Autonomous Agents and Multi-Agent Systems (2026) 40:27
https://doi.org/10.1007/s10458-026-09755-7

Multi-level graph partition via hierarchical learning for large-scale vehicle routing problems

Yuxin Pan[1] · Ruohong Liu[2] · Yize Chen[3] · Zhiguang Cao[4] · Fangzhen Lin[1]

Received: 17 June 2025 / Accepted: 4 May 2026
© The Author(s) 2026

Abstract
Vehicle Routing Problems (VRPs) involve multi-agent route optimization, with the objective of targeting optimal routes for a fleet of vehicles to serve a set of customers. Existing neural solvers based on the divide-and-conquer approach for VRPs in general, and capacitated VRP (CVRP) in particular, integrate the global partition of an instance with the local construction for each resulting subinstance to enhance generalization. However, during the global partition phase, misclusterings within subgraphs have a tendency to progressively compound throughout the multi-step decoding process of the learning-based partition policy. This suboptimal behavior of the partition policy, in turn, may lead to a dramatic deterioration in the performance of the overall decomposition-based system, despite using optimal local constructions. To address these challenges, we propose a versatile Hierarchical Learning-based Graph Partition (HLGP) framework, which is tailored to benefit the partition of CVRP instances by synergistically integrating global and local partition policies. Specifically, the global partition policy is tasked with creating a coarse multi-way partition to generate a sequence of simpler two-way partition subtasks. These subtasks mark the initiation of the subsequent K local partition levels. At each local partition level, subtasks exclusive to this level are assigned to the local partition policy which benefits from the insensitive local topological features to incrementally alleviate the compounded errors. This framework is versatile in the sense that it optimizes the involved partition policies towards a unified objective, which is harmoniously compatible with both reinforcement learning (RL) and supervised learning (SL) paradigms. Additionally, we decouple the synchronized training into individual training of each component to circumvent the instability issue. Furthermore, we point out the importance of treating the subproblems encountered during the partition process as individual training instances. Extensive experiments conducted on various CVRP benchmarks demonstrate the effectiveness and generalization capabilities of the HLGP framework under both scale and distribution shifts. The source code is available at https://github.com/panyxy/hlgp_cvrp.

Keywords Vehicle routing problem · Combinatorial optimization problem · Hierarchical learning

Extended author information available on the last page of the article

Published online: 23 May 2026

1 Introduction

Vehicle Routing Problems (VRPs) involve multi-agent route optimization aimed at determining optimal routes for a fleet of vehicles to serve a set of customers [1–5], with a broad spectrum of real-world applications in transportation [6] and logistic [7]. As a challenging class of NP-hard combinatorial optimization problems (COPs), exact methods [8] are often computationally prohibitive, even though they guarantee optimal solutions. Heuristic methods, such as LKH3 [9] and HGS [10], provide practical alternatives but rely heavily on domain expertise to design problem-specific, high-quality handcrafted local operators. More recently, learning-based neural solvers have emerged with empirically favorable performance, higher computational efficiency, and minimal reliance on domain expertise. Typically, neural solvers adopt one of three paradigms: the constructive method, the iterative method, and the divide-and-conquer method. The constructive method [11–16], which incrementally constructs a feasible solution, often suffers substantial performance degradation on out-of-distribution (OOD) instances, whereas the iterative method [17–22], which refines solutions by using a neural network to select heuristic local operators, requires numerous refinement steps with well-crafted local operators to obtain satisfactory solutions. By comparison, the divide-and-conquer method embraces either a heuristic-based partition policy [23–28] or a neural partition policy [29–32] to globally divide the entire instance into subinstances, which are then solved using a local construction policy. However, a failure in either component policy may lead to a significant performance drop. Moreover, heuristic-based partition rules often result in local optima, while neural partition policies may be vulnerable to distribution shifts. Hence, there is a pressing need for a more generalizable and robust partition policy, which is the focus of this paper.

In the divide-and-conquer paradigm, the local construction policy benefits from the local topological features within subproblems, which tend to be insensitive against distribution and scale shifts. This contributes to the (near-)optimality of solutions generated for individual subproblems [33–35]. However, during the multi-step decoding process of the learning-based partition policy for Capacitated VRP (CVRP) instances [30, 31], the decoding of clustered nodes at each step relies on the partial partition solution from the preceding step. This implies that inaccuracies in the earlier clustering steps have a tendency to propagate and result in a chain of misclusterings in subsequent steps, called compounded errors. Consequently, even with an optimal local construction policy, deficiencies in the partition task can lead to substantial deviations from the ideal performance of the overall system. Therefore, we argue that *the partition task occupies a critical role in the overall decomposition-based system for solving CVRP.* Furthermore, the success of the local construction policy inspires us to introduce a local partition policy, which is instead designed to progressively alleviate compounded errors by harnessing local topological features in the partition task. We thus consider to implement a hierarchical learning (HL) framework tailored specifically for the partition task in the CVRP, which is capable of seamlessly integrating both global and local partition policies. In prevailing HL frameworks, a high-level policy is adopted to derive a series of simpler subtasks which are then delegated to the low-level policy [36]. The primary objective of such frameworks is to facilitate the exploration in reinforcement learning, wherein the associated policies undergo the joint training [36, 37]. Yet, existing HL frameworks have not been extensively explored in addressing compounded errors within

the graph partition task of large-scale CVRP. In contrast, our study extends the HL framework to the partition task of CVRP and demonstrates its efficacy in mitigating compounded errors.

In this paper, we present a generalizable Hierarchical Learning-based Graph Partition (HLGP) framework, specifically tailored for the partition task that is critical to solving the large-scale CVRP. This framework synergistically integrates both global and local partition policies to alleviate compounded errors during the partition process, thereby contributing to the performance of the overall decomposition-based framework. To be specific, our method formulates the partition problem of CVRP using a multi-level HL framework. At the global partition level, the global partition policy is responsible for initiating a coarse multi-way partition to create a series of simpler two-way partition subtasks. These subtasks stand as the starting point for the subsequent K local partition levels. At each local partition level, the sequence of tailored subtasks is derived from the partition solution of the preceding level. These subtasks are then fed into the local partition policy to form a new partition solution. Such a setup allows the local partition policy to mitigate potential misclusterings arising from the preceding level by leveraging the insensitive local topological features inherent in subtasks. By enabling the local partition policy to traverse through subtasks at each local partition level, the compounded misclusterings can be mitigated progressively across levels as a consequence.

