PACER: TIME BUDGET-AWARE NEURAL LOCAL SEARCH FOR VEHICLE ROUTING PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Within neural combinatorial optimization, learning-augmented local search has emerged as a practical approach to large-scale instances: it iteratively selects a subproblem and re-optimizes it with a repair operator, where either component can be learned or expert-designed. However, existing methods are often budget-agnostic. The learned policy make the same decisions whether the compute budget (e.g., wall-clock time) is tight or generous. This overlooks the budget-dependent opportunity cost of each decision and locks them into a one-size-fits-all way of spending compute, which yields suboptimal performance across the budget. In this study, we introduce *PACER*, a time-budget-aware neural local search framework that explicitly schedules the **pace** of the search process conditioned on the given compute budget. At each improvement step, *PACER* jointly decides which subproblem to re-optimize and how much solver time to allocate to it. We train *PACER*'s subproblem selection and time allocation neural policy through imitation learning. To generate supervision, we roll out beam trees over subproblem selections and probe the solver's objective trajectory over time for each expanded subproblem. We then relabel each search state with downstream action values for any remaining budget through dynamic programming, turning each subproblem's value under different budgets into supervision. Experiments on 500-customer instances of the capacitated vehicle routing problem (CVRP) show that *PACER* improves the quality–runtime trade-off over budget-agnostic neural local search baselines and strong CVRP solvers under budgets of 1–50 seconds, reducing the optimality gap by up to 1.90%, and remains competitive at longer budgets beyond the training range. Counterfactual analyses that vary only the budget input show that the policy shifts both its solver-time allocation and its subproblem preferences as the remaining budget changes. These results suggest that the compute budget could reshape optimization decisions in the local search. Since real-world combinatorial optimization is a test-time computation that delivers solutions under finite budgets, learning how to spend a budget matters as much as learning where to search.

1 INTRODUCTION

Learning-based methods have emerged as a promising complement to classical solvers for NP-hard combinatorial optimization problems, such as vehicle routing problems (VRPs) and job shop scheduling problems (JSSPs) (Laporte, 2009; Li et al., 2025a). Learning-based methods seek to exploit patterns learned from data to improve solution quality within shorter time. Depending on where learning is introduced, these methods can support end-to-end solution construction (Kool et al., 2019; Kwon et al., 2020; Sun & Yang, 2023; Luo et al., 2025), automated solver configuration (Ye et al.; Schede et al., 2025), or the guidance of local search (Hottung & Tierney, 2020; Li et al., 2021; Xin et al., 2021; Cheng et al., 2023; Zheng et al., 2024; Hottung et al., 2025; Ouyang et al., 2025; Zhou et al., 2025; Pan et al., 2025; Yin et al., 2026).

Among these directions, learning-augmented local search, which we refer to as *neural local search*, has become a particularly effective paradigm for improving incumbent solutions on large-scale problems. Its appeal stems from the poor scalability of the alternatives: end-to-end neural constructive methods, whether autoregressive (Kool et al., 2019; Kwon et al., 2020) or non-autoregressive models (Sun & Yang, 2023; Luo et al., 2025; Li et al., 2024; 2025b), face substantial computational and

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.

generalization challenges when used as standalone solvers on large instances, while exact solvers such as Gurobi and VRPSolver (Pessoa et al., 2020) become prohibitively expensive.

Most learned policies in neural local search, however, remain agnostic to the compute budget. For instance, wall-clock time, typically the budget fixed in advance, is treated only as an external stopping rule. As a result, existing methods fail to adapt their decisions to the different budgets that arise across planning contexts. These methods either greedily select the most promising subproblem at each iteration and delegate it to an off-the-shelf solver (Li et al., 2021; Cheng et al., 2023; Yin et al., 2026), or train sequential selection policies via reinforcement or imitation learning that adapt to the incumbent solution and search history (Wu et al., 2021; Sonnerat et al., 2022; Hottung et al., 2025). In both cases, neither the choice of subproblem nor the effort spent re-optimizing it depends on the remaining time. From the same incumbent, they therefore select the same subproblem and solve it with the same effort, whether ample time remains or the deadline is imminent.

In practical deployment, optimization systems often operate under strict and varying compute budgets (Chmiela et al., 2021; Nof & Strichman, 2021; Kool et al., 2023). Meeting a budget well requires more than stopping when it runs out, because candidate subproblems differ widely in their solving profiles, that is, in how much and how fast they improve as budget is spent. A large subproblem may be worth its cost early in the search but wasteful near the end of the wall-clock time budget, whereas a small, quickly improving subproblem shows the opposite pattern. The value of each decision thus depends on the remaining budget (Boddy & Dean, 1994; Ma et al., 2026b). The budget should therefore be treated not merely as an evaluation constraint, but as a decision resource that the local search policy must allocate. Allocating it well is difficult, however, because these heterogeneous profiles are unknown until each subproblem is probed by the solver. Probing at test time is infeasible: subproblem–runtime pairs grow combinatorially over the search, and every probe consumes the very budget being allocated.

To fill this gap, we propose ***PACER***, a budget-aware learning framework that schedules the **pace** of neural local search. We instantiate *PACER* on the Capacitated Vehicle Routing Problem (CVRP), a representative testbed for large-scale combinatorial optimization, using wall-clock time as the primary budget. *PACER* follows an iterative decompose-and-improve paradigm: at each step, the incumbent solution is decomposed into subproblems, each a self-contained VRP that can be delegated to an off-the-shelf solver for a chosen duration and stitched back into the incumbent. Conditioning on both the search state and the remaining wall-clock budget, the learned neural network jointly selects which subproblem to solve and how much solver time to allocate to it, adapting its search behavior as the budget is consumed.

We evaluate *PACER* on large-scale CVRP-500 instances under aligned wall-clock budgets ranging from 1 to 50 seconds. Across these budgets, *PACER* improves solution quality over off-the-shelf solvers such as PyVRP and LKH, as well as learning-based decomposition baselines, and interpolates to intermediate budgets unseen during training. Our analysis shows that the budget shifts both subproblem selection and runtime allocation, and that the budget-aware policy outperforms an otherwise identical budget-blind variant. These findings support modeling the time budget as part of the policy state, a step toward budget-aware neural combinatorial optimization.

Our contributions are summarized as follows:

1. We formulate time-budgeted neural local search as a sequential decision problem with joint subproblem–time actions, in which the remaining wall-clock budget is part of the state rather than a passive stopping condition.
2. We propose *PACER*, a time-budget-aware neural local search framework that decomposes the incumbent solution into self-contained CVRP subproblems and learns to rank joint actions specifying both where to search and how much solver time to spend.
3. We develop an offline oracle-collection procedure that probes subproblem–runtime actions once via budget-agnostic beam expansion and relabels the resulting search tree under multiple budgets through dynamic programming. This turns a single round of solver calls into imitation-learning targets across budgets, decoupling CPU-bound solver probing from GPU-based neural network training for efficient data collection and training.
4. We show on CVRP-500 instances that *PACER* improves solution quality over off-the-shelf solvers and learning-based decomposition baselines across budgets from 1 to 50 seconds, and analyze how the budget shapes subproblem selection and time allocation.

2 Related Work

2.1 Learning-Augmented Combinatorial Optimization

Neural methods for combinatorial optimization have developed along two broad directions. One line learns end-to-end constructive policies, including autoregressive models (Kool et al., 2019; Kwon et al., 2020) and non-autoregressive or diffusion-based models (Sun & Yang, 2023; Luo et al., 2025; Li et al., 2024; 2025b). These approaches can produce solutions directly, but their performance and generalization often degrade as routing instances become large. A second line builds hybrid systems that combine learned components with classical search or solvers. This hybrid direction is especially relevant for large vehicle-routing instances, where mature local-search and metaheuristic solvers remain difficult to replace.

