STRUCTURED VALUE FUNCTION PARAMETERIZATION FOR NEURAL ROUTING SOLVERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning heuristics for combinatorial optimization problems through reinforcement learning and deep neural networks has seen a surge of interest in recent years. We revisit the use of value functions in this setting, as many implementations still rely on simple average reward baselines, computed from multiple samples of the same instance. For the class of binary programs with linear objectives, we formulate a Markov decision process, where the policy sequentially assigns the decision variables, and show that its value function decomposes exactly into the cost-weighted sum of the marginal inclusion probabilities of the undecided variables. We utilize this insight as an architectural prior by parameterizing the critic to predict those probabilities rather than a scalar directly. Our experiments show that the structured parameterization improves policy performance and reduces sensitivity to the critic learning rate, achieving state-of-the-art results on the challenging multi-task vehicle routing problem environment, consisting of 16 routing variants. Additionally, we show that the learned value function can be used at inference time to guide a truncated rollout search, which produces favorable results on the Pareto front for longer time budgets. Finally, our MDP framework also reveals that some states can be automatically stepped through when the constraints of the problem imply a single legal action. Such states contribute no policy gradient and, especially during early training, account for a substantial share of all states, leading to considerable efficiency gains.

1 INTRODUCTION

We are concerned with training deep neural networks to automatically learn heuristics for hard combinatorial optimization problems. This research area, known as neural combinatorial optimization (NCO), has attracted growing interest in domains such as vehicle routing (Kool et al., 2019; Kwon et al., 2020; Berto et al., 2025b), scheduling (Zhang et al., 2020; Kwon et al., 2021), assignment problems (Bagga & Delarue, 2023) and graph problems (Khalil et al., 2017; Sun & Yang, 2023).

One of the major paradigms is to learn a neural network that constructs a solution via sequential decision-making. Many of these works rely on the use of reinforcement learning (RL), given that no optimal solutions are easily available for training. However, the utilized RL algorithms often remain simple in their use of value estimation. They typically use an encoder-decoder architecture as their policy, trained with the REINFORCE algorithm (Williams, 1992), and perform multiple rollouts for a single instance, using the average reward across all rollouts as the value estimate (Kwon et al., 2020; 2021; Kim et al., 2022; Grinsztajn et al., 2023; Zhou et al., 2024; Liu et al., 2024; Berto et al., 2025b; Li et al., 2025; Pan et al., 2025; Gui et al., 2026).

While this provides a reasonable baseline for an instance, it ignores the step-by-step differences among states, essentially treating the entire solution construction as a single compound action and thus only providing a value estimate for the initial state, which represents the problem instance. The immediate consequence is an absence of credit assignment: every action along a trajectory receives the exact same advantage, which does not distinguish between a poor decision in an otherwise high-quality solution and a good decision in an otherwise poor solution. While this does not fundamentally bias the optimization process, learning must rely on this signal averaging out over a large number of trajectories. Additionally, it precludes temporal difference learning and bootstrapped ad-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

vantage estimation, both of which are otherwise considered standard tools for efficient and effective learning in the broader RL literature (Sutton & Barto, 2018; Schulman et al., 2016).

Even early work that investigated the use of actor-critic methods (Bello et al., 2017; Nazari et al., 2018; Kool et al., 2019) limited the critic to a baseline value for the entire instance. Kool et al. (2019) compare this to a greedy rollout baseline and adopt the latter, and the routing literature subsequently largely settled on rollout-based baselines, in particular the shared baseline averaging multiple rollouts of the same instance (Kwon et al., 2020). This issue is even discussed in the popular NCO benchmarking suite RL4CO (Berto et al., 2025a), which states that algorithms that use a learned critic are generally inferior, due to the difficulty of scoring the entire instance in one shot. However, we argue that this difficulty is specific to valuing only the instance, and that value functions can equally be evaluated for intermediate states, i.e. partial solutions.

In this work, we therefore reinvestigate the use of value functions in neural combinatorial optimization. We provide an MDP formulation for binary programs with linear objectives, based on the progressive assignment of the decision variables. We show that for this MDP the true value function decomposes as the cost-weighted sum of inclusion probabilities of the decision variables under the policy. We therefore propose to parameterize the value function with an equivalently structured output head, along with a second variant that additionally accounts for constraint structures which impose mutually exclusive choices on disjoint subsets of the variables. Both variants add a strong inductive architectural prior to the critic networks. We then use the proximal policy optimization algorithm (Schulman et al., 2017) with our proposed value functions to train a policy across a broad set of 16 different vehicle routing problems. Our final model achieves state-of-the-art performance among learned construction methods: under the same multi-start and augmentation decoding as the baselines, it reduces the average gap to PyVRP by more than 60% compared to the second-best model.

Based on our MDP framework, we also show that some states can be automatically stepped through when the constraints of the problem logically imply a single legal action. This removes 17.7–20.9% of the states processed during rollouts at the start of training and 5.7–7.3% for the converged policy. These states are moreover exactly the ones that contribute no policy gradient, so skipping them is not only cheaper but also removes a source of variance in the effective batch size.

Finally, we investigate an additional benefit of the already trained value function by using it at test time to score candidate actions. We utilize a truncated rollout procedure (Bertsekas et al., 1997; Bertsekas, 2020), in which each candidate action is scored by simulating the policy for h further steps and adding the learned value estimate of the state reached. Our contributions can be summarized as follows:

- We propose two novel structured value function parameterizations for binary programs with linear objectives, which estimate the inclusion probability of each decision variable and compute the expected return analytically as their cost-weighted sum. The second variant additionally models constraints that impose mutually exclusive choices on disjoint subsets of the variables. On the 16 MTVRP variants, this improves robustness to the critic learning rate, calibration at initialization, and overall policy performance.
- We introduce auto-stepping to the MTVRP environment, subsuming decisions that are logically implied by the constraints into the step function. Such states provide no policy gradient, so removing them introduces no bias, and they account for up to 20.9% of all states processed early in training.
- We show that the learned value functions are also useful at test time: truncating a rollout search after h steps and closing the remaining horizon with the value function substantially accelerates it compared to untruncated rollouts, with only a small loss in solution quality.

2 Related Work

Neural routing solvers. Constructive solvers build a solution by sequentially adding one node at a time to a partial solution until completion. Since early work (Bello et al., 2017; Nazari et al., 2018; Kool et al., 2019), these solvers typically use an encoder to embed the instance once and a light decoder that is queried at every step. More recently, decoder-only models that re-embed the remaining problem at every step (Drakulic et al., 2023; Pirnay & Grimm, 2024) have been proposed, trained

either by imitation of solver-generated solutions or by self-improvement, in which the model's own best sampled solutions become the next supervision targets. Building on POMO (Kwon et al., 2020), MTPOMO (Liu et al., 2024) and MVMoE (Zhou et al., 2024) train a single model on five and six VRP variants, respectively, and evaluate further variants zero-shot. Berto et al. (2025b) introduced the MTVRP environment with 48 possible variants but trained jointly on 16 of them, which we and subsequent work also use for training and evaluation. CaDA (Li et al., 2025) adds a dual-stream architecture with sparse attention, $\text{MoSES}_{\text{CaDA}}$ (Pan et al., 2025) adds a mix of low-rank adapter experts, one per basis variant, on top, and finally CCL (Gui et al., 2026) shares the context of multiple parallel rollouts of an instance to update the node embeddings. We adopt the MTVRP environment and benchmark suite but forgo many of the specialized architecture decisions and follow the decoder-only models for our actor and critic. Unlike them, however, we train with standard RL techniques, using our proposed structured value functions.

Value estimation. As discussed in the introduction, value estimation in this domain was often reduced to scoring only the instance and not any intermediate states. As such, early critics (Bello et al., 2017; Nazari et al., 2018) were replaced by rollout baselines (Kool et al., 2019), most prominently POMO (Kwon et al., 2020), which is still adopted by many current methods (Kim et al., 2022; Zhou et al., 2024; Liu et al., 2024; Li et al., 2025; Berto et al., 2025b; Pan et al., 2025; Gui et al., 2026). Per-state critics appear occasionally outside routing: L2D (Zhang et al., 2020) trains a constructive job-shop dispatcher with PPO, and improvement methods, which move between feasible solutions and never value partial ones, do the same (Wu et al., 2022; Ma et al., 2021). Q-learning (Khalil et al., 2017) and approximate policy iteration (Delarue et al., 2020) likewise learn values of partial solutions. All of them use generic scalar heads, comparable to our mean-pool baseline. Another issue for value learning in the multi-task setting is different reward scales across VRP variants. While Berto et al. (2025b) handle this via per-task reward normalization, our structured parameterization allows us to bypass this additional complexity, since the network only outputs probabilities and then computes a value estimate directly via the known distances.

Edge probabilities. Heatmap methods also predict edge probabilities (Joshi et al., 2019; Fu et al., 2021; Qiu et al., 2022; Sun & Yang, 2023; Ye et al., 2023) and are thus structurally related to our value functions. However, they aim to directly predict whether a variable is part of the optimal solution and are trained mostly via supervised learning. The goal is often to use them as a search prior for a heuristic or directly as a policy, but to the best of our knowledge they have not been used to parameterize a value function in a reinforcement learning setting, training another policy model.

3 METHOD

3.1 PROBLEM FORMULATION AND MDP DEFINITION

We consider combinatorial optimization problems defined over a domain of binary decision variables governed by instance-specific linear objectives and constraints. Formally, a problem instance $I = (c_I, \mathcal{X}_I)$ consists of a non-negative cost vector[1] $c_I \in \mathbb{R}^n_{\geq 0}$ and a feasible set of assignments $\mathcal{X}_I \subseteq \{0,1\}^n$. An *assignment* is a binary decision vector $x \in \{0,1\}^n$ and represents a *solution* only if it is in the feasible set. The optimization goal is to find such a feasible solution $x \in \mathcal{X}_I$ that minimizes the instance's objective function $\min_{x \in \mathcal{X}_I} c_I^\top x$. This formulation covers a broad class of widely applicable combinatorial optimization problems, including routing, linear and generalized assignment, knapsack, covering problems and more. We aim to solve such problems via sequential decision-making in which a parameterized policy π_θ constructively builds a feasible solution vector x for a given instance I. Therefore we define an episodic Markov Decision Process (MDP) based on the progressive assignment of the elements in the decision vector:

State: The state s_t at step t consists of the static instance parameters I and the current partial assignment of the decision variables $x^{(t)} \in \{0, 1, \star\}^n$, where $\star$ denotes an undecided variable. For the initial state s_0, all variables are undecided: $x^{(0)} = \star^n$.

[1] Negative costs can be handled via variable substitution, as is standard in binary linear programming.