Our proposed HLGP framework is versatile, featuring a unified objective that effortlessly accommodates both reinforcement learning (RL) and supervised learning (SL) paradigms for training the associated partition policies. It is worth noting that unlike prior approaches which directly train neural solvers via SL paradigms [38–40], our method explores the application of SL for training the partition policy, usually omitting the need for permutation information. Additionally, to reduce training instability, we propose the joint training of the involved policies in a disentangled approach. Moreover, by conducting in-depth analyses under both the RL and SL settings, we shed light on the importance of treating the subproblems encountered during the partition process as individual training instances. Empirically, the proposed HLGP framework convincingly demonstrates its superiority over prior state-of-the-art methods through extensive experiments on a range of CVRP benchmarks. Notably, our method can scale up to CVRP instances of 10,000 nodes with around 10% performance improvement over its baseline method.

The remainder of the paper is organized as follows. Section 2 reviews related learning-based methods for CVRP. Section 3 introduces the preliminaries of the CVRP formulation and the divide-and-conquer paradigm. Section 4 presents the proposed HLGP framework. Section 5 describes the experimental setup and reports the evaluation results. Finally, Section 6 concludes the paper and discusses potential extensions of HLGP to other COPs.

2 Related works

Constructive methods Constructive methods aim to develop end-to-end neural solvers for COPs, which incrementally infer complete solutions for given instances via the autoregressive mechanism. These methods feature efficient generation of (near-)optimal solutions for in-distribution instances. Among them, the Pointer Network [11] and the Transformer-based model [13] have emerged as two predominant neural architectures for neural solvers, trained

using either SL [11] or RL [12, 13]. Additionally, inherent properties of VRPs, such as the presence of multiple optima [15] and problem symmetry [16], are also considered to enhance the in-distribution performance of neural solvers. However, the delayed rewards inherent in training RL policies lead to high GPU memory demands required for the gradient backpropagation. Consequently, SL-driven policies, such as BQ [38], LEHD [39] and SIL [40], have resurfaced to alleviate training difficulties and moderately improve generalization performance on OOD instances. Moreover, tremendous efforts have been devoted to improving the generalization capabilities of neural solvers, including approaches such as meta-learning [41–44], knowledge distillation [45], or ensemble learning [34, 46, 47]. However, these constructive methods might still experience performance deterioration when encountering substantial distribution or scale shifts.

Iterative methods Unlike constructive methods, iterative methods offer the benefit of consistently refining a given solution through local updates until convergence. Both L2I [17] and NeuRewriter [18] utilize RL policies to choose from a set of predefined local improvement operators, iteratively refining solutions. Likewise, [19] interleaves the use of heuristic destroy operators and a set of learning-based repair policies to generate new solutions. Moreover, DACT [20] explores the influence of the expressiveness of solution representations on the performance of RL policy within an iterative framework. Additionally, both NeuralLKH [21] and Neural k-Opt [22] utilize RL policies to substitute heuristic rules for edge exchanges in k-opt algorithms. However, these iterative methods trade efficiency for improved performance and still rely heavily on domain-specific, well-crafted rules.

Divide-and-conquer methods Divide-and-conquer methods typically leverage local topological features that are insensitive to distribution or scale shifts, thereby mitigating the performance deterioration. Fu et al. [24], Kim et al. [25] and [28] attempt to transfer standard neural solvers for larger instances by applying them to multiple small-scale subgraphs, which are sampled using heuristic rules. In contrast, both L2D [26] and RGB [27] learn a policy to select among heuristically constructed subgraphs for iterative refinement. However, the use of heuristic rules may cause solutions to be trapped in local optima. Unlike the above methods, TAM [30], GLOP [31] and UDC [32] opt to use a learning-based policy to globally partition the entire instance into subinstances, which are subsequently solved by a pretrained local construction policy. In addition, [29] resort to a hierarchical RL framework where a local policy solves subproblems assigned by a jointly trained global policy. However, these neural partition policies may suffer performance degradation due to compounded errors during the partition process.

In summary, compared with constructive methods, the proposed HLGP framework improves generalization under distributional and scale shifts by dividing large problem instances into smaller subproblems. Compared with iterative methods, HLGP avoids the need for numerous handcrafted local operators. More importantly, compared with prior divide-and-conquer approaches, HLGP integrates both global and local partition policies to perform multi-level graph partition, which helps mitigate the compounding errors in node clustering during the partition process. Moreover, both RL and SL can be adopted for the efficient disentangled training of the corresponding partition policies to optimize the unified objective within the HLGP framework.

3 Preliminaries

CVRP formulation A CVRP instance I is defined as a 3-tuple $I = (G, D, N_{\max})$, where G is a graph consisting of a depot node v_0 and N_v customer nodes $\{v_i\}_{i=1}^{N_v}$, D represents the vehicle capacity, and $N_{\max}$ denotes the maximum number of permitted returns to the depot by vehicles. Each node is associated with coordinates (a_i, b_i), and each customer node is further assigned a demand d_i. $N_{\max}$ is accordingly defined as $\lceil \sum_{i=1}^{N_v} d_i/D \rceil + 1$ to prevent the occurrence of an empty feasible solution set. The distance between any pair of nodes is calculated using the Euclidean metric. In the CVRP, the vehicle needs to visit each customer exactly once, fulfills their demands without exceeding its capacity D, and returns to the depot to reload goods if necessary. A feasible solution $\mathcal{T} \in \mathbb{S}_{\mathcal{T}}$ can be described as a node permutation where the depot node can occur multiple times, while each customer node appears exactly once. Furthermore, the feasible solution $\mathcal{T}$ also can be decomposed into N_τ feasible subtours $\{\tau_i\}_{i=1}^{N_\tau}$. Each subtour τ_i begins and ends at the depot, with customer nodes visited in between. The travel cost $e(\tau_i)$ associated with τ_i is the sum of Euclidean distances between consecutive nodes along the subtour. Thus, the objective is to minimize the solution cost $e(\mathcal{T}) = \sum_{i=1}^{N_\tau} e(\tau_i)$. $\mathbb{S}_{\mathcal{T}}$ denotes the space of feasible solutions, as does the notation used for $\mathbb{S}$ in the following sections.

Global partition and local construction (GPLC) In the context of the CVRP, the partition task involves clustering nodes into distinct groups such that each cluster contains the depot node, each customer node belongs to exactly one cluster, and the total demand with each cluster does not exceed the vehicle capacity D. Each cluster of nodes forms a subgraph. A feasible partition solution $\mathcal{C} \in \mathbb{S}_{\mathcal{C}}$ comprises N_c subgraphs $\{c_i\}_{i=1}^{N_c}$. Each subgraph c_i consists of the depot node along with a subset of customer nodes which are distinct from those in other subgraphs. Formally, this implies $\cup_{i=1}^{N_c} c_i = G$ and for all $i \neq j$, $c_i \cap c_j = \{v_0\}$. The feasible partition solution $\mathcal{C}$ can alternatively be represented as a node list, where the order of customer nodes between two consecutive depot visits is ignored. Therefore, a feasible solution $\mathcal{T}$ can be equivalently seen as a feasible partition solution $\mathcal{C}$ when disregarding the permutation information of customer nodes in $\mathcal{T}$.