Learning-guided local search follows thehis hybrid pattern. Classical large-neighborhood and decomposition heuristics repeatedly restrict search to a local region of the incumbent and re-optimize that region (Ahuja et al., 2002; Ropke & Pisinger, 2006; Santini et al., 2023). Recent neural variants learn to guide decomposition, neighborhood selection, candidate generation, or solver calls while delegating the expensive re-optimization step to an established heuristic or solver (Hottung & Tierney, 2020; Li et al., 2021; Xin et al., 2021; Cheng et al., 2023; Zheng et al., 2024; Hottung et al., 2025; Ouyang et al., 2025; Zhou et al., 2025; Pan et al., 2025; Yin et al., 2026; Chen et al., 2026).

2.2 Solver-Level Runtime Control

Allocating computation over time has a long history in anytime algorithms and deliberation scheduling (Boddy & Dean, 1994). In optimization, algorithm-configuration methods tune solver parameters for better quality–runtime trade-offs (Adenso-Díaz & Laguna, 2006), and anytime configuration and algorithm selection recognize that different algorithms or configurations may be preferable at different runtimes (Jesus et al., 2020; Schede & Tierney, 2026). Real-time algorithm portfolios allocate limited computation across algorithms (Nof & Strichman, 2021), and in mixed-integer programming, learned schedules of primal heuristics inside branch-and-bound find good incumbents earlier (Chmiela et al., 2021; 2023). These works treat computation time as a key resource, but control it at the level of solver parameters, algorithm portfolios, or heuristic schedules.

3 Method

3.1 *PACER* Overview

PACER casts local search under a finite budget, such as wall-clock time, as a sequential resource-allocation process. At each step, the policy selects a joint action $a = (p, \tau)$: a subproblem p to re-optimize and the solver runtime τ to allocate to it. Since the value of this action depends on both the incumbent solution x and the remaining budget b, *PACER* learns a neural policy to rank joint actions conditioned on (x, b).

Figure 1 illustrates the overall *PACER* process. At each decision state, *PACER* supplements the current incumbent solution with the remaining wall-clock budget (step 1), generates candidate subproblems and feasible solver-time allocations (step 2), ranks the resulting joint subproblem–time actions (step 3 and step 4), delegates the selected action to a classical solver (step 5), and updates the solution and remaining budget until the budget is exhausted. Algorithm 1 in Appendix A summarizes this decision loop, which follows a generic decomposition-based local search framework (Ouyang et al., 2025). During inference, *PACER* also leverages multi-core CPUs to optimize disjoint subproblems in parallel to further improve the solution.

3.2 Time-Budgeted Local Search Formulation

We consider time-budgeted local search for a minimization problem, instantiated as the CVRP. For an instance I, let $c_I(x)$ denote the objective value of a feasible solution x. Starting from a warm-start solution x_0, the search iteratively improves the incumbent with a delegated anytime solver $\mathcal{S}$ and returns the best solution found within a wall-clock budget B_0; b_t denotes the remaining budget at step t, with $b_0 = B_0$. At each step, a decomposition operator $\mathcal{D}$ maps the incumbent x_t to a finite set of

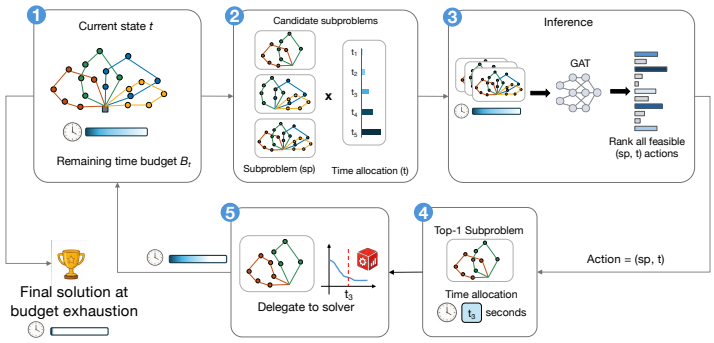


Figure 1: Overview of the *PACER* process for time-budget-aware neural local search. *PACER* learns to treat the remaining wall-clock budget as part of the decision state, ranks feasible subproblem–time actions, delegates the selected subproblem and time allocation to the solver, and repeats the process until budget exhaustion.

candidate subproblems $\mathcal{P}_t = \mathcal{D}(I, x_t)$, each a subset of routes or customers that can be re-optimized while the rest of the solution is held fixed.

Let $x_t^\star \in \arg\min_{0\le k\le t} c_I(x_k)$ denote the best-so-far incumbent solution. The search state is $s_t = (I, x_t, x_t^\star)$, which also determines $\mathcal{P}_t$, and the decision state is (s_t, b_t). Because solver $\mathcal{S}$ returns a feasible solution whenever interrupted and its improvement profile varies across subproblems, each action specifies both which subproblem to solve and for how long: $a_t = (p_t, \tau_t)$, where $p_t \in \mathcal{P}_t$ and $\tau_t \in \mathcal{T}$ is the solver time, including overhead, charged to the budget. The feasible action set $\mathcal{A}(s_t, b_t) = \{(p, \tau) : p \in \mathcal{P}_t,\ \tau \in \mathcal{T},\ \tau \le b_t\}$ therefore depends on the remaining budget. Executing a_t re-optimizes p_t and stitches the result back into the incumbent, $\tilde{x}_{t+1} = \mathsf{Solve}_{\mathcal{S}}(I, x_t, p_t, \tau_t)$; an acceptance rule such as hill-climbing sets $x_{t+1} = \mathsf{Accept}(x_t, \tilde{x}_{t+1})$, and $b_{t+1} = b_t - \tau_t$. Since $\mathcal{P}_{t+1} = \mathcal{D}(I, x_{t+1})$, each action also changes the subproblems available at future steps (Section 3.4).

We model the above time-budgeted local search as a budget-augmented Markov decision process whose reward is the normalized improvement of the best-so-far solution, $r_t = \big(c_I(x_t^\star) - c_I(x_{t+1}^\star)\big)/c_I(x_0)$, where $x_{t+1}^\star \in \arg\min_{x\in\{x_t^\star, x_{t+1}\}} c_I(x)$. Rewards telescope to $\sum_{t=0}^{T-1} r_t = \big(c_I(x_0) - c_I(x_T^\star)\big)/c_I(x_0)$, where T is the first step at which no feasible action remains, so maximizing cumulative reward maximizes the final improvement. For a policy π, the budget-conditioned value and action-value functions are

$$V^\pi(s, b) = \mathbb{E}_{\pi,\mathcal{S}}\Big[\sum_{k=t}^{T-1} r_k \,\Big|\, s_t = s,\ b_t = b\Big], \tag{1}$$

$$Q^\pi(s, b, a) = \mathbb{E}_{\mathcal{S}}\big[r + V^\pi(s', b - \tau) \,\big|\, s, b, a = (p, \tau)\big], \tag{2}$$

where s' is the successor state and the expectations are over policy decisions and solver stochasticity. *PACER* learns to approximate the optimal budget-aware policy $\pi^\star(s, b) \in \arg\max_{a\in\mathcal{A}(s,b)} Q^\star(s, b, a)$.

3.3 Initial Solution Construction

Following the sweep-based construction strategy (Gillett & Miller, 1974), each CVRP instance is warm-started with a feasible solution before the budget-aware local search begins. Each cluster, together with the depot, defines an independent and self-contained CVRP subproblem. Because these subproblems are smaller than the original instance and mutually independent at the initialization stage, they can be solved in parallel by leveraging multi-core CPU resources. Each cluster-level subproblem is solved with an off-the-shelf CVRP solver under a fixed configuration. The resulting per-cluster routes are then concatenated to form the initial incumbent solution.

