective when dealing with multiple experiments since configurations are saved both locally, as `yaml` files, and can be uploaded to monitoring software as Wandb (https://wandb.ai/) (or to any of the monitoring software supported by PyTorch Lightning).

A.3 Licenses

Table A.1: Reference code licenses and links.

Type	Asset	License	Link
Library	PyTorch [67]	BSD-3 License	link
	PyTorch Lightning [25]	Apache-2.0 License	link
	TorchRL+TensorDict [13]	MIT License	link
	Hydra [88]	MIT License	link
Dataset	TSPLIB [72]	Available for any non-commercial use	link
	CVRPLib [54]	Available for any non-commercial use	link
	DPP PDNs [66]	Apache-2.0	link
Solver	PyVRP [86]	MIT	link
	LKH3 [28]	Available for any non-commercial use	link
	OR-Tools [68]	Apache 2.0 License	link

We summarize the license of software that we employ in RL4CO in a non-exhaustive list in Table A.1. Original environments and models from the authors are acknowledged through their respective citations, with several links available in the library. RL4CO is licensed under the MIT license.

B Full Paper on arXiv

The full paper with its extensive Appendix – which covers additional details and experiments spanning several pages (around 40) – is available online at https://arxiv.org/abs/2306.17100.

[5] https://www.gnu.org/philosophy/free-sw.en.html
[6] https://github.com/ai4co
[7] https://github.com/ashleve/lightning-hydra-template