Obviously, both the feasible solution $\mathcal{T}$ (i.e., node permutation) in the CVRP and the feasible partition solution $\mathcal{C}$ (i.e., node clustering) in the partition problem of the CVRP are composed of discrete variables. This fact prompts prevalent learning-based methods to revolve around building stochastic policies as neural solvers, with the aim of handling whatever types of problems (permutation or clustering) within the context of CVRP. Let $\Delta(\cdot)$ denote the space of probability measures. In the GPLC paradigm [30, 31], a stochastic partition policy $\pi_{\text{part}}(\mathcal{C}|I) \in \Delta(\mathbb{S}_{\mathcal{C}})$ is used to derive a feasible partition solution $\mathcal{C}$ by dividing the graph G of the given instance I. Then, a set of subproblems $\{(c_i, D, 1)\}_{i=1}^{N_c}$ is yielded from the subgraphs $\{c_i\}_{i=1}^{N_c}$ in $\mathcal{C}$. Please note that, due to the feasibility of $\mathcal{C}$, the total demand within each subproblem $(c_i, D, 1)$ does not exceed the capacity D, and thus only a single depot visit is required without reloading. Therefore, a (near-)optimal local permutation policy $\pi^*_{\text{perm}}(\mathcal{T}|\mathcal{C}) \in \Delta(\mathbb{S}_{\mathcal{T}})$ is employed to generate a feasible solution $\mathcal{T} = \{\tau_i\}_{i=1}^{N_\tau}$, wherein each subtour τ_i represents a node permutation for the corresponding subgraph c_i in the partition solution $\mathcal{C}$. The objective is to identify an optimal partition policy π^*_{part} that

minimizes the solution cost $e(\mathcal{T})$ associated with the resulting solution $\mathcal{T}$. *However, prior methods lack a formal theoretical foundation for the GPLC paradigm in solving the CVRP. Therefore, we introduce* Theorem 1 *to establish the theoretical soundness of the GPLC approach for the CVRP.* Please refer to Appendix D.1 for the proof of Theorem 1.

Theorem 1 *Given a CVRP instance I, the primitive objective is to identify a (permutation) policy* $\pi(\mathcal{T}|I) \in \Delta(\mathbb{S}_{\mathcal{T}})$ *so as to minimize the expected cost, formally expressed as* $\min_\pi \mathbb{E}_{\mathcal{T}\sim\pi}[e(\mathcal{T})]$. *If* $\pi^*_{\text{perm}} \in \Delta(\mathbb{S}_{\mathcal{T}})$ *is optimal for each subproblem* $(c_i, D, 1)$, *then the primitive objective can be reformulated as the objective of the partition problem which is to identify a partition policy* $\pi_{\text{part}} \in \Delta(\mathbb{S}_{\mathcal{C}})$ *to minimize the expected cost, expressed as:*

$$\min_{\pi_{\text{part}}} \mathbb{E}_{\mathcal{C}\sim\pi_{\text{part}}}[\sum_{i=1}^{N_c} \mathbb{E}_{\tau_i\sim\pi^*_{\text{perm}}}(e(\tau_i))], \tag{1}$$

where $\pi^*_{\text{perm}}(\mathcal{T}|\mathcal{C}) = \prod_{i=1}^{N_c} \pi^*_{\text{perm}}(\tau_i|c_i)$ *implies that each* τ_i *is conditionally independent given its corresponding subgraph* c_i.

As aforementioned, a feasible partition solution $\mathcal{C}$ can be viewed as a node list, which implies that the partition policy can incrementally construct the partition solution via the autoregressive mechanism. Accordingly, the partition policy selects the current node $\mathcal{C}[n]$ conditioned on the partial partition solution $\mathcal{C}[0:n-1]$ (with $\mathcal{C}[0] = \emptyset$) and the given problem instance I, written as:

$$\pi_{\text{part}}(\mathcal{C}|I) = \prod_{n=1}^{N_{\text{sol}}} \pi_{\text{part}}(\mathcal{C}[n]|\mathcal{C}[0:n-1], I), \tag{2}$$

where N_{sol} denotes the length of partition solution. For brevity, we slightly abuse the notation N_{sol} to denote the length of different partition solutions in the following sections. Since the objective defined in (1) essentially aligns with that of RL, a common approach is to train neural solvers using RL techniques.

Theorem 1 presents an idealized algorithmic formulation for the GPLC framework, assuming access to an optimal local permutation policy. In practice, to obtain a feasible and implementable algorithm, we approximate this optimal local permutation policy using a neural solver. This approach is motivated by the empirical observation that local topological structures within subproblems remain relatively insensitive to distributional and scale shifts of the overall problem instance, which is also reported in prior works [33–35]. Such structural regularity provides a strong inductive bias, thereby enabling a neural network to effectively approximate the optimal local permutation policy across diverse instance distributions and problem scales. Accordingly, in our implementation, we employ a pretrained neural solver, as also adopted in GLOP [31], to approximate the optimal local permutation policy.

4 Hierarchical learning-based graph partition

Our proposed HLGP framework is built upon the GPLC paradigm. In line with prior works, we assume the optimal local permutation policy π^*_{perm} is obtainable by leveraging LKH3 [9] or the neural solver used in GLOP [31]. As indicated in (2), the decoding of the node at each time step hinges on the partial partition solution constructed in preceding steps. Consequently, inaccuracies in clusterings of earlier steps tend to propagate, resulting in a chain of misclusterings in subsequent steps. This issue becomes particularly pronounced under substantial distribution or scale shifts. Such empirical challenges in solving the CVRP thus motivate us to develop a HL framework for addressing the partition problem in the CVRP. We anticipate that this HL framework can progressively mitigate compounded errors by incorporating both global and local partition policies.

4.1 HL Formulation of partition problem

In this section, we begin by introducing the *feasible cost function* $f(\mathcal{C})$, which is responsible for quantifying the cost associated with a feasible partition solution $\mathcal{C}$, as formally defined in Definition 1. Following this, different forms of $f(\mathcal{C})$ are presented in the subsequent sections, designed to align with both RL and SL objectives for training the involved partition policies.

Definition 1 Let π^*_{part} denote the optimal partition policy obtained by optimizing the objective of the partition problem, as defined in (1). Given a cost function $f(\mathcal{C}) : \mathbb{S}_{\mathcal{C}} \to \mathbb{R}$, if π^*_{part} can be derived by solving the optimization problem $\min_{\pi_{\text{part}}} \mathbb{E}_{\mathcal{C}\sim\pi_{\text{part}}}[f(\mathcal{C})]$, then $f(\mathcal{C})$ is a feasible cost function.