3.4 Candidate Subproblem Generation

At each decision step t, *PACER* employs a decomposition operator $\mathcal{D}$ adopted from (Li et al., 2021; Santini et al., 2023) to the incumbent solution x_t and obtains a candidate set $\mathcal{P}_t = \mathcal{D}(I, x_t)$.

More specifically, for each route in the incumbent solution, *PACER* computes its route centroid and gathers spatially nearby routes until either a route-count limit or a customer-count cap is reached. Rather than using a single customer-count cap, *PACER* sweeps over a set of caps, denoted by $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, to generate candidate subproblems at multiple spatial and computational scales. This design leads to a diverse neighborhood set with **heterogeneous** optimization profiles, ranging from small, fast-to-solve subproblems to larger subproblems with higher improvement potential but longer solver time.

3.5 Motivating Analysis: Heterogeneous Subproblem–Runtime Profiles

PyVRP profiles at two search steps show that candidate subproblems differ in their response to solver time and can change rank across time allocations, motivating joint subproblem–runtime decisions. The full analysis and visualization can be found in Appendix E.

3.6 Offline Rollout and Oracle Trajectory Collection

PACER constructs supervision offline, separating expensive CPU-bound solver probing from GPU-based policy learning. The procedure has two stages: a budget-agnostic beam search probes the solver and records the outcomes of subproblem–runtime actions in a tree, and dynamic programming then relabels this tree under multiple budgets to produce budget-conditioned action values for training the policy π_θ (pseudocode in Appendix B).

3.6.1 Budget-Agnostic Beam-Tree Rollout

Since enumerating all action sequences is intractable, *PACER* builds a beam-search tree $\mathcal{G}(I) = (\mathcal{V}, \mathcal{E})$ for each training instance I. Each node u stores the incumbent $x(u)$, the best-so-far solution $x^\star(u)$, and the elapsed solver time $\ell(u)$, with $x(u_0) = x_0$ and $\ell(u_0) = 0$ at the root. At each retained node, the solver probes every action $a = (p, \tau)$ with $p \in \mathcal{P}(u) = \mathcal{D}(I, x(u))$ and τ drawn from a finite set of runtime milestones $\mathcal{T} = \{\tau^{(1)}, \ldots, \tau^{(L)}\}$. The successor v has $x(v) = \mathsf{Accept}\big(x(u), \mathsf{Solve}_{\mathcal{S}}(I, x(u), p, \tau)\big)$, $x^\star(v) \in \arg\min_{x \in \{x^\star(u), x(v)\}} c_I(x)$, and $\ell(v) = \ell(u) + \tau$, and the edge $e = (u, v)$ stores its action $a_e = (p_e, \tau_e)$ and immediate improvement $r_e = \big(c_I(x^\star(u)) - c_I(x^\star(v))\big)/c_I(x_0)$.

To keep the tree tractable yet diverse, expansion follows budget-agnostic rules with a fixed beam width. At each node, we retain the m_{imp} edges with the largest r_e, the m_{eff} edges with the largest efficiency $r_e/(\tau_e + \epsilon)$, and m_{rnd} random edges. Among the resulting successors, the next frontier keeps the n_{obj} nodes with the largest cumulative improvement $R(v) = \big(c_I(x_0) - c_I(x^\star(v))\big)/c_I(x_0)$, the n_{eff} nodes with the largest cumulative efficiency $R(v)/(\ell(v) + \epsilon)$, and n_{rnd} random nodes, until the collection budget is exhausted. Because no rule conditions on a deadline, the same tree can be reused for any training budget. The tree thus acts as a rollout-based expert (Bertsekas, 2021), striking a balance between myopic one-step evaluation and exhaustive expansion.

3.6.2 Budget-Conditioned Oracle Relabeling

For a target budget $B \in \mathcal{B}$, the remaining budget at node u is $b(u, B) = B - \ell(u)$, and its feasible outgoing edges are $\mathcal{E}(u, b) = \{e \in \mathrm{Out}(u) : \tau_e \le b\}$. The oracle value of u under remaining budget b is the largest additional improvement attainable within the subtree rooted at u:

$$V_{\mathcal{G}}(u, b) = \max\Big\{0, \max_{e=(u,v)\in\mathcal{E}(u,b)} \big[r_e + V_{\mathcal{G}}(v, b - \tau_e)\big]\Big\}, \tag{3}$$

where 0 corresponds to stopping. Each feasible edge then receives the label $Q^{\text{oracle}}_{\mathcal{G}}(u, b, e) = r_e + V_{\mathcal{G}}(v, b - \tau_e)$. These labels are exact within $\mathcal{G}$ and approximate $Q^\star$ to the extent that $\mathcal{G}$ covers good actions. Evaluating the recursion for every B turns a single tree into supervision across budgets.

3.7 Imitation Learning Based on Oracle Trajectories

3.7.1 Learning to Rank Budget-Conditioned Actions

PACER learns a budget-conditioned action ranker $q_\theta(u, b, a) = \widehat{Q}_{\theta,\mathcal{S}}(\bar{s}(u), b, a)$ from offline oracle supervision, where $\bar{s}(u)$ is the policy-observed state and $\mathcal{A}(u, b) = \{(p_e, \tau_e) : e \in \mathcal{E}_{\mathcal{G}}(u, b)\}$ is the feasible action set. Each edge's oracle value is assigned to its associated action $a = (p_e, \tau_e)$.

For each state–runtime pair, oracle values and predicted scores define distributions over $\mathcal{A}(u, b)$:

$$\rho(a \mid u, b) = \frac{\exp(Q_{\mathcal{G}}^{\text{oracle}}(u, b, a)/\alpha)}{\sum_{a' \in \mathcal{A}(u,b)} \exp(Q_{\mathcal{G}}^{\text{oracle}}(u, b, a')/\alpha)},$$
$$\pi_\theta(a \mid u, b) = \frac{\exp(q_\theta(u, b, a))}{\sum_{a' \in \mathcal{A}(u,b)} \exp(q_\theta(u, b, a'))}, \tag{4}$$

where $\alpha > 0$ controls the target distribution's sharpness. The policy is trained by minimizing listwise cross-entropy:

$$\mathcal{L}_{\text{rank}}(\theta) = - \sum_{(u,b) \in \mathcal{D}_{PACER}} \sum_{a \in \mathcal{A}(u,b)} \rho(a \mid u, b) \log \pi_\theta(a \mid u, b). \tag{5}$$

This objective teaches the policy to rank feasible actions by their budget-conditioned downstream value while retaining supervision from multiple promising actions.

3.7.2 Neural Network Architecture

The policy network scores the feasible action set jointly for each state–budget pair (u, b). Its input consists of the policy-observed search state $\bar{s}(u)$, the remaining budget b, and the feasible subproblem–time action set $\mathcal{A}(u, b)$. Each action $a \in \mathcal{A}(u, b)$ is represented as $a = (p, \tau)$, where p is the candidate subproblem and τ is its proposed solver time.

The state $\bar{s}(u)$ is represented as a graph over the depot and customers, with node and edge features encoding customer attributes, geometric proximity, and the incumbent route structure. A graph attention network (GAT) encoder computes node embeddings once for the current state. For each feasible action $a = (p, \tau)$, the embeddings of the customers in subproblem p are pooled to form an action-specific subproblem representation, which is concatenated with the remaining budget, the proposed solver time τ, and the residual budget $b - \tau$.

The same multi-layer perceptron (MLP) head is applied to each action-specific representation, producing one scalar score per feasible action: $\{q_\theta(u, b, a) : a \in \mathcal{A}(u, b)\}$. Thus, a single forward pass scores all feasible subproblem–time actions for the current state–budget pair.

3.8 Inference

At each step, *PACER* scores feasible subproblem–runtime actions in a single forward pass and selects

$$a_t = (p_t, \tau_t) = \arg \max_{a \in \mathcal{A}(s_t, b_t)} q_\theta(\bar{s}_t, b_t, a). \tag{6}$$

The selected subproblem is solved and stitched into the incumbent, and search continues with the updated budget. For parallel inference, additional high-scoring, route-disjoint subproblems are solved concurrently, sharing the selected runtime τ_t and budget charge.