Action: Let $U(s_t) = \{i \mid x_i^{(t)} = \star\}$ be the set of indices of currently undecided variables and $D(s_t) = \{i \mid x_i^{(t)} \neq \star\}$ its complement. The policy selects an action $a_t \in \mathcal{A}(s_t) \subseteq U(s_t)$, which corresponds to setting $x_{a_t} = 1$. Let $\mathcal{X}_I(s_t, a_t) \subseteq \mathcal{X}_I$ be the subset of complete feasible assignments that remain compatible after committing to action a_t:

$$\mathcal{X}_I(s_t, a_t) = \left\{ x \in \mathcal{X}_I \;\middle|\; x_{a_t} = 1,\ \forall i \in D(s_t),\ x_i = x_i^{(t)} \right\} \tag{1}$$

The action masking function $\mathcal{A}(s_t) = \{a \in U(s_t) \mid \mathcal{X}_I(s_t, a) \neq \emptyset\}$ restricts the available action space to ensure that the chosen action can still lead to a feasible complete assignment. While generically any valid undecided variable could be picked, problem-specific implementations may restrict $\mathcal{A}(s_t)$ further to enforce a particular construction order. In our routing application, the decision variables represent directed graph edges, and the action space is restricted to sequential extensions of the active path, which reduces the choice set without restricting the solution space.

Transition: Upon taking action a_t, the environment updates the partial assignment to $x^{(t+1)}$. Naively, only the selected variable is set, $x_{a_t}^{(t+1)} = 1$, leaving all other variables unchanged. However, the constraint structure of $\mathcal{X}_I$ often logically implies the values of other variables (e.g., mutually exclusive choices). To account for the downstream implications of the action a_t, we can define the state update rule for each variable $i \in \{1, \ldots, n\}$ as:

$$x_i^{(t+1)} = \begin{cases} 1, & \text{if } \forall x \in \mathcal{X}_I(s_t, a_t),\ x_i = 1, \\ 0, & \text{if } \forall x \in \mathcal{X}_I(s_t, a_t),\ x_i = 0, \\ \star, & \text{otherwise.} \end{cases} \tag{2}$$

Both the direct execution of the action a_t and consistency with the previous partial assignment $x^{(t)}$ are naturally captured in $\mathcal{X}_I(s_t, a_t)$. If this is not easy to compute, we can fall back to the naive update rule, or only account for some of the downstream implications.

To illustrate, consider a routing problem where the decision variables correspond to edges between nodes. If the policy selects an edge a_t that connects node i to node j, then all other outgoing edges from node i and incoming edges to node j must be auto-assigned to 0, since each customer node can only be visited once. Additionally, if the vehicle has reached its capacity or duration limit, the next edge must return to the depot, setting the corresponding edge to 1. While the first implication is commonly implemented in routing environments, to the best of our knowledge the second is not, leading to an unnecessary increase in the number of states processed by the policy model. We fix this and refer to the mechanism as an **auto-step**.

Reward: The reward r_t reflects the incremental cost incurred during the transition. Let $\Delta^{(t)} = \{i \mid x_i^{(t)} = \star, x_i^{(t+1)} \neq \star\}$ be the set of variables that transitioned from undecided to decided. The reward is defined as the negative sum of the costs of these newly decided variables, $r_t = -\sum_{i \in \Delta^{(t)}} (c_I)_i x_i^{(t+1)}$.

Termination: The process terminates at step T when taking action a_T leaves all variables decided ($U(s_{T+1}) = \emptyset$) or when all remaining undecided variables can be safely set to 0 without violating feasibility, which is optimal due to the non-negativity of the cost vector. This also completes the final assignment, which we denote by $x^{(T+1)}$.

Assuming instances are drawn from a distribution $I \sim \mathcal{D}$, we arrive at the standard episodic reinforcement learning objective $\max_\theta J(\theta) = \mathbb{E}_{I \sim \mathcal{D},\, a_0, \ldots, a_T \sim \pi_\theta}\left[\sum_{t=0}^{T} r_t\right]$. Because $\sum_{t=0}^{T} r_t = -c_I^\top x^{(T+1)}$, maximizing the return is equivalent to minimizing the linear cost.

3.2 Structured Value Function Parameterization

Value estimation is critical for successful policy optimization. Given a policy π, the value function $V^\pi(s_t)$ is the expected return when starting in state s_t and following π thereafter. Since the remaining rewards sum to $-\sum_{i \in U(s_t)} (c_I)_i x_i^{(T+1)}$, linearity of expectation expresses it through the cost

and the marginal inclusion probability of each undecided variable:

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\Big[\sum_{k=t}^{T} r_k \Big| s_t\Big] = -\sum_{i \in U(s_t)} (c_I)_i \mathbb{E}_{\pi}\big[x_i^{(T+1)} \big| s_t\big] = -\sum_{i \in U(s_t)} (c_I)_i \mathbb{P}_{\pi}\big(x_i^{(T+1)} = 1 \big| s_t\big). \quad (3)$$

During training, the value function is most commonly approximated by a neural network with a generic scalar output. However, Equation 3 motivates parameterizing the value function approximator in a structured way, injecting domain knowledge about the specific MDP: rather than predicting a scalar value directly, the model estimates the inclusion probabilities of each undecided decision variable and computes the expected return analytically as the cost-weighted sum of these probabilities. Besides this inductive bias, the parameterization has two structural advantages. First, since probabilities lie in $[0, 1]$ and costs are non-negative, the predicted value always falls within *instance-specific and state-dependent bounds* $[-\sum_{i \in U(s_t)} (c_I)_i, 0]$. This prevents implausible out-of-range predictions and makes the critic informative already at initialization: where a generic head predicts a small value independent of instance and state, the structured head assigns states with fewer or cheaper remaining variables a smaller remaining cost (Appendix A). Second, the network's outputs have a fixed scale, while the magnitude of the value, which varies with the costs and the number of remaining variables, is carried by the known cost coefficients. This removes the need to normalize value targets, which is particularly helpful for multi-task training, where variants differ in cost scale.

3.2.1 Application to Multi-Task Vehicle Routing

To demonstrate our approach, we utilize the Multi-Task Vehicle Routing Problem (MTVRP) environment (Berto et al., 2025b), which subsumes 16 routing variants that extend the Capacitated Vehicle Routing Problem (CVRP) with combinations of open routes (O), backhaul customers (B), duration limits (L) and time windows (TW). Formally, an instance I features a static set of customer nodes $C = \{1, \dots, m\}$ and a depot node $\{0\}$. The union of these sets defines the complete set of vertices $V = C \cup \{0\}$, where $|V| = m+1$. The binary decision variables correspond to the directed edges connecting these vertices, yielding $n = |V|^2 = (m+1)^2$ total variables representing pairs $(i, j) \in V \times V$. Each edge is associated with a static distance cost d_{ij}. For more details on the MTVRP and its constraints, see Appendix D.

Following standard practice, our action space restricts the generic formulation of selecting any arbitrary edge at each time step. Instead, we require the policy to build routes by sequentially extending the path, starting at the depot node. To prevent the model from processing irrelevant information, the state representation omits nodes whose incident decision variables have already been fully resolved by the instance's constraints or prior actions. We define the set of relevant nodes at step t as $V_t \subseteq V$:

$$V_t = \{i \in V \mid \exists j \in V \text{ s.t. } (i, j) \in U(s_t) \vee (j, i) \in U(s_t)\} \quad (4)$$

We denote the cardinality of this remaining vertex set as $m_t = |V_t|$. For each active node $i \in V_t$, we construct a raw feature vector $o_{i,t}$ containing its spatial coordinates and instance-specific constraint features (e.g., customer demand, time windows), with instance-level features (e.g., the duration limit) and the state of the current route (e.g., remaining capacity) carried by the depot node and an indicator marking the vehicle's current location (see Appendix D). These feature vectors are stacked to form the state observation matrix $O_t \in \mathbb{R}^{m_t \times d_{\text{in}}}$, which is passed through a transformer encoder to produce hidden node representations $h \in \mathbb{R}^{m_t \times d}$, where d is the hidden dimension.

To compute the value estimate from these embeddings, we compare three architectural approaches. The first is a standard **mean pooling baseline**, which averages over the remaining node embeddings and applies a linear decoder:

$$\bar{h}_t = \frac{1}{m_t} \sum_{i \in V_t} h_i, \qquad V_{\text{mean}}(s_t) = w^{\top} \bar{h}_t + b. \quad (5)$$

The second and third approaches are variants of our novel structured parameterizations. To maintain consistency with the global problem definition, we index the hidden representations h_i by their original node identities $i \in V_t$. The logit of an edge (i, j), $i, j \in V_t$, is then computed as a scaled dot product of the respective node embeddings, and the value as the distance-weighted sum of the resulting edge probabilities p_{ij}:

$$\tilde{\ell}_{ij} = \frac{(W_Q h_i)^{\top} (W_K h_j)}{\sqrt{d}}, \qquad V_{\text{structured}}(s_t) = -\sum_{(i,j) \in U(s_t)} d_{ij}\, p_{ij}, \quad (6)$$

with projection matrices $W_Q, W_K \in \mathbb{R}^{d\times d}$. We then introduce two variants for mapping these logits to edge probabilities. The first one is a **generic sigmoid** variant. It applies an independent sigmoid function to each logit, $p_{ij} = \sigma(\tilde{\ell}_{ij})$, and serves as a universally applicable baseline for any combinatorial problem mapped to our MDP framework. The second variant is a **problem-specific softmax**, which explicitly exploits the structural properties of routing constraints. Because each customer node must ultimately be visited exactly once, it must possess exactly one incoming and one outgoing edge in a complete tour. Consequently, the total selection probability of all outgoing edges for any given customer must sum to 1. We enforce this constraint by normalizing the masked logits ℓ_{ij}, which equal $\tilde{\ell}_{ij}$ for undecided edges $(i, j) \in U(s_t)$ with $i \neq j$ and $-\infty$ otherwise, with a softmax operation over the outgoing edges of each customer, while retaining an independent sigmoid for the depot node (which can support multiple outgoing routes):

$$p_{0j} = \sigma(\tilde{\ell}_{0j}), \qquad p_{ij} = \frac{\exp(\ell_{ij})}{\sum_{k\in V_t} \exp(\ell_{ik})}, \quad \text{for } i \neq 0. \tag{7}$$

3.3 Training

We train our policy with the proximal policy optimization (PPO) algorithm (Schulman et al., 2017), using the clipped surrogate objective with an entropy bonus (Appendix E.2). The advantages are computed via generalized advantage estimation (Schulman et al., 2016). For additional stability, we parameterize actor and critic with separate transformer networks and leave the exploration of shared parameters for future work. See Appendix E.1 for details. For data collection, we run full episodes of the MDP with P parallel environments, storing all states, actions and estimated advantage values in a buffer, and subsequently update the policy and value networks with one epoch of mini-batch gradient descent over the buffer. Further details are given in Appendix E. Besides saving computation, auto-stepping also affects the optimization: states with a forced action have zero policy gradient (Appendix B), so without auto-stepping they would shrink the effective batch size of an update and make it vary between batches, most strongly early in training (Table 3). Auto-stepping removes this source of variance.