By leveraging this well-defined feasible cost function $f(\mathcal{C})$, the objective of the partition problem can be expressed as minimizing the expected cost $\mathbb{E}_{\mathcal{C}}[f(\mathcal{C})]$. Then, we reformulate the partition problem using a multi-level HL framework, where the global partition policy π_{Gpart} and the local partition policy π_{Lpart} work together in synergy to accomplish the partition task. At the global partition level, π_{Gpart} is tasked with creating an initial coarse feasible partition $\mathcal{C}^{(0)} = \{c_1^{(0)}, \ldots, c_{N_c}^{(0)}\}$, where each $c_i^{(0)}$ (for $1 \le i \le N_c$) denotes a subgraph at this level. In this partition solution $\mathcal{C}^{(0)}$, each pair of subgraphs $(c_i^{(0)}, c_{i \bmod N_c+1}^{(0)})$ (for $1 \le i \le N_c$) is stipulated as neighboring subgraphs as defined by a specific heuristic rule. For instance, a simple heuristic involves rearranging the subgraphs in $\mathcal{C}^{(0)}$ in ascending order of the polar angles of their centroids, computed in a Polar coordinate system centered at the depot node. This coarse multi-way partition $\mathcal{C}^{(0)}$ serves as the entry point for the subsequent K local partition levels. At each local partition level $k \in \{1, ..., K\}$, subproblems specific to this level are sequentially constructed by reuniting pairs of neighboring subgraphs from the preceding partition solution $\mathcal{C}^{(k-1)}$. Each subproblem $I_j^{(k-1)}$ (for $1 \le j \le \lfloor \frac{N_c}{2} \rfloor$) is defined as follows:

$$I_j^{(k-1)} = (G_j^{(k-1)}, D, 2); \quad G_j^{(k-1)} = c_{(m+k-1) \bmod N_c+1}^{(k-1)} \cup c_{(m+k) \bmod N_c+1}^{(k-1)}, \tag{3}$$

where $m = 2(j-1)$. Thus, each local partition level contains $\lfloor \frac{N_c}{2} \rfloor$ subproblems. For each subproblem, the vehicle is allowed to return to the depot at most twice, as each subproblem is formed by merging two feasible subgraphs. Please note that each pair of consecutive subproblems $I_j^{(k-1)}$ and $I_{j+1}^{(k-1)}$ do not overlap in terms of the subgraphs they contain. Additionally, this technique for creating subproblems can be described as initially left-shifting the ordered subgraphs in $\mathcal{C}^{(k-1)}$ by $k-1$ places and then merging the neighboring subgraphs without overlaps. At each local partition level $k \geq 1$, the sequence of subproblems specific to this level is directed to the local partition policy π_{Lpart}, which enables π_{Lpart} to address potential misclusterings from the preceding level by utilizing the insensitive local topological features. As a result, the local partition policy can traverse through subproblems at each local partition level, gradually reducing accumulated misclusterings across levels. Moreover, upon completion of solving the subproblem $I_j^{(k-1)}$, the corresponding pair of subgraphs $(c^{(k-1)}_{(m+k-1) \bmod N_c+1}, c^{(k-1)}_{(m+k) \bmod N_c+1})$ in $\mathcal{C}^{(k-1)}$ is transitioned to the updated subgraph pair $(c^{(k)}_{(m+k-1) \bmod N_c+1}, c^{(k)}_{(m+k) \bmod N_c+1})$ in $\mathcal{C}^{(k)}$. Consequently, the partition solutions of the subproblems collectively result in an update of the partition solution from $\mathcal{C}^{(k-1)}$ to $\mathcal{C}^{(k)}$. Figure 1 illustrates the HLGP framework for the case where $K = 3$.

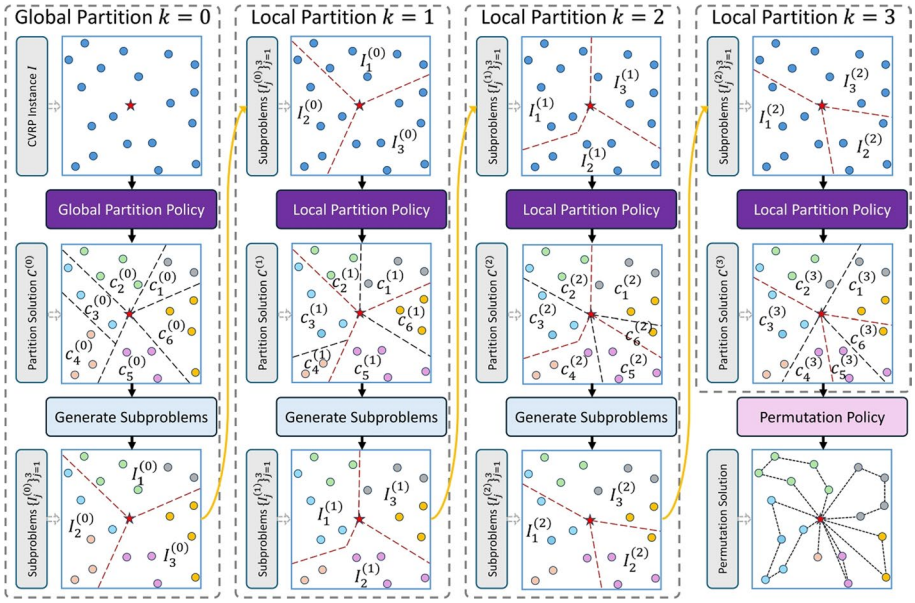

Fig. 1 This figure presents the HLGP framework with $K = 3$. The top row illustrates the problem instances, including the subproblems separated by the red lines. The blue dots represent nodes in the problem instances that have not yet been partitioned or permuted. The second row shows the result after the graph is divided into several subgraphs. The black lines indicate the partition boundaries, and different colors are used to distinguish the resulting node clusters. The third row illustrates the merging step, where two neighboring subgraphs are combined into a new subproblem for further partitioning. Both Tables 2 and 3 adopt the same color scheme. The subgraphs in the resulting partition solution $\mathcal{C}^{(3)}$, obtained from the HLGP framework, are fed to a permutation policy to derive the corresponding subtours for the CVRP solution $\mathcal{T}$