3.9 Budget-Effect Diagnostic

To isolate the effect of the budget, we run the policy on the same search states with different remaining budgets and compare the actions it selects. Each comparison falls into one of three cases: the action stays the same; it changes because one of the two actions is not affordable under the other budget; or it changes even though both actions are affordable under both budgets, meaning the policy has reordered its preferences. Appendix C gives the full definitions and evaluation protocol.

4 Experiments

4.1 Instances

PACER is trained and evaluated on synthetically generated large-scale CVRP instances with $n = 500$ customers. The coordinate and demand distributions follow the synthetic CVRP protocol used in Ma et al. (2026a): the depot and customer coordinates are sampled uniformly from $[0, 1] \times [0, 1]$, and customer demands are sampled as integers from $[1, 10]$. This study sets the vehicle capacity to 50. The training set contains 10,000 instances. The evaluation set contains 100 instances.

4.2 Solution Initialization

All methods start from the same warm-start solution x_0. To build x_0, we partition each instance into 10 clusters and solve them in parallel with LKH-3 (v3.0.13) (Helsgaun, 2017), using $\mathrm{RUNS} = 1$ and MAX_TRIALS equal to the number of customers.

4.3 Baseline Methods

PACER is compared with four baseline methods in two categories: (i) off-the-shelf full-instance routing solvers, **LKH-3** (Helsgaun, 2017) and **PyVRP** (Wouda et al., 2024); and (ii) learning-based decomposition methods, **L2D** (Li et al., 2021) and **UDC** (Zheng et al., 2024). Detailed descriptions and configurations are provided in Appendix D.

4.4 *PACER* Configuration

PACER uses PyVRP as its delegated subproblem solver, with eight solver-time allocations ranging from 0.001 to 10 seconds. Training supervision is relabeled under target budgets of $\{1, 10, 50\}$ seconds. We evaluate two inference variants: *PACER*-top1 executes only the highest-ranked subproblem–runtime action at each step, while *PACER*-top5 augments it with up to five additional route-disjoint subproblems, solved concurrently using the runtime allocation of the highest-ranked (top1) action for every selected subproblem. Full configuration details are provided in Appendix F.

4.5 Evaluation

We evaluate all methods at wall-clock search budgets ranging from 1 to 100 seconds, including varying overheads for each method. Performance is measured by the mean percentage objective gap to PyVRP at the same search budget, with lower values indicating better performance. The full evaluation protocol and hardware details are provided in Appendix G.

5 Results

5.1 Comparison with Baseline Methods under Different Time Budgets

PACER-top5 achieves the best mean objective value at every interpolated budget from 1 to 50 seconds, while PyVRP overtakes *PACER* in the extrapolated regime beyond 50 seconds. Table 1 compares all methods under aligned global wall-clock budgets of $t_{\mathrm{init}} + B$, where the shared solution initialization takes $t_{\mathrm{init}} = 2.52$ seconds on average. Unlike most evaluations of neural local search, which report a single final result, we compare all methods across multiple budgets. For fair comparison, UDC uses its own initialization procedure and is therefore run for the full $t_{\mathrm{init}} + B$ budget, whereas the other methods start from the same shared initial solution and are given an additional improvement budget of B seconds. Figure 2 presents the same comparison as an anytime-style curve. Since *PACER* is run separately for each prescribed budget rather than as a single continuous anytime solver, the curve should be interpreted as budget-indexed performance points connected for visualization; comparisons remain meaningful at the evaluated budget ticks.

These results show that coupling a learned search policy with a classical optimization solver can better exploit limited computational budgets than applying the solver in isolation. The objective gap of *PACER*-top5 relative to PyVRP widens from -0.92% at 1 second to a peak advantage of -1.90% at 5 seconds, then narrows to -0.12% at 50 seconds. PyVRP catches up under the extrapolated 60- and 100-second budgets, indicating that *PACER* is most effective within the trained short-to-medium budget range. *PACER*-top1 is also competitive, outperforming PyVRP from 1 through 20 seconds and remaining within 0.27% of PyVRP at 100 seconds.

Figure 3 provides a trajectory-level view of the same phenomenon. Each colored stair curve corresponds to a *PACER*-top1 rollout on the same instance but conditioned on a different declared budget. The trajectories diverge across budgets, showing that *PACER* changes the search path rather than merely running a fixed policy until a different stopping time. Longer declared budgets tend to induce longer solver calls earlier in the trajectory, while tighter budgets produce faster, smaller improvements.

Table 1: Mean objective value and relative gap to PyVRP on CVRP-500 under global wall-clock budgets. Lower is better for both metrics. The average initialization time is $t_{\text{init}} = 2.52\,\text{s}$.

	CVRP-500: Global budget $t_{\text{init}} + B$															
	1s		5s		10s		20s		30s		50s		60s		100s	
Method	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
PyVRP	63.50	+0.00	63.16	+0.00	62.86	+0.00	62.24	+0.00	61.86	+0.00	61.51	+0.00	**61.44**	**+0.00**	**61.36**	**+0.00**
LKH-3	63.83	+0.52	63.41	+0.42	63.23	+0.63	63.03	+1.31	62.80	+1.55	62.67	+1.91	62.65	+1.97	62.58	+1.97
L2D	64.18	+1.05	63.44	+0.45	63.35	+0.81	63.26	+1.66	63.26	+2.28	63.26	+2.84	63.25	+2.93	63.23	+3.02
UDC†	–	–	–	–	65.81	+4.73	65.06	+4.57	64.66	+4.59	64.37	+4.70	64.32	+4.72	64.13	+4.54
PACER-top1	63.21	−0.45	62.61	−0.85	62.37	−0.74	61.95	−0.44	61.88	+0.08	61.54	+0.10	61.51	+0.12	61.53	+0.27
PACER-top5	**62.93**	**−0.92**	**61.97**	**−1.90**	**61.82**	**−1.64**	**61.62**	**−0.96**	**61.54**	**−0.49**	**61.41**	**−0.12**	61.47	+0.05	61.44	+0.14

Note. t_{init} denotes the time used to construct the shared initial solution, and B denotes the additional improvement budget shown in the column header. Obj. denotes the final CVRP objective value, and Gap denotes the paired relative objective gap to PyVRP under the same global wall-clock budget. † UDC does not report results at 1s and 5s because it does not complete initialization for the full evaluation set under these budgets.

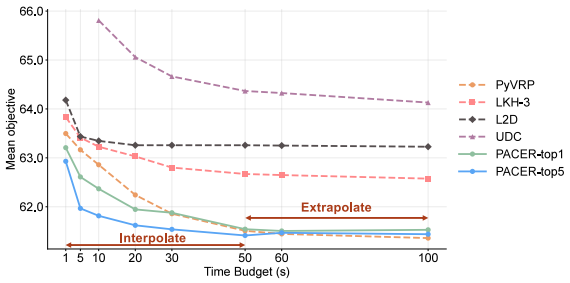


Figure 2: Mean objective values achieved by the compared methods under different global wall-clock budgets. Lower is better. *PACER* is run separately for each prescribed budget rather than as a single continuous anytime solver, so the curve should be interpreted as budget-indexed performance points connected for visualization. *PACER*-top5 dominates the 1–50 second interpolation regime and remains close to PyVRP under extrapolated 60- and 100-second budgets.