3.4 Value-Guided Truncated Rollout

The structured value functions are trained to support policy optimization, but an accurate estimate of V^π is also useful at inference time to look ahead and re-rank candidate actions. The classical *rollout algorithm* (Bertsekas et al., 1997) scores each candidate action $a \in \mathcal{A}(s_t)$ by tentatively committing to it and simulating a fixed *base policy* to termination, and then commits to the best-scoring candidate. *Simulation-Guided Beam Search* (SGBS) (Choo et al., 2022) applies this to neural routing solvers with the greedy policy $\pi_\theta^{\text{greedy}}(s_t) := \arg\max_{a\in\mathcal{A}(s_t)} \pi_\theta(a \mid s_t)$ as the base policy, restricts the candidates to the top-k actions of π_θ, and keeps a beam of b real trajectories instead of a single one. Simulating to termination makes a real decision step cost $O(b\,k\,(T-t))$ policy evaluations and the full algorithm $O(b\,k\,T^2)$, which is problematic since T grows with the problem size. We therefore truncate the simulation at $h \ll T - t$ steps and bootstrap the remainder with $\hat{V}_\phi^\pi$, reducing the cost to $O(b\,k\,h)$ per step and $O(b\,k\,T\,h)$ overall (Algorithm 1).

4 Experiments

4.1 Datasets, Metrics, Baselines and Hardware

We train and evaluate on MTVRP instances with 50 and 100 customers (MTVRP-50 and MTVRP-100), generating training instances on the fly and using the held-out validation and test sets of Berto et al. (2025b), which contain 1,000 instances per variant. We report the average solution cost, the average runtime per instance, and the relative gap $100\,(\hat{x} - x^*)/|x^*|$ between the average cost $\hat{x}$ of a method and the average cost x^* of the reference solver PyVRP (Wouda et al., 2024), a state-of-the-art open-source metaheuristic for broad classes of vehicle routing problems. We compare against the neural solvers MTPOMO (Liu et al., 2024), MVMoE (Zhou et al., 2024), RF-TE (Berto et al., 2025b), CaDA (Li et al., 2025), CCL (Gui et al., 2026), CCL_{ReLD}, i.e. CCL with the decoder of Huang et al. (2025), and $\text{MoSES}_{\text{CaDA}}$ (Pan et al., 2025). For MTPOMO and MVMoE we use the checkpoints retrained by Berto et al. (2025b), since the original models were trained on only five and

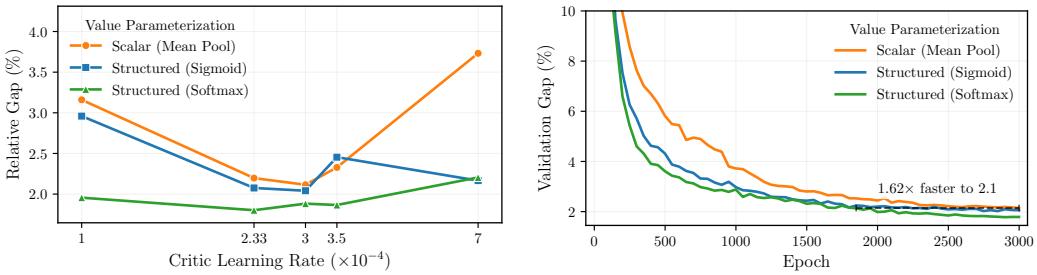


Figure 1: Value-function parameterizations on MTVRP-50 under greedy decoding. Left: final gap against the critic learning rate. Right: validation curves of the best run per parameterization.

six of the variants, respectively, and for the others the publicly available code and checkpoints. Each baseline uses its strongest decoding: multi-start with one start per customer and $\times 8$ augmentation. For comparability, we re-run all baselines on our own hardware: all neural models on one 40GB NVIDIA A100 GPU, and PyVRP on an Intel Xeon Platinum 8468 CPU with a time limit of n seconds for n customers.

4.2 Results

In our main experiments, we compare the different value function parameterizations introduced in Section 3.2. Table 1 shows the average relative gap across the three different parameterizations, for both problem sizes and under different decoding configurations. We can see that the structured parameterizations outperform the mean-pool baseline, with the problem-specific softmax variant performing best. Figure 1 shows the sensitivity of the final model performance to the critic learning rate, as well as the validation curves for the best model of each parameterization variant. We can see that the structured parameterizations are more robust to changes in the critic learning rate, and that they converge faster and to better solutions than the mean-pool baseline, although they only change the output layer and add no hyperparameters.

Table 1: Mean gap (%) over the 16 MTVRP variants per value-function parameterization. Brackets: fraction of the mean-pool baseline's remaining gap to PyVRP that the structured variants close.

	MTVRP-50			MTVRP-100		
decoding	mean	sigmoid	softmax	mean	sigmoid	softmax
greedy	2.138	2.065 (−3.4%)	**1.828** (−14.5%)	2.755	2.681 (−2.7%)	**2.372** (−13.9%)
greedy $\times 8$ aug.	1.194	1.183 (−0.9%)	**1.094** (−8.4%)	1.727	1.697 (−1.7%)	**1.516** (−12.3%)
multistart (n)	0.807	0.790 (−2.1%)	**0.653** (−19.0%)	1.399	1.362 (−2.6%)	**1.156** (−17.4%)
multistart (n) $\times 8$ aug.	0.461	0.459 (−0.4%)	**0.401** (−12.9%)	0.935	0.918 (−1.8%)	**0.786** (−16.0%)

We now measure how often the auto-step mechanism is invoked in practice, and thus how much compute it saves, since this depends on the specific problem class and instance distribution. In our vehicle routing case, for example, it mainly activates at the end of a route when the vehicle has to return to the depot. Table 3 shows the average fraction of steps in an episode that are auto-stepped, for both an untrained and a fully trained policy, simulating the early and late stages of training. We can see that the effect is strongest early in training, with up to 20.9% of states being auto-stepped, and that this fraction drops to 5.7–7.3% for the converged policy, since the untrained policy acts randomly and produces inefficient routes that must return to the depot more often. Similarly, we can also see that the effect is stronger for the more constrained time-windowed instances, which require more routes to serve all customers. Finally, we evaluate the learned value function at test time with the truncated rollout procedure of Section 3.4. Figure 2 plots the relative gap against the average decoding time per instance on MTVRP-100 for standard decoding methods and truncated rollouts with various settings. Since the rollout is sequential and evaluates both the policy and the value function, it lags behind at very small time budgets ($<$ 1s per instance), but with increasing budgets it finds better solutions than the standard decoding methods. The untruncated rollout ($h = \infty$) is

Table 2: Relative gap (%) and single-instance latency (seconds, batch size 1) on the 16 MTVRP variants, all measured by us on one A100. Decoding: *ms* n = multi-start with one start per customer, $\times 8$ = dihedral augmentation, *ro* $k \times b$ = truncated rollout with k candidates and beam width b at depth h=1. The equivalent table with the direct cost values is Table 7.