In the HLGP framework, the global partition policy π_{Gpart} is designed to incrementally construct the partition solution $\mathcal{C}^{(0)}$ via the autoregressive mechanism, where $\mathcal{C}^{(0)}$ is also viewed as a node list. Let $\mathcal{C}^{(0)}[n]$ and $\mathcal{C}^{(0)}[0:n-1]$ denote the *n*-th selected node and the partial solution consisting of the first $n-1$ nodes in $\mathcal{C}^{(0)}$, respectively. The global partition policy thus factorizes as:

$$\pi_{\text{Gpart}}(\mathcal{C}^{(0)}|I) = \prod_{n=1}^{N_{\text{sol}}} \pi_{\text{Gpart}}(\mathcal{C}^{(0)}[n]|\mathcal{C}^{(0)}[0:n-1], I). \tag{4}$$

In contrast, the local partition policy π_{Lpart} aims to separately address subproblems produced from the previous partition solution $\mathcal{C}^{(k-1)}$ to construct the updated partition solution $\mathcal{C}^{(k)}$. Let $\mathcal{C}_j^{(k-1)}$ denote the partition solution corresponding to the subproblem $I_j^{(k-1)}$. Again, the partition solution $\mathcal{C}_j^{(k-1)}$ can be either represented as a node list where $\mathcal{C}_j^{(k-1)}[n]$ and $\mathcal{C}_j^{(k-1)}[0:n-1]$ denote the *n*-th node and the corresponding partial solution, respectively, or decomposed into two subgraphs $c^{(k)}_{(m+k-1) \bmod N_c+1}$ and $c^{(k)}_{(m+k) \bmod N_c+1}$, both of which belong to the updated partition solution $\mathcal{C}^{(k)}$. Thus, the local partition policy can be expressed as:

$$\pi_{\text{Lpart}}(\mathcal{C}^{(k)}|\mathcal{C}^{(k-1)}, k) = \prod_{j=1}^{\lfloor \frac{N_c}{2} \rfloor} \prod_{n=1}^{N_{\text{sol}}} \pi_{\text{Lpart}}(\mathcal{C}_j^{(k-1)}[n]|\mathcal{C}_j^{(k-1)}[0:n-1], I_j^{(k-1)}). \tag{5}$$

Please note that in the LHS of (5), the parameter *k*, which represents the partition level, is explicitly included as an input to the local partition policy. This inclusion is essential, as the parameter *k* governs the construction of different sets of subproblems for each level. As a result, the objective of HLGP framework is to minimize the expected cost by optimizing both π_{Gpart} and π_{Lpart}, written as:

$$\min_{\pi_{\text{Gpart}}, \pi_{\text{Lpart}}} \mathbb{E}_{\mathcal{C}^{(0)} \sim \pi_{\text{Gpart}}} \mathbb{E}_{\mathcal{C}^{(1)} \sim \pi_{\text{Lpart}}} \cdots \mathbb{E}_{\mathcal{C}^{(K)} \sim \pi_{\text{Lpart}}} [f(\mathcal{C}^{(K)})]. \tag{6}$$

In contrast to the objective of the conventional GPLC paradigm, which is written as $\mathbb{E}_{\mathcal{C} \sim \pi_{\text{part}}}[f(\mathcal{C})]$, the HLGP framework introduces a multi-level hierarchical structure that employs the global partition policy and the local partition policy to collaboratively generate a sequence of partition solutions $\{\mathcal{C}^{(k)}\}_{k=0}^{K}$, as defined by the objective in (6). The final solution $\mathcal{C}^{(K)}$ is regarded as the output of the HLGP framework and is evaluated using the feasible cost function $f(\cdot)$. This hierarchical approach offers the opportunity to progressively reduce compounded errors in partition solutions from $\mathcal{C}^{(0)}$ to $\mathcal{C}^{(K)}$ across levels. In the subsequent sections, we present methods based on either RL or SL to optimize the involved policies with respect to the HLGP objective defined in (6).

4.2 RL-driven HLGP

In the HLGP framework, the imperative task at hand involves the joint optimization of the global and local partition policies, as illustrated by the objective in (6). To address this intricate optimization problem through RL algorithms, a rigorous formulation based on a

multi-level Markov Decision Process (MDP) is required. However, (6) essentially revolves around evaluating the final partition solution $\mathcal{C}^{(K)}$ at the K-th level. For any partition solution $\mathcal{C}^{(k)}$ with $0 \leq k < K$, its corresponding evaluation depends not only on itself but also on the subsequent partition solutions $\mathcal{C}^{(k+1)}, \ldots, \mathcal{C}^{(K)}$. Thus, when a RL algorithm attempts to update both the global and local partition policies using samples associated with $\mathcal{C}^{(k)}$ for $0 \leq k < K$, the absence of direct evaluative feedback for these partition solutions primarily contributes to the instability concern during the joint training. To address this issue, we equivalently transform the HLGP objective defined in (6) into one that supports direct evaluation at each level, as described in Theorem 2.

Theorem 2 *Let $g(c_i) = \sum_{i=1}^{N_c} \mathbb{E}_{\tau_i \sim \pi^*_{\text{perm}}(\cdot|c_i)}[e(\tau_i)]$. It is evident that $f(\mathcal{C}) = \sum_{i=1}^{N_c} g(c_i)$ acts as a feasible cost function, since this definition of $f(\mathcal{C})$ directly aligns with the objective of the partition problem defined in (1). Then, the optimization problem defined in (6) can be transformed equivalently as follows:*

$$\begin{aligned} \min_{\pi_{\text{Gpart}}, \pi_{\text{Lpart}}} \quad & \mathbb{E}_{\mathcal{C}^{(0)} \sim \pi_{\text{Gpart}}}[f(\mathcal{C}^{(0)})] + \mathbb{E}_{\mathcal{C}^{(0)} \sim \pi_{\text{Gpart}}} \mathbb{E}_{\mathcal{C}^{(1)} \sim \pi_{\text{Lpart}}}[f(\mathcal{C}^{(1)}) - f(\mathcal{C}^{(0)})] + \\ & \cdots + \mathbb{E}_{\mathcal{C}^{(0)} \sim \pi_{\text{Gpart}}} \mathbb{E}_{\mathcal{C}^{(1)} \sim \pi_{\text{Lpart}}} \cdots \mathbb{E}_{\mathcal{C}^{(K)} \sim \pi_{\text{Lpart}}}[f(\mathcal{C}^{(K)}) - f(\mathcal{C}^{(K-1)})]. \end{aligned} \tag{7}$$

In (7), the evaluation of the partition solution $\mathcal{C}^{(k)}$ at level $k \geq 1$ is defined as $f(\mathcal{C}^{(k)}) - f(\mathcal{C}^{(k-1)})$. This expression can be further derived as:

$$\begin{aligned} f(\mathcal{C}^{(k)}) - f(\mathcal{C}^{(k-1)}) &= \textstyle\sum_{j=1}^{\lfloor \frac{N_c}{2} \rfloor} [h(\mathcal{C}^{(k)}, k, m) - h(\mathcal{C}^{(k-1)}, k, m)]; \\ h(\mathcal{C}, k, m) &= g(c_{(m+k-1) \bmod N_c+1}) + g(c_{(m+k) \bmod N_c+1}), \end{aligned} \tag{8}$$

where $m = 2(j-1)$, and the function $h(\mathcal{C}, k, m)$ serves as the evaluation metric for the subgraph pair $(c_{(m+k-1) \bmod N_c+1}, c_{(m+k) \bmod N_c+1})$ within the partition solution $\mathcal{C}$.