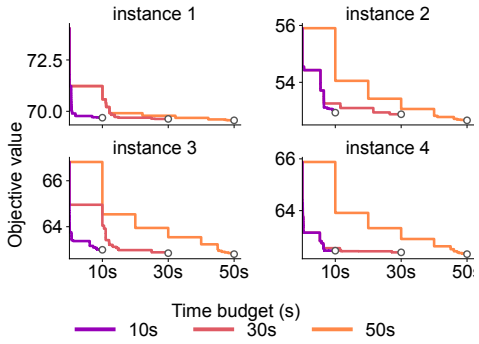


Figure 3: *PACER*-top1 anytime trajectories for four representative CVRP-500 instances under declared budgets of 10, 30, and 50 seconds. Colored step curves trace the incumbent objective value over elapsed time (lower is better), and gray circles mark the terminal solution reached under each budget. The trajectories diverge before their respective deadlines, indicating that budget conditioning changes the search path rather than merely truncating a common trajectory.

5.2 Breakdown of the Budget Effect

To characterize how the time budget affects subproblem–time decisions, we compare how learned policy selects actions at matched states given different remaining budgets. Each action selection change in the selected top-1 action is assigned to one of two mechanisms: (1) feasibility expansion,

where the longer-budget top action is infeasible under the shorter budget, and (2) preference changes, where the selected top action changes even though both actions are feasible under the shorter budget. Each mechanism is further divided according to whether the selected subproblem remains the same or changes.

Table 2 shows that most selected top-1 actions remain unchanged: 72.76% when extending the remaining budget from 1s to 50s and 70.56% for the $10/50$s shift. Feasibility expansion accounts for 25.75% and 26.01% of all matched states, respectively, representing 94.53% and 88.37% of the observed action changes. Preference changes among commonly feasible actions account for the remaining 1.49% and 3.42% of matched states. Thus, across the pooled matched states, changes in preference among already feasible actions and a newly feasible subproblem–time allocation are two major factors driving the action flip.

At the root, for $1/50$s shift, the selected action changes in 32.57% of instances, comprising 24.71% with newly feasible actions and 7.86% with preference changes among commonly feasible actions. For $10/50$s shift, the feasible root action sets are identical at both budgets, so all 26.33% of selected-action changes arise from reranking: 9.81% of instances retain the same subproblem but change its solve time, while 16.52% change the selected subproblem. This comparison shows that varying compute budget can meaningfully alter both subproblem selection and time allocation.

Table 2: Mutually exclusive budget-effect decomposition on testing instances. Each row sums to 100% before rounding.

Scope	Budgets	Mean states per instance	Unchanged	Same subproblem		Different subproblem	
				Newly runtime feasible	Preference changes	Newly runtime feasible	Preference changes
All states	$1/50$s	22.90	72.76%	3.72%	0.35%	22.03%	1.14%
All states	$10/50$s	84.91	70.56%	7.45%	0.60%	18.56%	2.82%
Root states	$1/50$s	1.00	67.43%	8.93%	3.56%	15.78%	4.30%
Root states	$10/50$s	1.00	73.67%	0.00%	9.81%	0.00%	16.52%

Note. An action specifies a subproblem and its solve time τ. Unchanged means the selected top action is identical at both budgets. Same/different subproblem indicates whether the selected subproblem changes. Newly runtime feasible means the longer-budget top action is infeasible under the shorter budget. Preference changes means both selected actions are feasible under the shorter budget, but the selected top action changes. Percentages pool matched states; all states includes roots. Top actions use deterministic tie-breaking and retained-tree values under the collector's clipped solver-call clock, excluding online decision overhead.

6 Discussion and Conclusion

This paper presents *PACER*, a time-budget-aware neural local search framework for large-scale CVRP. *PACER* formulates decomposition-based improvement as a sequential decision problem in which each action jointly specifies a subproblem and its solver time, making the remaining wall-clock budget part of the policy state. Because the joint action space is large and invoking solvers during training is expensive, *PACER* decouples solver probing from policy learning: all solver calls are made once during offline beam-tree construction, and dynamic programming relabels the same tree under multiple budgets. The resulting state–budget–action records let the policy learn budget-dependent rankings without solvers in the training loop.

Under aligned wall-clock budgets, *PACER* achieves the best quality–time trade-off among the compared solvers and learning-based baselines within the 1–50 second range, and remains competitive at longer budgets beyond the training range. Our analysis shows that the budget does more than determine when to stop: it determines which actions are affordable, shifts the policy toward longer solver calls when more time remains, and changes the policy's preferences among actions affordable under both budgets. More broadly, *PACER* illustrates how learned policies can be combined with classical operations research tools, here dynamic programming and mature routing solvers, to make search budget-aware.

AI USE STATEMENT

We used generative AI tools to assist with manuscript revision, including suggestions on wording, organization, and clarity, as well as LaTeX editing, reference formatting, and adaptation to the conference template. The authors reviewed the resulting changes against the manuscript and underlying research materials and take responsibility for the final content, including all claims, citations, and artifacts produced with AI assistance.

ETHICS STATEMENT

I acknowledge that I and all co-authors of this work have read and commit to adhering to the ICLR Code of Ethics. To the best of our knowledge, this work does not involve potential violations of the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of this work. The source code will be made available upon the acceptance. Section 3 describes the *PACER* decision framework, candidate subproblem generation, offline oracle collection, imitation learning, and inference procedures. Section 4 documents the synthetic instance-generation protocol and random seeds, initialization procedure, baseline methods, training budgets, and evaluation budgets. Appendices D and F provide detailed baseline and *PACER* configurations, respectively. Appendix G specifies the evaluation budgets, metrics, and computing environment. Algorithm 2 in the appendix provides pseudocode for offline beam-tree collection. These descriptions provide the methodological and experimental context for reproducing and interpreting the reported comparisons.

APPENDIX

A *PACER* SEARCH FRAMEWORK

Algorithm 1 Time-Budget-Aware Neural Local Search Framework

Require: CVRP instance I, total budget B_0, policy π_θ, solver $\mathcal{S}$, decomposition operator $\mathcal{D}$, time milestones $\tau \in \mathcal{T}$

Construct an initial feasible solution x_0
Set best-so-far solution $x_0^\star \leftarrow x_0$, remaining budget $b_0 \leftarrow B_0$, and $t \leftarrow 0$
while $b_t > 0$ **do**
  Generate candidate subproblems $\mathcal{P}_t \leftarrow \mathcal{D}(I, x_t)$
  Set policy-observed state $s_t \leftarrow (I, x_t, \mathcal{P}_t)$
  Score feasible actions using the budget-conditioned policy $\pi_\theta(a \mid s_t, b_t)$
  Select top action $a_t = (p_t, \tau_t)$, optionally with additional route-disjoint actions for parallel execution
  Solve and stitch the selected subproblem: $\tilde{x}_{t+1} \leftarrow \mathsf{Solve}_{\mathcal{S}}(I, x_t, p_t, \tau_t)$
  Update incumbent solution $x_{t+1} \leftarrow \mathsf{Accept}(x_t, \tilde{x}_{t+1})$
  Update best-so-far solution $x_{t+1}^\star \leftarrow \arg\min_{x \in \{x_t^\star, x_{t+1}\}} c_I(x)$
  Update remaining budget $b_{t+1} \leftarrow [b_t - \tau_t]_+$
  Set $t \leftarrow t + 1$
end while

return Best feasible solution found $x_t^\star$

B BEAM SEARCH TREE

Algorithm 2 Oracle trajectory collection

Require: CVRP instance I, maximum budget $B_{\max}$, action times $\mathcal{T}$, solver $\mathcal{S}$, decomposition strategy $\mathcal{P}$, parallel width M, expansion sizes K_Δ, K_η, K_R