	$n = 50$		$n = 100$		$n = 50$		$n = 100$		$n = 50$		$n = 100$		$n = 50$		$n = 100$	
Solver	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
	CVRP				VRPTW				OVRP				VRPL			
MTPOMO (ms $n \times 8$)	1.40	0.12	2.35	0.23	2.32	0.13	3.88	0.25	3.19	0.12	5.01	0.23	1.72	0.12	2.71	0.23
MVMoE (ms $n \times 8$)	1.24	0.17	2.04	0.34	2.29	0.19	3.80	0.37	2.96	0.18	4.68	0.35	1.50	0.17	2.39	0.34
RF-TE (ms $n \times 8$)	1.28	0.12	1.86	0.23	2.03	0.13	3.18	0.25	2.66	0.12	4.09	0.24	1.49	0.12	2.08	0.23
CaDA (ms $n \times 8$)	1.18	0.12	1.95	0.22	1.49	0.13	2.54	0.24	2.46	0.12	4.10	0.23	1.31	0.12	2.11	0.23
CCL (ms $n \times 8$)	0.98	0.21	1.70	0.37	0.92	0.23	2.34	0.40	1.94	0.22	3.69	0.39	1.11	0.21	1.86	0.37
CCL_{ReLD} (ms $n \times 8$)	0.87	0.23	1.46	0.39	0.85	0.25	1.83	0.43	1.55	0.23	3.04	0.40	1.00	0.23	1.61	0.39
$\text{MoSES}_{\text{CaDA}}$ (ms $n \times 8$)	0.81	0.29	1.68	0.45	1.28	0.31	2.37	0.49	1.68	0.29	3.66	0.46	0.97	0.29	1.90	0.46
Ours (greedy)	1.85	0.23	1.99	0.37	1.55	0.25	2.07	0.39	2.42	0.23	3.16	0.36	2.30	0.23	2.33	0.36
Ours (greedy $\times 8$)	1.17	0.24	1.28	0.44	0.86	0.33	1.29	0.52	1.47	0.24	2.13	0.39	1.36	0.25	1.47	0.43
Ours (ms n)	0.57	0.38	0.83	1.00	0.53	0.50	1.01	1.16	0.94	0.31	1.71	0.93	0.68	0.39	0.99	1.02
Ours (ms $n \times 8$)	0.34	0.75	0.54	4.09	0.30	0.84	0.66	4.28	0.58	0.72	1.22	4.19	0.41	0.75	0.64	4.09
Ours (ro 4×64)	0.40	0.60	0.66	1.77	0.57	0.72	1.19	2.08	0.39	0.65	0.90	1.91	0.53	0.59	0.83	2.41
Ours (ro $4 \times 64 \times 8$)	**0.17**	2.68	**0.26**	11.5	**0.21**	2.91	**0.56**	12.1	**0.18**	2.84	**0.46**	11.9	**0.24**	2.68	**0.37**	11.4
	VRPB				OVRPTW				VRPBL				VRPBLTW			
MTPOMO (ms $n \times 8$)	3.52	0.12	5.01	0.23	1.46	0.14	2.89	0.26	4.77	0.12	6.42	0.23	2.10	0.14	3.64	0.25
MVMoE (ms $n \times 8$)	3.22	0.17	4.60	0.35	1.48	0.20	2.88	0.40	4.42	0.18	5.95	0.35	2.07	0.20	3.62	0.39
RF-TE (ms $n \times 8$)	2.97	0.12	4.03	0.23	1.32	0.13	2.34	0.26	3.82	0.12	5.18	0.23	1.85	0.13	2.94	0.26
CaDA (ms $n \times 8$)	2.79	0.12	4.13	0.22	0.94	0.14	1.75	0.25	3.49	0.12	5.21	0.22	1.35	0.13	2.37	0.25
CCL (ms $n \times 8$)	2.30	0.20	3.67	0.37	0.53	0.24	1.45	0.42	2.89	0.21	4.43	0.37	0.93	0.23	2.17	0.41
CCL_{ReLD} (ms $n \times 8$)	1.90	0.23	3.08	0.39	0.48	0.26	1.10	0.43	2.47	0.23	3.71	0.39	0.85	0.25	1.72	0.43
$\text{MoSES}_{\text{CaDA}}$ (ms $n \times 8$)	2.05	0.29	3.75	0.46	0.88	0.31	1.67	0.50	2.97	0.29	4.80	0.46	1.29	0.32	2.32	0.49
Ours (greedy)	2.67	0.23	3.25	0.36	0.88	0.27	1.43	0.41	3.94	0.23	4.44	0.36	1.52	0.26	2.07	0.41
Ours (greedy $\times 8$)	1.79	0.22	2.27	0.36	0.49	0.34	0.82	0.55	2.46	0.27	2.88	0.39	0.92	0.34	1.29	0.58
Ours (ms n)	1.06	0.27	1.68	0.78	0.29	0.52	0.62	1.18	1.56	0.42	2.26	0.96	0.54	0.51	0.98	1.19
Ours (ms $n \times 8$)	0.72	0.62	1.21	3.84	0.15	0.89	0.38	4.48	1.09	0.79	1.62	4.06	0.29	0.86	0.64	4.32
Ours (ro 4×64)	0.50	0.54	1.07	2.25	0.36	0.75	0.78	2.13	1.07	0.59	1.74	2.37	0.65	0.73	1.17	2.97
Ours (ro $4 \times 64 \times 8$)	**0.26**	2.64	**0.60**	11.5	**0.12**	3.07	**0.36**	12.7	**0.54**	2.71	**0.99**	11.5	**0.24**	2.92	**0.55**	12.2
	VRPBTW				VRPLTW				OVRPB				OVRPBL			
MTPOMO (ms $n \times 8$)	1.85	0.13	3.25	0.25	2.80	0.13	4.42	0.25	2.99	0.13	5.27	0.23	3.04	0.12	5.32	0.23
MVMoE (ms $n \times 8$)	1.83	0.19	3.27	0.39	2.72	0.19	4.32	0.37	2.75	0.18	4.91	0.36	2.84	0.18	4.96	0.35
RF-TE (ms $n \times 8$)	1.63	0.13	2.60	0.26	2.42	0.13	3.64	0.25	2.49	0.12	4.27	0.24	2.51	0.12	4.31	0.24
CaDA (ms $n \times 8$)	1.08	0.13	1.98	0.24	1.85	0.13	3.05	0.24	2.15	0.13	4.15	0.23	2.16	0.13	4.16	0.23
CCL (ms $n \times 8$)	0.70	0.23	1.79	0.41	1.32	0.23	2.68	0.40	1.56	0.22	3.48	0.38	1.55	0.22	3.47	0.38
CCL_{ReLD} (ms $n \times 8$)	0.62	0.25	1.35	0.44	1.17	0.25	2.23	0.42	1.32	0.24	2.79	0.39	1.31	0.24	2.77	0.40
$\text{MoSES}_{\text{CaDA}}$ (ms $n \times 8$)	0.98	0.31	1.93	0.49	1.73	0.31	2.87	0.49	1.80	0.29	3.79	0.47	1.82	0.29	3.77	0.47
Ours (greedy)	1.18	0.26	1.66	0.40	2.02	0.25	2.57	0.39	2.04	0.23	3.04	0.36	2.04	0.34	3.04	0.58
Ours (greedy $\times 8$)	0.64	0.33	1.01	0.58	1.16	0.33	1.63	0.53	1.23	0.23	1.97	0.36	1.23	0.25	1.97	0.36
Ours (ms n)	0.36	0.52	0.74	1.18	0.75	0.50	1.27	1.16	0.75	0.25	1.56	0.75	0.76	0.31	1.57	0.76
Ours (ms $n \times 8$)	0.19	0.85	**0.47**	4.30	0.45	0.84	0.86	4.27	0.46	0.62	1.10	3.86	0.47	0.70	1.10	3.91
Ours (ro 4×64)	0.50	0.73	1.04	2.72	0.64	0.72	1.30	2.90	0.33	0.56	0.90	2.30	0.37	0.61	0.91	1.77
Ours (ro $4 \times 64 \times 8$)	**0.19**	2.92	0.48	12.2	**0.25**	2.89	**0.65**	12.1	**0.16**	2.76	**0.48**	11.9	**0.18**	2.84	**0.49**	11.9
	OVRPBLTW				OVRPBTW				OVRPL				OVRPLTW			
MTPOMO (ms $n \times 8$)	1.22	0.14	2.47	0.26	1.20	0.14	2.46	0.26	3.21	0.12	5.05	0.23	1.49	0.14	2.90	0.26
MVMoE (ms $n \times 8$)	1.27	0.21	2.48	0.39	1.24	0.21	2.48	0.40	3.03	0.18	4.74	0.35	1.50	0.20	2.89	0.39
RF-TE (ms $n \times 8$)	1.12	0.14	2.01	0.27	1.12	0.14	2.01	0.27	2.68	0.12	4.09	0.24	1.32	0.13	2.34	0.26
CaDA (ms $n \times 8$)	0.74	0.14	1.42	0.26	0.74	0.14	1.42	0.25	2.46	0.12	4.12	0.23	0.95	0.14	1.74	0.25
CCL (ms $n \times 8$)	0.40	0.25	1.22	0.43	0.41	0.25	1.24	0.43	1.93	0.22	3.63	0.38	0.53	0.24	1.50	0.42
CCL_{ReLD} (ms $n \times 8$)	0.38	0.27	0.87	0.45	0.37	0.27	0.89	0.45	1.55	0.23	3.01	0.40	0.48	0.26	1.08	0.44
$\text{MoSES}_{\text{CaDA}}$ (ms $n \times 8$)	0.71	0.33	1.41	0.51	0.70	0.32	1.42	0.51	1.67	0.30	3.67	0.47	0.88	0.32	1.67	0.50
Ours (greedy)	0.74	0.31	1.15	0.47	0.75	0.28	1.15	0.42	2.44	0.32	3.21	0.50	0.90	0.29	1.42	0.46
Ours (greedy $\times 8$)	0.37	0.37	0.65	0.59	0.37	0.37	0.64	0.57	1.48	0.23	2.12	0.39	0.50	0.34	0.83	0.54
Ours (ms n)	0.22	0.53	0.47	1.21	0.22	0.53	0.46	1.21	0.94	0.31	1.71	0.94	0.29	0.52	0.63	1.18
Ours (ms $n \times 8$)	**0.10**	0.91	**0.27**	4.51	0.11	0.91	**0.26**	4.53	0.59	0.72	1.23	4.20	0.15	0.89	0.37	4.47
Ours (ro 4×64)	0.34	0.76	0.75	3.10	0.35	0.76	0.74	3.01	0.40	0.64	0.92	2.55	0.36	0.75	0.79	3.04
Ours (ro $4 \times 64 \times 8$)	0.11	3.11	0.32	12.8	**0.10**	3.11	0.32	12.8	**0.19**	2.83	**0.48**	11.9	**0.12**	3.07	**0.35**	12.6

Table 3: Fraction of auto-stepped steps per episode for the randomly initialized (*untrained*) and final (*trained*) policy, over all 16 variants (all), the 8 with time windows (TW) and the 8 without (no TW).

	MTVRP-50			MTVRP-100		
policy	all	TW	no TW	all	TW	no TW
untrained	17.7%	25.5%	6.4%	20.9%	30.7%	9.6%
trained	7.3%	11.0%	3.2%	5.7%	8.6%	2.5%

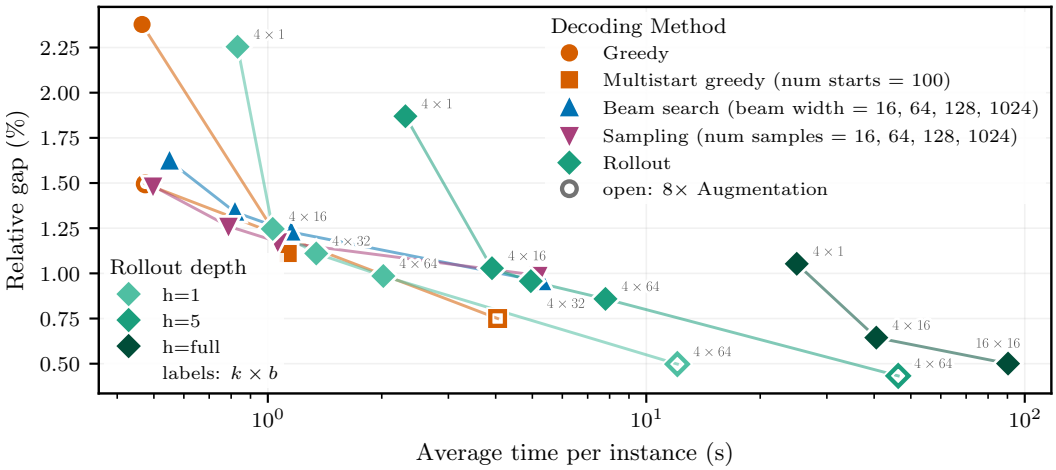


Figure 2: Relative gap against average decoding time per instance on MTVRP-100, averaged over the 16 variants on the first 16 test instances each, and hence not directly comparable to Table 2. Per-variant results are in Figure 4.

strong even at small k and b, but takes at least 25s per instance. Truncation provides a better trade-off, and early truncation at $h = 1$ yields a better Pareto front than $h = 5$ across most of the time range. Table 2 compares our model under several decodings, including the truncated rollout with $h = 1$, against the baselines. Under the multi-start decoding with $\times 8$ augmentation used for all baselines, our model outperforms every baseline on all 16 variants at both problem sizes, and on average (Table 6) reduces the gap to PyVRP by more than 60% relative to the second-best method. Even without augmentation, our multi-start decoding outperforms all baselines with augmentation on average.

5 Conclusion

We revisited value estimation in NCO and proposed two novel structured value function parameterizations for binary programs with linear objectives, whose bounded, cost-weighted outputs provide a strong inductive bias and improve performance, convergence speed and robustness to the critic learning rate over a generic scalar critic. Our final model outperforms all compared construction methods on the 16 MTVRP variants, reducing the average gap to PyVRP by more than 60% relative to the second-best model. We also introduced auto-stepping, removing 17.7–20.9% of processed states early in training and 5.7–7.3% for the converged policy, and used the critic to truncate rollout search at test time, accelerating it at a small loss in quality. Future work could adapt the critic to search, as it is trained only for policy optimization, and extend the framework to quadratic objectives, which require pairwise co-inclusion probabilities, and to objectives with min or max operators.