Please see Appendix D.2 for the proof of Theorem 2. Theorem 2 breaks down the HLGP objective defined in (6) into $K+1$ components, with each component associated with the direct evaluation of a specific partition solution. Notably, except for the evaluation of $\mathcal{C}^{(0)}$ which solely considers its own cost $f(\mathcal{C}^{(0)})$, the evaluation of $\mathcal{C}^{(k)}$ for each level $k \geq 1$ hinges on the difference between its own cost, $f(\mathcal{C}^{(k)})$, and the cost of the preceding level, $f(\mathcal{C}^{(k-1)})$. At each local partition level $k \geq 1$, the local partition policy is responsible for resolving each subproblem $I_j^{(k-1)}$, where $1 \leq j \leq \lfloor \frac{N_c}{2} \rfloor$. This involves updating each subgraph pair $(c^{(k-1)}_{(m+k-1) \bmod N_c+1}, c^{(k-1)}_{(m+k) \bmod N_c+1})$ to the corresponding pair $(c^{(k)}_{(m+k-1) \bmod N_c+1}, c^{(k)}_{(m+k) \bmod N_c+1})$. The evaluation difference between the updated subgraph pair and its original counterpart is computed as $h(\mathcal{C}^{(k)}, k, m) - h(\mathcal{C}^{(k-1)}, k, m)$. The cumulative sum of these differences constitutes the overall evaluation discrepancy between partition solutions, which underlies the derivation of (8). Given the optimization problem stated above, we present the formulation of a multi-level MDP to address the multi-level graph partition problem within the HLGP framework.

In the multi-level MDP framework, at the global partition level $k = 0$, at each time step $t \geq 1$, the state $x_t^{(0)} \in \mathbb{X}^{(0)}$ comprises the problem instance I and the partial partition solution $\mathcal{C}^{(0)}[0:t-1]$, where $\mathcal{C}^{(0)}[0] = \emptyset$. The initial state distribution $\mu^{(0)}$ is aligned with the distribution of problem instances, denoted by $p_I(\cdot)$. The action $u_t^{(0)} \in \mathbb{U}^{(0)}$ involves selecting a node denoted as $\mathcal{C}^{(0)}[t]$, from the set of unvisited customer nodes and the depot, subject to the capacity constraint. Let i_t index subgraphs such that at time step t, the agent

is constructing the i_t-th subgraph $c_{i_t}^{(0)}$. When a feasible customer node is selected, it is assigned to the current subgraph $c_{i_t}^{(0)}$. When the depot node is selected, the current subgraph is finalized and a new subgraph $c_{i_{t+1}}^{(0)}$ is initiated, after which the policy continues clustering the remaining unvisited customer nodes. To ensure that each cluster contains at least one customer node, the depot node cannot be selected consecutively. Thus, the transition function can be clearly described based on the above rules. If the subgraph $c_{i_t}^{(0)}$ is completed, then the reward $r_t^{(0)}$ is set as $-g(c_{i_t}^{(0)})$; otherwise, it remains at 0. The global partition policy, parameterized by θ_G, is thus specified as $\pi_{\theta_G}(u_t^{(0)}|x_t^{(0)})$, yielding the probability distribution over actions given the state $x_t^{(0)}$.

At each local partition level $k \geq 1$, the local partition policy is tasked with solving a sequence of subproblems $\{I_j^{(k-1)}\}_{j=1}^{\lfloor \frac{N_c}{2} \rfloor}$, which are derived from the preceding partition solution $\mathcal{C}^{(k-1)}$. In this context, we use j_t as an index for subproblems, indicating that the j_t-th subproblem denoted as $I_{j_t}^{(k-1)}$, is currently being addressed but remains incomplete at time step t. The state $x_t^{(k)} \in \mathbb{X}^{(k)}$ consists of the sequence of subproblems and the partial solution of the currently active subproblem $I_{j_t}^{(k-1)}$. The initial state distribution $\mu^{(k)}$ corresponds to the distribution over subproblem sequences. The action $u_t^{(k)} \in \mathbb{U}^{(k)}$ involves selecting a feasible node for solving $I_{j_t}^{(k-1)}$. When solving the subproblem $I_{j_t}^{(k-1)}$, the transition function is defined in the same manner as that used in the global partition level. Once the subproblem $I_{j_t}^{(k-1)}$ is solved, the index j_t advances to the next subproblem $I_{j_{t+1}}^{(k-1)}$. The partial solution in the state is then reset to empty for the this subproblem. The reward $r_t^{(k)}$ is assigned as $-(h(\mathcal{C}^{(k)}, k, m) - h(\mathcal{C}^{(k-1)}, k, m))$, where $m = 2(j_t - 1)$, if the subproblem $I_{j_t}^{(k-1)}$ is successfully solved, otherwise, the reward remains at 0. Accordingly, the local partition policy parameterized by θ_L, is defined as $\pi_{\theta_L}(u_t^{(k)}|x_t^{(k)})$. Let $T^{(k)}$ and $\omega^{(k)}$ denote the horizon and the trajectory at level *k*, respectively. The objective of the multi-level MDP framework is to maximize the sum of expected returns across all levels, as defined below:

$$J(\theta_G, \theta_L) = \mathbb{E}_{\omega^{(0)}}[\sum_{t=1}^{T^{(0)}} r_t^{(0)}] + \cdots + \mathbb{E}_{\omega^{(0)}} \cdots \mathbb{E}_{\omega^{(K)}}[\sum_{t=1}^{T^{(K)}} r_t^{(K)}], \tag{9}$$

Proposition 1 *At any partition level $k \geq 0$, the definitions of the state and action within the multi-level MDP framework are sufficient to ensure the preservation of the Markovian property at that level.*