Construct an initial feasible solution x_0
Let $F(x)$ be the CVRP objective, where lower is better
Create root node $v_0 = (x_0, t_0 = 0)$
Initialize oracle tree $\mathcal{G} \leftarrow \{v_0\}$ and frontier $\mathcal{Q} \leftarrow \{v_0\}$
while $\mathcal{Q}$ is not empty **do**
  Pop node $v = (x_v, t_v)$ from $\mathcal{Q}$
  if $t_v \geq B_{\max}$ **then**
    continue
  end if
  Generate candidate subproblems $\mathcal{P}(x_v)$
  Construct atomic actions

$$\mathcal{A}(v) = \{(p, \tau) : p \in \mathcal{P}(x_v), \tau \in \mathcal{T}\}.$$

  Construct feasible parallel joint actions

$$\mathcal{J}(v) = \{\mathbf{a} \subseteq \mathcal{A}(v) : |\mathbf{a}| \leq M,\ \mathrm{NonOverlap}(\mathbf{a})\}.$$

  for each joint action $\mathbf{a} \in \mathcal{J}(v)$ **do**
    Solve selected subproblems in $\mathbf{a}$ using $\mathcal{S}$
    Measure elapsed wall-clock cost $c(\mathbf{a})$
    if $t_v + c(\mathbf{a}) > B_{\max}$ **then**
      continue
    end if
    Stitch local solution(s) back into x_v to obtain x'
    Create child node $u = (x', t_v + c(\mathbf{a}))$
    Add edge $e = (v, \mathbf{a}, u, c_e)$ to $\mathcal{G}$, where $c_e = c(\mathbf{a})$
    Compute immediate improvement

$$\Delta(e) = F(x_v) - F(x')$$

    Compute time efficiency

$$\eta(e) = \frac{\Delta(e)}{\max(c_e, \epsilon)}$$

  end for
  Select expansion edges

$$\mathcal{E}_v = \mathrm{Top}_{K_\Delta}(\Delta) \cup \mathrm{Top}_{K_\eta}(\eta) \cup \mathrm{Random}_{K_R}.$$

  Push children of edges in $\mathcal{E}_v$ into $\mathcal{Q}$
end while
return Oracle search tree $\mathcal{G}$

C BUDGET-EFFECT DIAGNOSTIC

To separate the effect of the budget signal from trajectory-distribution shift, a diagnostic is performed on the testing dataset by fixing a policy-observed state s and varying only the remaining budget input. For each probed state s and two remaining budgets $b_1 < b_2$, let $\mathcal{A}(s, b) = \{a = (p, \tau) : \tau \leq b\}$ denote the feasible solver-probed subproblem–time action set under budget b, where p is the selected subproblem and τ is the solver time allocated to it. The common feasible action set is $\mathcal{A}_\cap(s; b_1, b_2) = \mathcal{A}(s, b_1) \cap \mathcal{A}(s, b_2)$. This common set excludes actions whose time allocation exceeds the smaller remaining budget.

Using the learned budget-conditioned scorer $q_\theta(s, b, a)$, the top-ranked action under budget b is defined as $a^\star_b = \arg\max_{a \in \mathcal{A}(s,b)} q_\theta(s, b, a)$. Each matched state–budget pair $(s; b_1, b_2)$ is then classified into one of the following five mutually exclusive categories:

1. **Budget-invariant action**: the top-ranked action is unchanged, $a^\star_{b_1} = a^\star_{b_2}$.
2. **Same-subproblem, compute-time feasibility switch**: the selected subproblem is invariant, but the longer-budget top action $a^\star_{b_2}$ uses a newly feasible solver time, i.e., $b_1 < \tau(a^\star_{b_2}) \leq b_2$, $p(a^\star_{b_1}) = p(a^\star_{b_2}), \qquad a^\star_{b_2} \notin \mathcal{A}(s, b_1)$.
3. **Same-subproblem, compute-time rerank**: the selected subproblem is unchanged and both top actions are feasible under the common action set, but the preferred solver time changes, $p(a^\star_{b_1}) = p(a^\star_{b_2}), \qquad a^\star_{b_1} \neq a^\star_{b_2}, \qquad a^\star_{b_1}, a^\star_{b_2} \in \mathcal{A}_\cap(s; b_1, b_2)$.
4. **Feasibility-induced subproblem–compute-time switch**: the longer-budget top action is newly feasible and selects a different subproblem, $p(a^\star_{b_1}) \neq p(a^\star_{b_2}), \qquad a^\star_{b_2} \notin \mathcal{A}(s, b_1)$.
5. **Reranking-induced subproblem–compute-time switch**: both top actions are feasible under the common action set, but the preferred subproblem changes, $p(a^\star_{b_1}) \neq p(a^\star_{b_2}), \qquad a^\star_{b_1}, a^\star_{b_2} \in \mathcal{A}_\cap(s; b_1, b_2)$.

The share of testing pairs assigned to each category is reported to quantify how the remaining budget changes the learned action ranking through feasibility expansion and action reranking. The budget-effect diagnostic is applied to ten testing instances, and the action shift at each state is probed under budgets of 1 s, 5 s, and 10 s, yielding three budget-comparison pairs: $(1\,\text{s}, 5\,\text{s})$, $(1\,\text{s}, 10\,\text{s})$, and $(5\,\text{s}, 10\,\text{s})$. For each pair, the top action at matched probed states is compared, and the root state, i.e., the state immediately after warm-up, is analyzed separately.

D BASELINE METHODS

D.1 OFF-THE-SHELF ROUTING SOLVERS

Two high-performance heuristic solvers designed for vehicle routing problems are considered: **LKH-3** version 3.0.13 (Helsgaun, 2017) and **PyVRP** (Wouda et al., 2024). For each evaluation instance, the solver is initialized with the shared warm-start CVRP solution. Each solver is then run on the full instance under the same wall-clock search budget B.

D.2 LEARNING-BASED DECOMPOSITION METHODS

L2D (Li et al., 2021) is a budget-agnostic learned baseline that greedily selects subproblems to delegate to LKH-3. We use its official code and checkpoints, which are trained with LKH (`RUNS` $= 5$, `MAX_TRIALS` equal to the number of customers), and run it from the same warm-start solution under the same budget B as *PACER*.

UDC (Zheng et al., 2024) is a neural divide-and-conquer method for large-scale routing. Since it constructs its own initial solution, it receives a total runtime of $t_{\text{init}} + B$, where t_{init} is *PACER*'s initialization time.

E Motivating Analysis: Heterogeneous Subproblem–Runtime Profiles

To characterize the heterogeneity of subproblem–runtime profiles, we utilize PyVRP as the off-the-shelf CVRP solver and measure how the objective improves as PyVRP is given more time on each candidate subproblem at two search steps: Step 0 uses subproblems generated from the warm-start solution (Section 3.3), and Step 1 uses subproblems generated after one improvement step (Section 3.4). As Figure 4 shows, these profiles are heterogeneous in three respects. First, their shapes differ across subproblems: some deliver most of their improvement within very short solver calls and then saturate, others improve only after longer allocations, and some remain nearly flat. Second, their ranking differs across time allocations: a subproblem that looks unattractive under a short allocation can become competitive under a longer one. Third, they depend on the search state: the candidates regenerated at Step 1 exhibit different profiles from those at Step 0. Because a subproblem's value thus depends on both the time allocated to it and the current search state, which subproblem to solve and how long to solve it must be decided jointly.

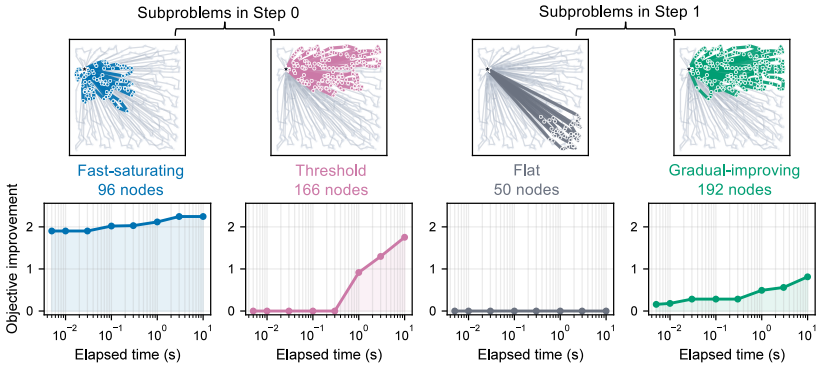


Figure 4: Heterogeneous subproblem–runtime profiles at two local-search steps. Candidate subproblems differ in both their attainable improvement and their response to solver time, so their ranking depends on the time allocated.