AI USE STATEMENT

In this work, we used generative AI tools for assistance and feedback in the writing process, as an additional tool for literature search, for producing code in order to plot figures or format tables from the raw experimental data and to implement parts of the proposed method. We have not used generative AI tools for the initial ideation process, for finding and refining hypotheses, for the development of the experimental protocol or translation. The generation or manipulation of synthetic data or assistance for mathematical claims/proofs is not applicable to this work. We have reviewed all AI-assisted work by reading and testing generated code, cross-checking produced tables and figures against the original data, and performing additional literature search through traditional means. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

For reproducibility, detailed descriptions of all architectural, training and evaluation details and hyperparameters are provided in the main text in Section 3 and Appendices E.1 and E.2. We provide relevant pseudocode in Algorithm 1. The actual source code along with the trained models and exact commands and config files for reproducing the models and results will be made publicly available upon acceptance of the paper. Our used test datasets are already publicly available through the related work (Berto et al., 2025b). The utilized hardware is described in Section 4.1.

REFERENCES

Puneet S. Bagga and Arthur Delarue. Solving the Quadratic Assignment Problem using Deep Reinforcement Learning, October 2023. URL http://arxiv.org/abs/2310.01604. arXiv:2310.01604.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Workshop Track Proceedings*, 2017. URL https://arxiv.org/abs/1611.09940.

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, and Jinkyoo Park. RL4CO: an extensive reinforcement learning for combinatorial optimization benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2025a. doi: 10.1145/3711896.3737433.

Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. RouteFinder: Towards Foundation Models for Vehicle Routing Problems. *Transactions on Machine Learning Research*, June 2025b. ISSN 2835-8856.

Dimitri P. Bertsekas. *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Athena Scientific, Belmont, MA, 2020.

Dimitri P. Bertsekas, John N. Tsitsiklis, and Cynara Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997. doi: 10.1023/A:1009635226865.

Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. Simulation-guided beam search for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 35, pp. 8760–8772, 2022.

Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation quotienting for efficient neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://openreview.net/forum?id=BRqlkTDvvm.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 7474–7482. AAAI Press, 2021. doi: 10.1609/AAAI.V35I8.16916. URL https://doi.org/10.1609/aaai.v35i8.16916.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256. PMLR, 2010. URL https://proceedings.mlr.press/v9/glorot10a.html.

Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant RL populations for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 48485–48509, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/97b983c974551153d20ddfabb62a5203-Abstract-Conference.html.

Shuangchun Gui, Suyu Liu, Xuehe Wang, and Zhiguang Cao. Chain-of-Context Learning: Dynamic Constraint Understanding for Multi-Task VRPs. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*. OpenReview.net, 2026. URL https://openreview.net/forum?id=AhE6aSlz5g.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034. IEEE, 2015. doi: 10.1109/ICCV.2015.123.

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking Light Decoder-based Solvers for Vehicle Routing Problems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. *CoRR*, abs/1906.01227, 2019. URL http://arxiv.org/abs/1906.01227. arXiv: 1906.01227.

Andrej Karpathy. Nanochat: The best ChatGPT that $100 can buy. https://github.com/karpathy/nanochat, 2025.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning Combinatorial Optimization Algorithms over Graphs. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://openreview.net/forum?id=kHrE2vi5Rvs.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/f231f2107df69eab0a3862d50018a9b2-Abstract.html.

Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 5138–5149, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/29539ed932d32f1c56324cded92c07c2-Abstract.html.

Han Li, Fei Liu, Zhi Zheng, Yu Zhang, and Zhenkun Wang. CaDA: Cross-Problem Routing Solver with Constraint-Aware Dual-Attention. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-Second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025.

Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. 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*, KDD '24, pp. 1898–1908, New York, NY, USA, August 2024. Association for Computing Machinery. ISBN 979-8-4007-0490-1. doi: 10.1145/3637528.3672040.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Qing Luo, Fu Luo, Ke Li, and Zhenkun Wang. Efficient Decoder Scaling Strategy for Neural Routing Solvers, February 2026. URL http://arxiv.org/abs/2603.00430. arXiv:2603.00430 [cs].

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://openreview.net/forum?id=TmLqkYn71gV.

MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement learning for solving the vehicle routing problem. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in neural information processing systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf.

Yuxin Pan, Zhiguang Cao, Chengyang Gu, Liu Liu, Peilin Zhao, Yize Chen, and Fangzhen Lin. Multi-Task Vehicle Routing Solver via Mixture of Specialized Experts under State-Decomposable MDP. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL https://openreview.net/forum?id=ezSyZM6Lp7.

Jonathan Pirnay and Dominik G. Grimm. Self-Improvement for Neural Combinatorial Optimization: Sample Without Replacement, but Improvement. *Transactions on Machine Learning Research*, March 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=agT8ojoH0X.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

David R. So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. Primer: Searching for efficient transformers for language modeling. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://arxiv.org/abs/2109.08668.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://openreview.net/forum?id=JV8Ff0lgVV.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. doi: 10.1007/BF00992696.

Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: A high-performance VRP solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024. doi: 10.1287/ijoc.2023.0055.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):5057–5069, 2022.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced ant systems for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2309.14032.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, volume 32, pp. 12360–12371, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1621–1632, 2020.

Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Chi Xu. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pp. 61804–61824, Vienna, Austria, July 2024. JMLR.org.

A Initial Calibration of the Value Parameterizations

Section 3.2 argued that one of the favorable properties of the structured value function parameterizations is that they are already informative at the start of training. We show this here more concretely, first through analytical arguments and then also empirically.

Standard initialization schemes for neural networks such as Xavier (Glorot & Bengio, 2010) or He initialization (He et al., 2015) typically aim to ensure consistent signal propagation, such that the outputs of the network have approximately zero mean and constant small variance. For the policy network, this means that the initial action logits are close to zero and the initial policy is close to

uniform over its feasible actions. For the structured value function parameterizations, the same holds for the edge logits $\tilde{\ell}_{ij}$, which are also close to zero and uniform across (i, j). For the scalar-based mean-pool value function, this means its initial value estimate is close to zero, independent of the instance and state.

For the sigmoid parameterization, assuming on average zero logits, we get $p_{ij} = \sigma(\tilde{\ell}_{ij}) \approx \frac{1}{2}$ for every undecided edge, thus we expect for the initial value function:

$$V_\sigma^{\pi_0}(s_t) \approx -\frac{1}{2} \sum_{(i,j)\in U(s_t)} d_{ij}, \tag{8}$$

predicting half the total remaining cost mass. This already carries real state and instance information. The fewer edges remain in the state, the smaller the predicted cost, and instances with higher cost coefficients receive larger predictions. With no further knowledge about the constraint structure, predicting half the remaining cost on average appears reasonable. However, in our problems we know that every customer must be served by exactly one outgoing edge, therefore predicting half the remaining cost is an overestimate. This is fixed by the softmax parameterization, for which on average zero logits lead to $p_{ij} \approx \frac{1}{k_i}$, where $k_i = |\{j : (i, j) \in U(s_t)\}|$ is the number of feasible successors of i. Each customer i therefore contributes the mean distance to its feasible successors, $\bar{d}_i = \frac{1}{k_i} \sum_{j:(i,j)\in U(s_t)} d_{ij}$. Since the depot has no such constraint, its row remains a sigmoid at $p_{0j} \approx \frac{1}{2}$. This results in an overall initial value function of

$$V_{\text{softmax}}^{\pi_0}(s_t) \approx -\sum_{i\in V_t\setminus\{0\}} \bar{d}_i - \frac{1}{2} \sum_{j:(0,j)\in U(s_t)} d_{0j}. \tag{9}$$

This, similarly to the sigmoid parameterization, already carries real state and instance information, but also does not have the same overestimation problem, since it approximately considers the actual number of decision variables that will be included in the final solution. Note that the above analysis is only approximate. However, it makes clear that both structured parameterizations align more closely with the true value function than the mean-pool baseline, which is completely uninformative at initialization time. We also show this empirically in Figure 3 (left), where we first collect a small dataset of states by sampling random instances and performing a random episode with a freshly initialized policy, and then score these states both with freshly initialized value functions of all types and by repeatedly rolling out the same untrained policy from each state to produce a Monte-Carlo estimate of the true return-to-go. We can see that the mean-pool baseline produces a roughly constant value for all states regardless of the number of decision variables remaining or the instance cost coefficients, while the structured parameterizations already produce a meaningful ordering of the states. The sigmoid parameterization is off by a scaling factor and generally overestimates the expected cost, but the softmax parameterization is well calibrated out of the box.

Figure 3 (right) additionally displays the value loss curves during training for the three parameterizations for the same runs as in Figure 1 (right). We can clearly see that the faster convergence of the policy performance also correlates with accelerated convergence in the value loss, which is consistent with the improved inductive biases of the structured value function parameterizations, including their initial calibration.

B Forced States Contribute No Policy Gradient

The policy computes $\pi_\theta(a \mid s)$ as a softmax over the logits $\ell_\theta(s)$ of the currently legal actions $\mathcal{A}(s)$ only.[2] If a state has only a single legal action, $\mathcal{A}(s) = \{a\}$, then

$$\pi_\theta(a \mid s) = \frac{\exp \ell_{\theta,a}(s)}{\sum_{b\in\mathcal{A}(s)} \exp \ell_{\theta,b}(s)} = \frac{\exp \ell_{\theta,a}(s)}{\exp \ell_{\theta,a}(s)} = 1, \tag{10}$$

so the policy assigns probability 1 to that action, irrespective of θ. As such, the policy ratio of the PPO objective equation 12 is also constant $\rho(\theta) = 1/1 = 1$, and the policy entropy is $H\big(\pi_\theta(\cdot \mid s)\big) = -1 \cdot \log 1 = 0$. Therefore this state's term in the PPO objective reduces to

$$\min\big(1 \cdot \hat{A},\ \text{clip}(1, 1-\epsilon, 1+\epsilon)\,\hat{A}\big) + \beta \cdot 0 = \hat{A}, \tag{11}$$

[2] In the implementation, illegal actions are actually handled by setting their logits to $-\infty$, which is equivalent but makes batching easier.

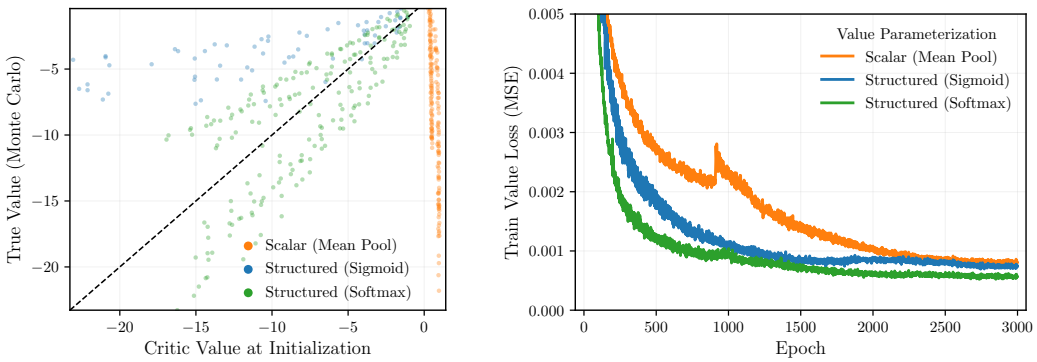


Figure 3: Value-function parameterizations before and during training. Left: predicted value against true return-to-go (estimated via Monte Carlo with 256 samples) for states collected on MTVRP-20 under **freshly initialized and untrained policy and value function**. The softmax variant sits on the diagonal displaying good calibration out of the box (see Equation 9), while the generic sigmoid is off by a scaling factor, consistently overestimating the remaining cost (see Equation 8), however ordering the states approximately correctly and thus also already providing useful information at initialization time. The scalar baseline is roughly constant and therefore uninformative before training starts. Right: value loss of the three parameterizations during training on MTVRP-50. The validation gap curves of these runs are shown in Figure 1 (right). As can be seen, the structured value heads' initial calibration and other properties accelerate learning.

which does not depend on θ, so its policy gradient is zero.