The proof of Proposition 1 is included in Appendix D.4. Notably, at each level $k < K$, although (9) isolates the evaluation exclusively for the trajectory $\omega^{(k)}$, the influence of subsequent trajectories $\omega^{(k+1)}, ..., \omega^{(K)}$ on the evaluation still remains. This implies that the underlying MDP at level *k* remains nonstationary. Furthermore, in (9), the sampling of the trajectory $\omega^{(k)}$ for $k \geq 1$ is controlled by the initial state distribution $\mu^{(k)}$ and the local partition policy π_{θ_L}. However, the distribution $\mu^{(k)}$ heavily depends on the preceding trajectories $\omega^{(0)}, \ldots, \omega^{(k-1)}$, which are themselves determined by both the global and local partition policies. To preserve a stationary underlying MDP at each level, we introduce a

surrogate initial state distribution $\hat{\mu}^{(k)}$, which achieves this by eliminating the dependency of the initial state distribution on prior trajectories. We thus take the following optimization problem as an approximation:

$$\begin{aligned} \hat{J}(\theta_G, \theta_L) &= L(\theta_G, \lambda_G, 0) + \textstyle\sum_{k=1}^{K} L(\theta_L, \lambda_L, k); \\ L(\theta, \lambda, k) &= \mathbb{E}_{\omega^{(k)} \sim \hat{\mu}^{(k)}, \pi_\theta} [\textstyle\sum_{t=1}^{T^{(k)}} r_t^{(k)}] + \lambda \mathcal{H}(\pi_\theta), \end{aligned} \tag{10}$$

where $\hat{\mu}^{(k)}$ denotes the surrogate initial state distribution at level k, $\mathcal{H}(\pi_\theta)$ is the entropy term that encourages exploration, and λ represents a hyperparameter that balances the entropy regularization. The entropy term is typically defined to minimize the KL divergence between the policy and a uniform distribution. Please note that $\hat{\mu}^{(0)} = \mu^{(0)}$. In the approximation objective defined in (10), at each level $0 \leq k \leq K$, the trajectory $\omega^{(k)}$ is evaluated solely based on its own cumulative rewards, without being influenced by the evaluations of subsequent trajectories. The surrogate initial state distribution $\hat{\mu}^{(k)}$ is activated only during gradient backpropagation, where it severs the dependency between the trajectory $\omega^{(k)}$ and preceding trajectories. As a result, the optimization for π_{θ_G} and π_{θ_L} is decoupled.

In the context of RL-driven HLGP, we incorporate the surrogate objective defined in (10) into the REINFORCE algorithm [48] to update π_{θ_G} and π_{θ_L}. In each iteration $n \geq 0$ of REINFORCE, the current global partition policy, denoted as $\pi_{\theta_G^n}$, is used to sample the trajectory $\omega^{(0)}$, which is then used to update $\pi_{\theta_G^n}$. At each local partition level $k \geq 1$, the current local partition policy, denoted as $\pi_{\theta_L^n}$, is additionally leveraged to sample the partition solution $\mathcal{C}^{(k-1)}$, which is essential for constructing the surrogate initial state distribution $\hat{\mu}^{(k)}$. Subsequently, the trajectory $\omega^{(k)}$ is sampled and used to update $\pi_{\theta_L^n}$. The pseudocode is provided in Algorithm 1 of Appendix C.

Furthermore, in the standard theoretical analysis of REINFORCE algorithm presented in [49], the upper bound of regret includes the term $||\frac{d}{\mu}||_\infty$, where d and μ denote the stationary state distribution and the initial state distribution, respectively. However, existing methods using REINFORCE algorithm for whatever types of problems (permutation or partition) in the context of CVRP ignore the potential risks highlighted in the regret bound. We exemplify the partition problem as a case study to elucidate this issue. The support set of μ consists solely of the problem instances I. During the partition process, at each step $t > 1$, let c_{i_t} denote the i_t-th subgraph currently under construction. The partition policy is indeed responsible for solving a subproblem, denoted as $I(t) = (G(t), D(t), N_{\max}(t))$. The graph $G(t)$ consists of the depot and the unvisited customers. The vehicle capacity $D(t)$ is calculated by subtracting the total demand of the already visited nodes in c_{i_t}, from the initial capacity D, and it reverts back to D once c_{i_t} is fully formed. The number of allowable returns to the depot, $N_{\max}(t)$, is decremented in accordance with the number of completed subgraphs. In contrast, the support set of d thus includes the subproblems $I(t)$. This substantial discrepancy between support sets inevitably results in an unbounded value of $||\frac{d}{\mu}||_\infty$ in the regret bound. This insight inspires us to incorporate the subproblems $I(t)$, encountered during the partition process, into the training of involved partition policies to reduce the mismatch between support sets.

In the practical implementation, a problem instance I is initially generated from the instance distribution $p_I(\cdot)$ and used to train the policy π_{θ_G} via RL. At each time step t, if a new subgraph c_{i_t} is formed, then the resulting subproblem $I(t)$ is treated as an individual problem instance, denoted as $I \leftarrow I(t)$, for the training of π_{θ_G}. This procedure continues

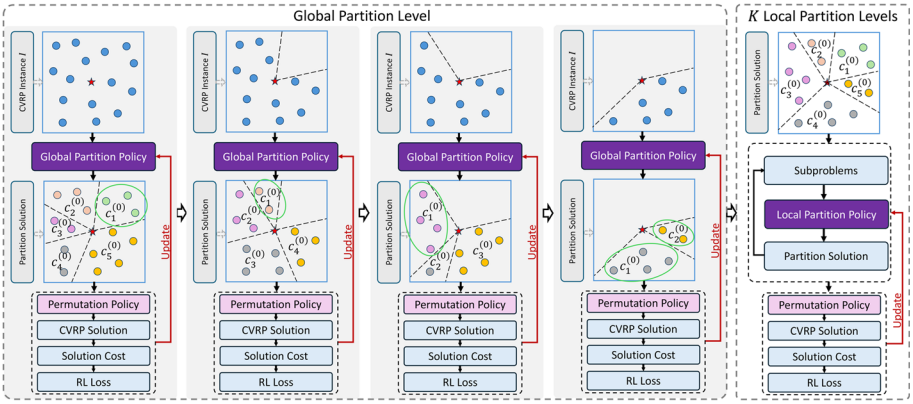

Fig. 2 This figure illustrates an example of the RL-driven HLGP framework. In this example, the global partition process unfolds over four stages. Initially, a problem instance is fed to the global partition policy to sample a partition solution. The RL loss is then computed with the aid of a pretrained permutation policy. During this process, once the first subgraph $c_1^{(0)}$ is completed, the remaining nodes form a new subproblem. This subproblem is treated as an individual problem instance for further training of the global partition policy. The complete partition solution of this subproblem replaces the partial solution from the previous stage. This iterative procedure continues until a new complete partition solution is constructed, retaining only the subgraphs enclosed within the green circles. This final partition solution is subsequently refined through K local partition levels

iteratively until $G = \emptyset$ in the current instance I, after which a new instance I is drawn from $p_I(\cdot)$. For efficiency, we do not use all subproblems, but instead include only those that emerge when a new subgraph is completed in the current problem instance I. During inference, the partition solution $\mathcal{C}^{(0)}$ is formed by sequentially replacing the partial partition solution with the complete partition solution of the corresponding subproblem. An example is shown in Fig. 2. The training and inference procedures that leverage encountered subproblems can be similarly applied to π_{θ_L}. Additionally, we adopt the isomorphic Graph Neural Network (GNN) architecture used in GLOP [31] as the backbone for both π_{θ_G} and π_{θ_L}.