F *PACER* CONFIGURATION

F.1 CANDIDATE SUBPROBLEM GENERATION

At each decision step, *PACER* constructs a set of candidate subproblems from the current incumbent solution. The number of neighboring routes considered when forming subproblems is not explicitly restricted; specifically, `route_neighbors` is set to 99, which effectively removes the route-neighborhood restriction on CVRP-500. Instead, the decomposition space is controlled through the maximum number of customers allowed in each subproblem. To generate a diverse set of subproblems with heterogeneous optimization profiles, the maximum number of customers per subproblem is varied by setting $\texttt{max_nodes} \in \{50, 100, 150, 200, 300\}$. The union of the resulting subproblems forms the candidate set at each state. On CVRP-500, this procedure produces roughly 250 candidate subproblems per instance, yielding a large action space with varying improvement potential and solver-time requirements.

F.2 SOLVER-TIME ACTIONS

The solver-time choices are discretized into eight choices: $\mathcal{T} = \{0.001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10\}$ seconds. Therefore, each *PACER* action consists of both a subproblem choice and a solver-time allocation, with up to $250 \times 8 = 2,000$ combinations. This joint action space allows *PACER* to decide not only where to search, but also how much time to allocate to the selected subproblem.

F.3 DELEGATED SUBPROBLEM SOLVER

Unless otherwise stated, PyVRP is used as the delegated subproblem solver for both offline supervision collection and policy evaluation.

F.4 BUDGET-CONDITIONED RELABELING

Each completed beam tree is relabeled under multiple target budgets, $\mathcal{B} = \{1, 10, 50\}$ seconds. This relabeling procedure allows the same underlying search trajectories to provide supervision for both (very-)short-budget and long-budget decision making. As a result, the stored training data contain budget-dependent action preferences across different remaining budgets.

F.5 STORED SUPERVISION RECORDS

During offline collection, each stored graph state contains node features, route-induced graph edges, subproblem node sets, subproblem-level features, global context features, and tabular subproblem features. Each supervised action record contains both the selected subproblem and the delegated solver-time choice. Together, these records define compact state–budget–action supervision examples for training the *PACER* policy.

F.6 NEURAL NETWORK ARCHITECTURE

The model uses a graph neural encoder to embed the current CVRP solution state. Node features are encoded by a multi-layer GNN with hidden dimension 128, three message-passing layers, and four attention heads. The final scoring head is a three-layer MLP that maps the concatenated subproblem-action representation to a scalar action score.

F.7 HYPERPARAMETERS

The neural network is optimized with AdamW using learning rate 1×10^{-3}, weight decay 1×10^{-4}, gradient clipping at norm 5.0, and batch size 64.

F.8 INFERENCE

At each inference step, *PACER* scores all candidate (subproblem, time-milestone) actions and takes the highest-scoring one as a pivot (referred to as *PACER*-top1). Starting from the pivot's subproblem,

it greedily selects up to $M = 5$ additional subproblems (referred to as *PACER*-top5) in descending order of policy score, admitting a subproblem only if its routes are disjoint from those of all already-selected ones. The resulting set of up to $M+1$ mutually route-disjoint subproblems is solved concurrently, each allotted the pivot's predicted time milestone as its solve budget. Because the selected subproblems share no routes, their improving moves are non-conflicting and can be applied together in a single batched update before the next decision step.

G EVALUATION

G.1 WALL-CLOCK BUDGETS

Unlike prior works that report performance at a single runtime checkpoint, this study evaluates all methods under multiple explicit wall-clock checkpoints. This protocol compares how quickly each method reaches high-quality solutions, with method-specific overheads counted in the reported wall-clock time. The main evaluation budgets are $B \in \{1, 5, 10, 20, 30, 40, 50, 60, 100\}$ seconds. Because *PACER* is trained on target budgets $\{1, 10, 50\}$ seconds, the 5-, 20-, 30-, and 40-second evaluations test interpolation across budgets, while the 60- and 100-second evaluations test extrapolation beyond the training range.

G.2 EVALUATION METRICS

With the wall-clock time limit aligned across methods, performance is evaluated using the relative objective gap to the PyVRP baseline under the same search budget. Let c_B^m denote the final objective value produced by method m under budget B, and let c_B^{PyVRP} denote the corresponding final objective value produced by PyVRP. The relative gap is computed as $\mathrm{Gap}^m_{\mathrm{PyVRP}}(B) = 100 \cdot \frac{c_B^m - c_B^{\mathrm{PyVRP}}}{c_B^{\mathrm{PyVRP}}}$. A lower relative gap indicates better performance, with negative values indicating that method m achieves a lower objective than PyVRP under the same budget. Unless otherwise stated, the mean relative gap over the evaluation instances is reported.

G.3 COMPUTING ENVIRONMENT

All reported comparisons are conducted on GPU nodes equipped with NVIDIA A100 80GB GPUs and AMD Milan CPUs. This includes both learning-based methods, such as L2D, UDC, and *PACER*, and pure CPU-based baselines, such as LKH-3 and PyVRP. For learning-based methods, neural inference runs on the GPU, while the associated solver calls and CPU-side operations run on the AMD Milan CPUs. For CPU-only baselines, the methods run on the AMD Milan CPUs of the same GPU nodes without using the GPU.

Offline oracle-trajectory collection is the only component conducted on separate CPU-only nodes equipped with AMD Rome CPUs, due to the availability of high-core-count CPU clusters. These offline trajectories are used solely for supervision-data generation and are excluded from both the reported evaluation budgets and the final method comparisons.

REFERENCES

Belarmino Adenso-Díaz and Manuel Laguna. Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search. *Operations Research*, 54(1):99–114, February 2006. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.1050.0243. URL https://pubsonline.informs.org/doi/10.1287/opre.1050.0243.

Ravindra K. Ahuja, Özlem Ergun, James B. Orlin, and Abraham P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, November 2002. ISSN 0166-218X. doi: 10.1016/S0166-218X(01)00338-9. URL https://www.sciencedirect.com/science/article/pii/S0166218X01003389.

Dimitri Bertsekas. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific, 2021. URL https://books.google.com/books?hl=en&lr=&id=Hbo-EAAAQBAJ&oi=fnd&pg=PR1&dq=Rollout,+Policy+Iteration,+and+Distributed+Reinforcement+...&ots=bDl0gDT6ra&sig=rHM_tD_jSWuLTjzv5ny09hBhkUA.

Mark Boddy and Thomas L Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285, June 1994. ISSN 00043702. doi: 10.1016/0004-3702(94)90054-X. URL https://linkinghub.elsevier.com/retrieve/pii/000437029490054X.

Lvda Chen, Yang Li, and Junchi Yan. MaskCO: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 133725–133753, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/d85816bf41e54d9b0847e02249f4fcc6-Paper-Conference.pdf.

Hanni Cheng, Haosi Zheng, Ya Cong, Weihao Jiang, and Shiliang Pu. Select and optimize: Learning to solve large-scale tsp instances. In *International Conference on Artificial Intelligence and Statistics*, pp. 1219–1231. PMLR, 2023. URL https://proceedings.mlr.press/v206/cheng23a.html.

Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to Schedule Heuristics in Branch and Bound. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24235–24246. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/cb7c403aa312160380010ee3dd4bfc53-Abstract.html.