C Truncated Rollout Algorithm

For completeness, we provide the pseudocode of the Value-Guided Truncated Rollout procedure in Algorithm 1.

D The Multi-Task Vehicle Routing Problem

The MTVRP environment, introduced in Berto et al. (2025b), encompasses 16 different vehicle routing problems by adding any combination of 4 additional constraints on top of the standard Capacitated Vehicle Routing Problem (CVRP). The variants add open routes (O), backhaul customers (B), duration limits (L) and time windows (TW).

Base problem (CVRP). An instance consists of a set of customer nodes $C = \{1, \dots, m\}$ and a depot node $\{0\}$. Each node in the entire vertex set $V = C \cup \{0\}$ is associated with 2D-coordinates, sampled uniformly at random from the unit square $[0, 1]^2$. The distance between any two nodes is computed as the Euclidean distance, denoted as d_{ij}. Each customer i carries a demand q_i drawn uniformly from $\{1, \dots, 10\}$. The vehicle has a fixed capacity Q and the number of vehicles is not constrained. A solution is a set of routes, each starting and ending at the depot, that together serve every customer exactly once and whose cumulative demand of served customers never exceeds Q. The objective is total travelled distance.

Open routes (O). Vehicles are not required to return to the depot, or equivalently all edges $(i, 0)$ from a customer to the depot have an associated travel distance of zero: $d_{i0} = 0$.

Backhaul (B). A fraction of customers (by default 20%) are *backhaul* nodes. Their demand q_i is picked up rather than delivered, unlike the regular *linehaul* customers. The vehicle capacity must still be respected, and within a route all *linehaul* customers must be served before any *backhaul* customer.

Algorithm 1 Value-Guided Truncated Rollout. With $h = \infty$ the value function is never queried and the procedure reduces to SGBS (Choo et al., 2022). With $h = \infty$, $b = 1$ and $k = |\mathcal{A}(s)|$ it is the classical rollout algorithm with the greedy base policy.

Require: instance I, policy π_θ, value function $\hat{V}_\phi^\pi$, candidate count k, beam width b, horizon $h \geq 1$

$\mathcal{B} \leftarrow \{(s_0, 0)\}$ ▷ real trajectories: state s, accumulated reward R
while any $(s, R) \in \mathcal{B}$ is not terminal **do**
  $\mathcal{P} \leftarrow \emptyset$ ▷ candidate pool, shared across the beam
  for each $(s, R) \in \mathcal{B}$ **do** ▷ both for loops can be parallelized, the while loop is sequential
    if s is not terminal **then**
      for each $a \in$ top-$k\, \pi_\theta(\cdot \mid s)$ **do**
        $(s', r) \leftarrow \textsc{Step}(s, a)$ ▷ tentatively commit to a
        $\hat{Q}_h(s, a) \leftarrow r + \textsc{Lookahead}(s', h-1)$
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{(s',\ R + r,\ R + \hat{Q}_h(s, a))\}$
      end for
    else
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{(s,\ R,\ R)\}$ ▷ already complete solution, exact scores
    end if
  end for
  $\mathcal{B} \leftarrow$ state and reward of the b entries of $\mathcal{P}$ with the highest score
end while
return solution of the $(s, R) \in \mathcal{B}$ with the highest R

function $\textsc{Lookahead}(s, h)$ ▷ estimate return-to-go from s
  $G \leftarrow 0$, $i \leftarrow 0$
  while $i < h$ **and** s is not terminal **do**
    $(s, r) \leftarrow \textsc{Step}(s, \pi_\theta^{\text{greedy}}(s))$ ▷ follow greedy base policy
    $G \leftarrow G + r,\ i \leftarrow i + 1$
  end while
  return $G + \mathbb{1}[s \text{ not terminal}]\, \hat{V}_\phi^\pi(s)$ ▷ If not terminal, bootstrap return with value function
end function

Duration limit (L). A route's total travelled distance must not exceed a limit L. It is sampled uniformly at random from the interval $[2 \max_{i \in C} d_{0i}, 2\sqrt{2}]$, where $2\sqrt{2}$ is the length of a round trip along the diagonal of the unit square and $2 \max_{i \in C} d_{0i}$ is the instance-specific lower bound that guarantees that every customer can be served and the instance is feasible overall.

Time windows (TW). Each customer i has a time window $[e_i, l_i]$ and a service time s_i, such that $e_i + s_i \leq l_i$. A vehicle arriving before e_i has to wait, and arriving after l_i is infeasible. Here, a unit speed of 1 is assumed, such that travel time between two nodes is equal to the distance d_{ij}.

Numerical tolerance in the feasibility check. We want to note that when recomputing the PyVRP solutions on our own hardware and passing them through the environment for cost and feasibility results, we observed that some solutions that PyVRP reports are infeasible according to the environment implementation as the ML models observe them. The root cause is partially that PyVRP processes the instance through integer values and thus the original float values, like distances, are scaled and rounded. To compensate, we increased the scaling factor which reduced the number of infeasible solutions, but the problem partially also stems from the fact that the implementation of the environment as proposed in Berto et al. (2025b) does not use any slack in the feasibility check. In particular, we observed that some solutions that should be feasible are rejected by the environment because of small numerical errors, for example when adding up the demands of customers on a route, when the vehicle capacity is fully utilized[3]. Some earlier implementations absorb such rounding errors with a small slack, typically in the action mask itself: POMO (Kwon et al., 2020) and

[3]Routes that use the entire vehicle capacity exactly can occur, since the original demands are integers between 1 and 10, with a fixed vehicle capacity depending on problem size. Then, the environment scales the demands by vehicle capacity and implicitly assumes a vehicle capacity of 1. When it later computes the cumulative route demand, a small numerical error without any slack can reject a feasible route.

Table 4: Different RouteFinder (Berto et al., 2025b) models evaluated with our feasibility slack $\varepsilon = 10^{-4}$ environment modification used throughout this paper and without it (none), as in the original implementation. Each cell is the unweighted mean over the 16 seen MTVRP variants of the per-variant gap (%) to PyVRP. Δ denotes the change from switching the tolerance on (the ε column minus the none column).

		MTVRP50 Gap (%)			MTVRP100 Gap (%)		
Model	Decoding	none	ε	Δ	none	ε	Δ
RF-POMO	greedy	6.44	6.39	-0.05	7.51	7.49	-0.02
	greedy $\times 8$	3.61	3.58	-0.03	5.14	5.11	-0.03
	ms n	3.68	3.65	-0.03	5.08	5.06	-0.02
	ms n $\times 8$	2.15	2.12	-0.03	3.65	3.63	-0.02
RF-MoE	greedy	6.53	6.49	-0.04	7.28	7.24	-0.04
	greedy $\times 8$	3.66	3.63	-0.03	4.91	4.89	-0.02
	ms n	3.75	3.72	-0.03	4.89	4.86	-0.03
	ms n $\times 8$	2.17	2.14	-0.03	3.48	3.45	-0.03
RF-TE	greedy	5.68	5.63	-0.05	6.53	6.50	-0.03
	greedy $\times 8$	3.53	3.49	-0.04	4.56	4.53	-0.03
	ms n	3.23	3.20	-0.03	4.35	4.32	-0.03
	ms n $\times 8$	2.07	2.04	-0.03	3.21	3.19	-0.02

MVMoE (Zhou et al., 2024) allow 10^{-5} on capacity, time-window and duration constraints, MT-POMO (Liu et al., 2024) 10^{-6} on capacity, and Kool et al. (2019) 10^{-5} when validating solutions but not during construction, while Drakulic et al. (2023) avoid the issue for the CVRP by keeping demands integer. The environment of Berto et al. (2025b) uses no slack. In order not to give PyVRP an unfair advantage, we add a tolerance of $\varepsilon = 10^{-4}$ to the environment implementation and apply it also to all baselines we report. The value was selected as the minimum value such that all solutions from PyVRP were accepted by the environment. With the current sampling of the customers, there is no restriction on how close two customers can be, and as such their distance may be smaller than the tolerance ε. Unfortunately, without a change in the instance generation procedure, there is no easy way of reconciling both issues. We therefore believe the best current way to handle this is to introduce the tolerance as we have done to ensure PyVRP solutions are accepted and run all ML models through the same environment with the same tolerance, to ensure exact comparability. However, given the inconsistent handling of this issue across the related work, future work may consider this problem and propose more robust implementations to handle such issues. Finally, note that while we believe the issues discussed here are important to consider, they affect the results only in small ways. The gaps and costs reported for all methods only change slightly. We present some additional experiments for this in Table 4 displaying the relative gaps of different RouteFinder (Berto et al., 2025b) models when run with and without the tolerance in the environment's constraint checks. As expected, since some numerical errors do not cause feasible routes to be rejected anymore, the additional tolerance can only improve the results, but the effect overall is very small compared to the increased performance of our models compared to the baselines.

E Training, Architecture Details and Hyperparameters

In the following we describe the training details and hyperparameters used for the experiments in this paper. Section E.1 describes in more detail the actual architecture used for our models, while Section E.2 discusses the training procedure and Table 5 documents the hyperparameters used for the final models that produced the results in Table 2.

E.1 Model

Actor and critic are separate transformer networks of identical structure besides their output heads and with no shared parameters. For the specific architecture we follow the recommendations of Karpathy (2025) and use pre-norm residual blocks with RMSNorm (Zhang & Sennrich, 2019) and an additional normalization layer after the last block. The feedforward blocks use a standard 2-layer

MLP at $4\times$ the model dimension with a squared-ReLU nonlinearity (So et al., 2021). We also use the width-to-depth ratio of Karpathy (2025), setting the model dimension to 64 times the depth. With $L = 6$ layers, this gives a model dimension of 384 and a feed-forward dimension of 1536. There is no positional encoding, since a state is a set of nodes rather than a sequence.

We follow these recommendations, since they represent common best practices for transformer architectures in vision and language modeling. However, it is possible that they are suboptimal for our tasks, since they are algorithmic in nature and not as knowledge intensive as natural language or vision tasks. Work on the TSP under supervised learning suggests that less wide architectures are more parameter efficient (Luo et al., 2026), however impacts on latency and training stability under reinforcement learning and with mixed precision are not well understood. Therefore we leave further architecture search for future work and stick to these common recommendations.

The actor's output head is a node-wise linear projection to a single logit per node, which is then masked and normalized with a softmax to produce the action distribution over the feasible actions. We follow Bello et al. (2017) and clip the logits with $C\tanh(\cdot)$ to the range $[-C, C]$ before the softmax, choosing $C = 20$. The critic's output head is our proposed structured value function, for which we use the softmax variant in our final experiments.

E.2 Training Setup

We optimize the policy with the clipped PPO objective (Schulman et al., 2017) and an entropy bonus:

$$\mathcal{L}(\theta) = \mathbb{E}_t\Big[\min\big(\rho_t(\theta)\hat{A}_t,\ \mathrm{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\big) + \beta\, H\big(\pi_\theta(\cdot \mid s_t)\big)\Big],\ \rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{12}$$

where ρ_t is the importance ratio against the behavior policy that collected the batch, ϵ the clip range, and β the coefficient of the entropy bonus H. The advantages $\hat{A}_t$ are computed via generalized advantage estimates (Schulman et al., 2016).

Training instances are sampled on the fly, uniformly at random over the 16 MTVRP variants. We follow Berto et al. (2025b) and make an arbitrary separation into epochs, where every epoch consists of 10240 instances. At each iteration, we sample a batch of P parallel environments, each with a single instance, and run the current policy by sampling an action from the policy distribution at each step until the episode terminates. The resulting transitions are stored in a buffer, which is then used to optimize the actor and critic networks. We use a mini-batch size of 4096 transitions for optimization, and perform a single pass over the buffer per training step. During training we use PyTorch automatic mixed precision with bf16 for efficiency, however the inference results are all obtained in fp32 precision to ensure comparability with the baselines. For optimization we use Muon (Jordan et al., 2024), with AdamW (Loshchilov & Hutter, 2019) as the fallback optimizer for the parameters Muon does not apply to and a cosine learning rate schedule (Loshchilov & Hutter, 2017) with linear warmup. The $n{=}100$ model is initialized from the $n{=}50$ checkpoint rather than trained from scratch.

F Additional Results

Table 2 reports gaps per variant. Table 7 gives the corresponding objective values, shown here because they do not fit into the same table. Additionally, we present aggregate results across all variants in Table 6. Finally, we show the per-variant breakdown of the decoding results in Figure 4.

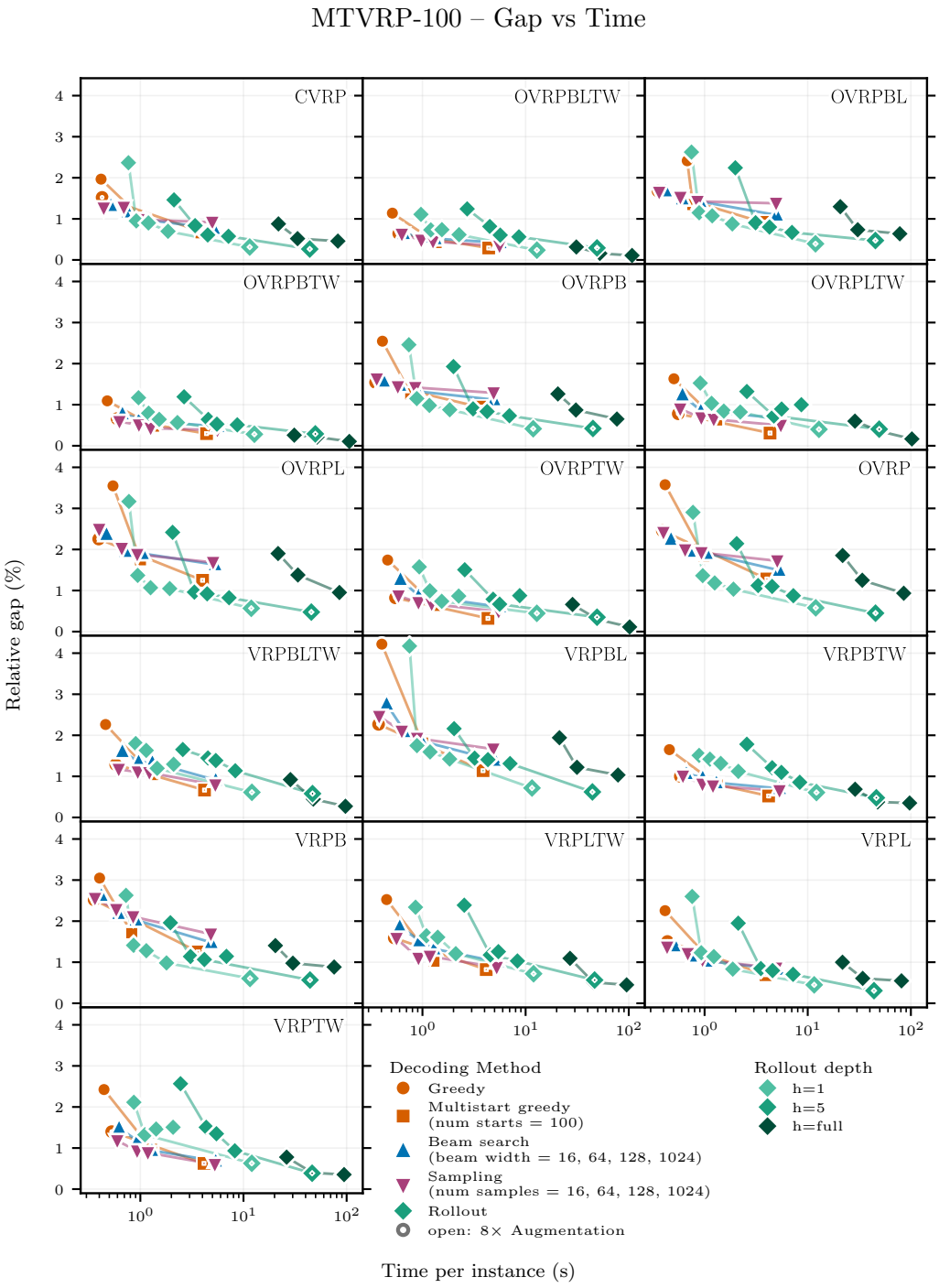


Figure 4: Relative gap against decoding time for each of the 16 MTVRP variants individually at n=100. Find aggregate results in Figure 2.

Table 5: Relevant hyperparameters for training the models. When relevant, the settings are meant per device in a multi-GPU setup, trained with distributed data parallel (DDP). As the n=100 model is a fine-tuned version of the n=50 model, some hyperparameters differ, which are denoted in this table.

Architecture (actor and critic are separate networks)	
Layers / model dim / heads / FF dim	6 / 384 / 6 / 1536
Normalization	RMSNorm
Feed-forward nonlinearity	squared ReLU
Logit clipping (only for actor)	$C\tanh(x)$, $C = 20$
Value Parameterization	Structured softmax variant
Initialization (MTVRP50 / MTVRP100)	Random / MTVRP50 checkpoint
PPO	
Parallel environments	1024
Minibatch size (number of states per update step)	4096
PPO Epochs (passes over the buffer per step)	1
PPO clip range	0.3
Entropy bonus coefficient	10^{-3}
Advantage normalization	yes
Value-loss clipping	yes
Optimization	
Instances per epoch	10240
Number of epochs (MTVRP50 / MTVRP100)	3000 / 500
Optimizer (actor and critic)	Muon, weight decay 10^{-7}
Fallback optimizer for biases and scalars	AdamW, weight decay 10^{-7}
Actor LR MTVRP50 (Muon / AdamW)	$3 \cdot 10^{-4}$ / 10^{-4}
Critic LR MTVRP50 (Muon / AdamW)	$2.33 \cdot 10^{-4}$ / 10^{-4}
Actor LR MTVRP100 (Muon / AdamW)	$3 \cdot 10^{-6}$ / 10^{-6}
Critic LR MTVRP100 (Muon / AdamW)	$3 \cdot 10^{-6}$ / 10^{-6}
LR schedule	cosine annealing to 0, linear warmup from 0
Warmup steps	700 / 500
Gradient-norm clip (actor / critic)	3.0 / 1.0
Number of GPUs (MTVRP50 / MTVRP100)	4 (DDP) / 1
Wall clock (MTVRP50 / MTVRP100)	$\approx<$ 2 days / $\approx$ 15 h

Table 6: MTVRP averages over the 16 seen variants: mean gap (%) to PyVRP and mean single-instance latency (seconds, batch size 1). See Table 2 and Table 7 for the per-variant results.

	$n = 50$		$n = 100$	
Solver	Gap	Time	Gap	Time
PyVRP	*	50.0	*	100
MTPOMO (multi-start n + $\times 8$ aug.)	2.39	0.13	3.94	0.24
MVMoE (multi-start n + $\times 8$ aug.)	2.27	0.19	3.75	0.37
RF-TE (multi-start n + $\times 8$ aug.)	2.04	0.13	3.19	0.25
CaDA (multi-start n + $\times 8$ aug.)	1.70	0.13	2.89	0.24
CCL (multi-start n + $\times 8$ aug.)	1.25	0.23	2.52	0.39
CCL_{ReLD} (multi-start n + $\times 8$ aug.)	1.07	0.24	2.03	0.42
$\text{MoSES}_{\text{CaDA}}$ (multi-start n + $\times 8$ aug.)	1.39	0.30	2.67	0.48
Ours (greedy)	1.83	0.26	2.37	0.41
Ours (greedy + $\times 8$ aug.)	1.09	0.29	1.52	0.47
Ours (multi-start n)	0.65	0.42	1.16	1.04
Ours (multi-start n + $\times 8$ aug.)	0.40	0.79	0.79	4.21
Ours (rollout k=4, b=64, h=1)	0.48	0.67	0.98	2.45
Ours (rollout k=4, b=64, h=1 + $\times 8$ aug.)	**0.20**	2.87	**0.48**	12.1

Table 7: Objective (tour length) and single-instance latency (seconds, batch size 1) on the 16 seen MTVRP variants — the same runs as Table 2, with objectives in place of gaps and the PyVRP reference shown as a row. Decoding: *ms* n = multi-start with one start per customer, $\times 8$ = dihedral augmentation, *ro* $k\times b$ = truncated rollout with k candidates and beam width b at depth h=1.