Antonia Chmiela, Ambros Gleixner, Pawel Lichocki, and Sebastian Pokutta. Online Learning for Scheduling MIP Heuristics, April 2023. URL http://arxiv.org/abs/2304.03755. arXiv:2304.03755 [math].

Billy E. Gillett and Leland R. Miller. A Heuristic Algorithm for the Vehicle-Dispatch Problem. *Operations Research*, 22(2):340–349, April 1974. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.22.2.340. URL https://pubsonline.informs.org/doi/10.1287/opre.22.2.340.

Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017. URL http://akira.ruc.dk/~keld/research/LKH/LKH-3_REPORT.pdf.

André Hottung and Kevin Tierney. Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem, November 2020. URL http://arxiv.org/abs/1911.09539. arXiv:1911.09539 [cs, stat].

André Hottung, Paula Wong-Chung, and Kevin Tierney. Neural Deconstruction Search for Vehicle Routing Problems, October 2025. URL http://arxiv.org/abs/2501.03715. arXiv:2501.03715 [cs].

Alexandre D. Jesus, Arnaud Liefooghe, Bilel Derbel, and Luís Paquete. Algorithm selection of anytime algorithms. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 850–858, Cancún Mexico, June 2020. ACM. ISBN 978-1-4503-7128-5. doi: 10.1145/3377930.3390185. URL https://dl.acm.org/doi/10.1145/3377930.3390185.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems!, February 2019. URL http://arxiv.org/abs/1803.08475. arXiv:1803.08475 [cs, stat].

Wouter Kool, Laurens Bliek, Danilo Numeroso, Yingqian Zhang, Tom Catshoek, Kevin Tierney, Thibaut Vidal, and Joaquim Gromicho. The EURO Meets NeurIPS 2022 Vehicle Routing Competition. In *Proceedings of the NeurIPS 2022 Competitions Track*, August 2023. URL https://proceedings.mlr.press/v220/kool23a.html.

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, pp. 21188–21198. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/f231f2107df69eab0a3862d50018a9b2\-Abstract.html.

Gilbert Laporte. Fifty Years of Vehicle Routing. *Transportation Science*, 43(4):408–416, 2009. ISSN 0041-1655. doi: 10.1287/trsc.1090.0301. URL https://www.jstor.org/stable/25769465.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems*, volume 34, pp. 26198–26211. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/dc9fa5f217a1e57b8a6adeb065560b38\-Abstract.html.

Sirui Li, Wenbin Ouyang, Yining Ma, and Cathy Wu. LEARNING-GUIDED ROLLING HORIZON OPTIMIZA- TION FOR LONG-HORIZON FLEXIBLE JOB-SHOP SCHEDULING. 2025a.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast T2T: Optimization Consistency Speeds Up Diffusion-Based Training-to-Testing Solving for Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30179–30206. Curran Associates, Inc., 2024. doi: 10.52202/079017-0950. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/352b13f01566ae34affacc60e98c16af-Abstract-Conference.html.

Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as Search Operator for Test-Time Scaling of Diffusion-based Combinatorial Optimization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=9JM03CQwzC.

Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=TbTJJNjumY.

Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. Ml4co-bench-101: Benchmark machine learning for classic combinatorial problems on graphs. *Advances in Neural Information Processing Systems*, 38, 2026a. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/aa9340420e8d78021c35ae984e1bda85-Abstract-Datasets_and_Benchmarks_Track.html.

Yichuan Ma, Linyang Li, Yongkang chen, Peiji Li, Xiaozhe Li, Qipeng Guo, Dahua Lin, and Kai Chen. Timely Machine: Awareness of Time Makes Test-Time Scaling Agentic, January 2026b. URL http://arxiv.org/abs/2601.16486. arXiv:2601.16486 [cs.CL].

Yair Nof and Ofer Strichman. Real-time solving of computationally hard problems using optimal algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 89(7):693–710, July 2021. ISSN 1012-2443, 1573-7470. doi: 10.1007/s10472-020-09704-4. URL https://link.springer.com/10.1007/s10472-020-09704-4.

Wenbin Ouyang, Sirui Li, Yining Ma, and Cathy Wu. Learning to Segment for Vehicle Routing Problems. In *The Fourteenth International Conference on Learning Representations*, October 2025. URL https://openreview.net/forum?id=pN261iTKvr.

Yuxin Pan, Ruohong Liu, Yize Chen, Zhiguang Cao, and Fangzhen Lin. Hierarchical Learning-based Graph Partition for Large-scale Vehicle Routing Problems, February 2025. URL http://arxiv.org/abs/2502.08340. arXiv:2502.08340 [cs].

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523, June 2020. ISSN 1436-4646. doi: 10.1007/s10107-020-01523-z. URL https://link.springer.com/article/10.1007/s10107-020-01523-z.

Stefan Ropke and David Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2006. ISSN 0041-1655. URL https://www.jstor.org/stable/25769321.

Alberto Santini, Michael Schneider, Thibaut Vidal, and Daniele Vigo. Decomposition Strategies for Vehicle Routing Heuristics. *INFORMS Journal on Computing*, 35(3):543–559, May 2023. ISSN 1091-9856, 1526-5528. doi: 10.1287/ijoc.2023.1288. URL https://pubsonline.informs.org/doi/10.1287/ijoc.2023.1288.

Elias Schede and Kevin Tierney. A method for the automated configuration of anytime portfolios of algorithms. *European Journal of Operational Research*, 329(2):577–590, March 2026. ISSN 03772217. doi: 10.1016/j.ejor.2025.07.024. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221725005491.

Elias Schede, Moritz Seiler, Kevin Tierney, and Heike Trautmann. Deep reinforcement learning for instance-specific algorithm configuration. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1190–1198, NH Malaga Hotel Malaga Spain, July 2025. ACM. ISBN 979-8-4007-1465-8. doi: 10.1145/3712256.3726480. URL https://dl.acm.org/doi/10.1145/3712256.3726480.

Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a Large Neighborhood Search Algorithm for Mixed Integer Programs, May 2022. URL http://arxiv.org/abs/2107.10201. arXiv:2107.10201 [math.OC].

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization, December 2023. URL http://arxiv.org/abs/2302.08224. arXiv:2302.08224.

Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: A High-Performance VRP Solver Package. *INFORMS Journal on Computing*, 36(4):943–955, July 2024. ISSN 1091-9856. doi: 10.1287/ijoc.2023.0055. URL https://pubsonline.informs.org/doi/10.1287/ijoc.2023.0055.

Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning Large Neighborhood Search Policy for Integer Programming. In *Advances in Neural Information Processing Systems*, volume 34, pp. 30075–30087. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/fc9e62695def29ccdb9eb3fed5b4c8c8-Abstract.html.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem. In *Advances in Neural Information Processing Systems*, volume 34, pp. 7472–7483. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/3d863b367aa379f71c7afc0c9cdca41d-Abstract.html.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization.

Zhuoli Yin, Yi Ding, Reem Khir, and Hua Cai. ViTSP: A Vision Language Models Guided Framework for Solving Large-Scale Traveling Salesman Problems. In C. Vondrick,

B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *The Fourteenth International Conference on Learning Representations*, volume 2026, pp. 132572–132599, March 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/d6e71be5a6cb61dfa3f9ffbdd6e05b3c-Paper-Conference.pdf.

Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. UDC: A Unified Neural Divide-and-Conquer Framework for Large-Scale Combinatorial Optimization Problems, 2024. URL http://arxiv.org/abs/2407.00312. arXiv:2407.00312 [cs].

Shipei Zhou, Yuandong Ding, Chi Zhang, Zhiguang Cao, and Yan Jin. DualOpt: A Dual Divide-and-Optimize Algorithm for the Large-scale Traveling Salesman Problem, January 2025. URL http://arxiv.org/abs/2501.08565. arXiv:2501.08565 [cs] version: 1.