	$n=50$		$n=100$		$n=50$		$n=100$		$n=50$		$n=100$		$n=50$		$n=100$	
Solver	Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time
	CVRP				VRPTW				OVRP				VRPL			
PyVRP	10.366	50.0	15.561	100	16.037	50.0	25.422	100	6.507	50.0	9.719	100	10.585	50.0	15.717	100
MTPOMO (ms $n\times 8$)	10.511	0.12	15.927	0.23	16.410	0.13	26.408	0.25	6.714	0.12	10.206	0.23	10.767	0.12	16.142	0.23
MVMoE (ms $n\times 8$)	10.494	0.17	15.878	0.34	16.405	0.19	26.387	0.37	6.699	0.18	10.174	0.35	10.743	0.17	16.092	0.34
RF-TE (ms $n\times 8$)	10.498	0.12	15.850	0.23	16.363	0.13	26.230	0.25	6.679	0.12	10.117	0.24	10.742	0.12	16.044	0.23
CaDA (ms $n\times 8$)	10.488	0.12	15.864	0.22	16.276	0.13	26.068	0.24	6.667	0.12	10.118	0.23	10.724	0.12	16.049	0.23
CCL (ms $n\times 8$)	10.467	0.21	15.825	0.37	16.185	0.23	26.017	0.40	6.633	0.22	10.078	0.39	10.703	0.21	16.009	0.37
CCL_{ReLD} (ms $n\times 8$)	10.456	0.23	15.788	0.39	16.174	0.25	25.888	0.43	6.607	0.23	10.015	0.40	10.691	0.23	15.970	0.39
$\text{MoSES}_{\text{CaDA}}$ (ms $n\times 8$)	10.449	0.29	15.822	0.45	16.242	0.31	26.025	0.49	6.616	0.29	10.075	0.46	10.687	0.29	16.015	0.46
Ours (greedy)	10.557	0.23	15.870	0.37	16.286	0.25	25.948	0.39	6.664	0.23	10.027	0.36	10.828	0.23	16.082	0.36
Ours (greedy $\times 8$)	10.487	0.24	15.761	0.44	16.175	0.33	25.749	0.52	6.602	0.24	9.926	0.39	10.729	0.25	15.948	0.43
Ours (ms n)	10.424	0.38	15.690	1.00	16.123	0.50	25.678	1.16	6.568	0.31	9.886	0.93	10.656	0.39	15.873	1.02
Ours (ms $n\times 8$)	10.401	0.75	15.644	4.09	16.085	0.84	25.590	4.28	6.544	0.72	9.838	4.19	10.629	0.75	15.818	4.09
Ours (ro 4×64)	10.407	0.60	15.664	1.77	16.128	0.72	25.724	2.08	6.532	0.65	9.807	1.91	10.641	0.59	15.847	2.41
Ours (ro $4\times 64\times 8$)	10.383	2.68	15.601	11.5	16.072	2.91	25.564	12.1	6.518	2.84	9.764	11.9	10.610	2.68	15.774	11.4
	VRPB				OVRPTW				VRPBL				VRPBLTW			
PyVRP	9.686	50.0	14.358	100	10.513	50.0	16.928	100	10.185	50.0	14.757	100	18.599	50.0	29.809	100
MTPOMO (ms $n\times 8$)	10.027	0.12	15.077	0.23	10.666	0.14	17.418	0.26	10.671	0.12	15.705	0.23	18.989	0.14	30.895	0.25
MVMoE (ms $n\times 8$)	9.998	0.17	15.018	0.35	10.668	0.20	17.415	0.40	10.635	0.18	15.635	0.35	18.984	0.20	30.889	0.39
RF-TE (ms $n\times 8$)	9.974	0.12	14.937	0.23	10.652	0.13	17.324	0.26	10.573	0.12	15.521	0.23	18.942	0.13	30.686	0.26
CaDA (ms $n\times 8$)	9.956	0.12	14.951	0.22	10.612	0.14	17.224	0.25	10.540	0.12	15.525	0.22	18.849	0.13	30.516	0.25
CCL (ms $n\times 8$)	9.909	0.20	14.884	0.37	10.568	0.24	17.174	0.42	10.479	0.21	15.411	0.37	18.771	0.23	30.455	0.41
CCL_{ReLD} (ms $n\times 8$)	9.870	0.23	14.800	0.39	10.564	0.26	17.114	0.43	10.436	0.23	15.304	0.39	18.756	0.25	30.321	0.43
$\text{MoSES}_{\text{CaDA}}$ (ms $n\times 8$)	9.884	0.29	14.896	0.46	10.605	0.31	17.210	0.50	10.487	0.29	15.465	0.46	18.839	0.32	30.502	0.49
Ours (greedy)	9.945	0.23	14.824	0.36	10.606	0.27	17.170	0.41	10.586	0.23	15.412	0.36	18.881	0.26	30.425	0.41
Ours (greedy $\times 8$)	9.860	0.22	14.683	0.36	10.565	0.34	17.068	0.55	10.435	0.27	15.182	0.39	18.769	0.34	30.194	0.58
Ours (ms n)	9.789	0.27	14.598	0.78	10.543	0.52	17.033	1.18	10.344	0.42	15.090	0.96	18.700	0.51	30.103	1.19
Ours (ms $n\times 8$)	9.756	0.62	14.531	3.84	10.529	0.89	16.992	4.48	10.296	0.79	14.997	4.06	18.653	0.86	30.000	4.32
Ours (ro 4×64)	9.735	0.54	14.511	2.25	10.550	0.75	17.061	2.13	10.293	0.59	15.013	2.37	18.719	0.73	30.157	2.97
Ours (ro $4\times 64\times 8$)	9.711	2.64	14.444	11.5	10.525	3.07	16.989	12.7	10.239	2.71	14.903	11.5	18.643	2.92	29.972	12.2
	VRPBTW				VRPLTW				OVRPB				OVRPBL			
PyVRP	18.302	50.0	29.471	100	16.363	50.0	25.750	100	6.898	50.0	10.332	100	6.899	50.0	10.332	100
MTPOMO (ms $n\times 8$)	18.640	0.13	30.430	0.25	16.822	0.13	26.888	0.25	7.104	0.13	10.876	0.23	7.109	0.12	10.882	0.23
MVMoE (ms $n\times 8$)	18.637	0.19	30.435	0.39	16.808	0.19	26.861	0.37	7.088	0.18	10.839	0.36	7.095	0.18	10.845	0.35
RF-TE (ms $n\times 8$)	18.600	0.13	30.237	0.26	16.759	0.13	26.688	0.25	7.070	0.12	10.773	0.24	7.072	0.12	10.778	0.24
CaDA (ms $n\times 8$)	18.500	0.13	30.055	0.24	16.666	0.13	26.534	0.24	7.046	0.13	10.760	0.23	7.048	0.13	10.762	0.23
CCL (ms $n\times 8$)	18.429	0.23	29.998	0.41	16.578	0.23	26.439	0.40	7.005	0.22	10.691	0.38	7.006	0.22	10.691	0.38
CCL_{ReLD} (ms $n\times 8$)	18.416	0.25	29.868	0.44	16.555	0.25	26.325	0.42	6.989	0.24	10.620	0.39	6.990	0.24	10.619	0.40
$\text{MoSES}_{\text{CaDA}}$ (ms $n\times 8$)	18.481	0.31	30.041	0.49	16.645	0.31	26.488	0.49	7.022	0.29	10.723	0.47	7.024	0.29	10.722	0.47
Ours (greedy)	18.518	0.26	29.960	0.40	16.693	0.25	26.411	0.39	7.039	0.23	10.646	0.36	7.039	0.34	10.646	0.58
Ours (greedy $\times 8$)	18.419	0.33	29.770	0.58	16.552	0.33	26.170	0.53	6.983	0.23	10.535	0.36	6.984	0.25	10.536	0.36
Ours (ms n)	18.367	0.52	29.690	1.18	16.486	0.50	26.077	1.16	6.949	0.25	10.493	0.75	6.951	0.31	10.494	0.76
Ours (ms $n\times 8$)	18.337	0.85	29.611	4.30	16.437	0.84	25.972	4.27	6.930	0.62	10.445	3.86	6.931	0.70	10.446	3.91
Ours (ro 4×64)	18.393	0.73	29.778	2.72	16.467	0.72	26.084	2.90	6.921	0.56	10.425	2.30	6.924	0.61	10.426	1.77
Ours (ro $4\times 64\times 8$)	18.336	2.92	29.614	12.2	16.403	2.89	25.916	12.1	6.909	2.76	10.381	11.9	6.911	2.84	10.383	11.9
	OVRPBLTW				OVRPBTW				OVRPL				OVRPLTW			
PyVRP	11.673	50.0	19.162	100	11.673	50.0	19.162	100	6.507	50.0	9.719	100	10.513	50.0	16.928	100
MTPOMO (ms $n\times 8$)	11.816	0.14	19.635	0.26	11.813	0.14	19.633	0.26	6.715	0.12	10.210	0.23	10.670	0.14	17.419	0.26
MVMoE (ms $n\times 8$)	11.822	0.21	19.637	0.39	11.819	0.21	19.638	0.40	6.704	0.18	10.180	0.35	10.671	0.20	17.417	0.39
RF-TE (ms $n\times 8$)	11.804	0.14	19.547	0.27	11.805	0.14	19.547	0.27	6.681	0.12	10.117	0.24	10.652	0.13	17.325	0.26
CaDA (ms $n\times 8$)	11.760	0.14	19.434	0.26	11.760	0.14	19.434	0.25	6.667	0.12	10.120	0.23	10.612	0.14	17.224	0.25
CCL (ms $n\times 8$)	11.720	0.25	19.396	0.43	11.721	0.25	19.399	0.43	6.632	0.22	10.073	0.38	10.568	0.24	17.183	0.42
CCL_{ReLD} (ms $n\times 8$)	11.717	0.27	19.329	0.45	11.717	0.27	19.333	0.45	6.607	0.23	10.012	0.40	10.563	0.26	17.111	0.44
$\text{MoSES}_{\text{CaDA}}$ (ms $n\times 8$)	11.756	0.33	19.433	0.51	11.755	0.32	19.433	0.51	6.616	0.30	10.076	0.47	10.605	0.32	17.211	0.50
Ours (greedy)	11.759	0.31	19.383	0.47	11.761	0.28	19.382	0.42	6.665	0.32	10.031	0.50	10.608	0.29	17.168	0.46
Ours (greedy $\times 8$)	11.717	0.37	19.286	0.59	11.717	0.37	19.284	0.57	6.603	0.23	9.926	0.39	10.565	0.34	17.069	0.54
Ours (ms n)	11.699	0.53	19.252	1.21	11.699	0.53	19.250	1.21	6.567	0.31	9.886	0.94	10.543	0.52	17.034	1.18
Ours (ms $n\times 8$)	11.685	0.91	19.213	4.51	11.686	0.91	19.212	4.53	6.545	0.72	9.838	4.20	10.529	0.89	16.991	4.47
Ours (ro 4×64)	11.713	0.76	19.305	3.10	11.714	0.76	19.303	3.01	6.533	0.64	9.808	2.55	10.551	0.75	17.062	3.04
Ours (ro $4\times 64\times 8$)	11.686	3.11	19.223	12.8	11.685	3.11	19.224	12.8	6.519	2.83	9.766	11.9	10.525	3.07	16.988	12.6