4.3 SL-driven HLGP

In this section, we pivot towards an SL training strategy to optimize the objective of the partition problem within the HLGP framework, as defined in (6). Under this paradigm, the optimal partition policy π^*_{part} is presumed to be available in advance. Consequently, the optimal partition solution $\bar{\mathcal{C}} = \{\bar{c}_1, \ldots, \bar{c}_{N_c}\}$ for each instance I is accordingly obtained from π^*_{part}. We therefore define the feasible cost function as $f(\mathcal{C}) = -\mathbf{1}(\mathcal{C} = \bar{\mathcal{C}}) = -\mathbf{1}(c_1 = \bar{c}_1, \ldots, c_{N_c} = \bar{c}_{N_c})$, where $\mathbf{1}(\cdot)$ denotes the indicator function. This definition of the feasible cost function is consistent with Definition 1, as the objective $\mathbb{E}_{\mathcal{C}}[f(\mathcal{C})]$ is minimized only when the optimal partition solution is obtained. Recall that the optimal solution $\bar{\mathcal{T}}$ of a CVRP instance can be equivalently seen as the optimal partition solution $\bar{\mathcal{C}}$ when disregarding the node permutation information in $\bar{\mathcal{T}}$. Accordingly, by setting $\bar{\mathcal{C}} = \bar{\mathcal{T}}$, the feasible cost function can be defined as $f(\mathcal{C}) = -\mathbf{1}(\mathcal{C}[0] = \bar{\mathcal{C}}[0], \ldots, \mathcal{C}[N_{\text{sol}}] = \bar{\mathcal{C}}[N_{\text{sol}}])$. Upon utilizing this design of $f(\mathcal{C})$, Theorem 3 simplifies the optimization objective of the HLGP framework.

Theorem 3 *Given the feasible cost function $f(\mathcal{C}) = -\mathbf{1}(\mathcal{C} = \bar{\mathcal{C}})$, the optimization objective of the HLGP framework for a problem instance I is to identify the policies π_{θ_G} and π_{θ_L} so as to minimize:*

$$L(\theta_G, \theta_L, \bar{\mathcal{C}}) = -\log \pi_{\theta_G}(\bar{\mathcal{C}}|I) - \sum_{i=1}^{N_c} \log \pi_{\theta_L}(\bar{\mathcal{C}}_i|I_i), \tag{11}$$

where $I_i = (G_i, D, 2)$ denotes the subproblem with $G_i = \bar{c}_i \cup \bar{c}_{i \bmod N_c+1}$, $\bar{\mathcal{C}}_i = \{\bar{c}_i, \bar{c}_{i \bmod N_c+1}\}$ represents the corresponding label, and $1 \leq i \leq N_c$.

Please see Appendix D.3 for the proof of Theorem 3. We can observe that the optimization objectives for π_{θ_G} and π_{θ_L} are totally disentangled, as established in Theorem 3. Accordingly, for instances sampled from $p_I(\cdot)$, the optimization objective is to minimize:

$$J(\theta_G, \theta_L) = \mathbb{E}_{(I,\bar{\mathcal{C}})\sim p_I, \pi^*_{\text{part}}}[L(\theta_G, \theta_L, \bar{\mathcal{C}})], \tag{12}$$

where the problem instance I and its corresponding label $\bar{\mathcal{C}}$ are sampled from the instance distribution $p_I(\cdot)$ and the oracle partition policy π^*_{part}, respectively. As a result, the performance of π_{θ_G} and π_{θ_L} is evaluated on the sampled trajectories induced by π^*_{part}. However, since the oracle partition policy π^*_{part} is practically unavailable for the NP-hard partition problem, it is infeasible to directly get supervised labels. Inspired by recent techniques known as self-imitation learning [40, 41], our goal is to acquire high-quality labels for training instances by constructing a reasonably strong behavioral policy $\hat{\pi}_{\text{part}}$. The behavioral policy is constructed using beam search guided by the partition policy. Beam search, originally utilized for test-time improvement in COPs [11, 50], is repurposed to generate high-quality labels for supervised training. This design leverages the fact that, compared to greedy rollout, beam search can produce superior solutions, albeit at the cost of computational efficiency. The working pipeline of $\hat{\pi}_{\text{part}}$ can be described as follows (see Fig. 3): At the global partition level, the initial partition solution $\bar{\mathcal{C}}^{(0)}$ is generated using beam search guided by the policy π_{θ_G}. Subsequently, for each local partition level $k \geq 1$, the corresponding partition solution $\bar{\mathcal{C}}^{(k-1)}$ is further refined locally using beam search with the policy π_{θ_L}. This process continues until the final partition solution $\bar{\mathcal{C}}^{(K)}$ is obtained. Thus, during training, the final partition solution $\bar{\mathcal{C}} = \bar{\mathcal{C}}^{(K)}$ serves as the supervisory label. The resulting practical loss function is thus defined as:

$$\hat{J}(\theta_G, \theta_L) = \mathbb{E}_{(I,\bar{\mathcal{C}})\sim p_I, \hat{\pi}_{\text{part}}}[L(\theta_G, \theta_L, \bar{\mathcal{C}})] + \text{reg}(\theta_G, \theta_L), \tag{13}$$

where $\text{reg}(\theta_G, \theta_L) = \lambda_G \frac{||\theta_G||^2}{2} + \lambda_L \frac{||\theta_L||^2}{2}$ corresponds to the L2 regularization term, with hyperparameters λ_G and λ_L. Therefore, this loss function is incorporated into the imitation learning algorithm to iteratively optimize the policies π_{θ_G} and π_{θ_L}. At each iteration $n \geq 0$, the algorithm deploys the behavioral policy $\hat{\pi}^n_{\text{part}}$, which relies on θ^n_G and θ^n_L, to collect labeled instances $(I, \bar{\mathcal{C}})$. Online gradient updates are then executed based on estimated gradients to update θ^n_G and θ^n_L. The detailed pseudocode is provided in Algorithm 2 of Appendix C.

Here, we delve deeper to illustrate the training process for the global partition policy π_{θ_G} as a case study to elucidate the intricacies of the SL algorithm for the partition problem. A similar analysis can be conducted for the local partition policy π_{θ_L}. The global partition policy π_{θ_G} requires to imitate the complete trajectory induced by the behavioral policy $\hat{\pi}_{\text{part}}$. Following the formulation in (4), the global partition policy can directly

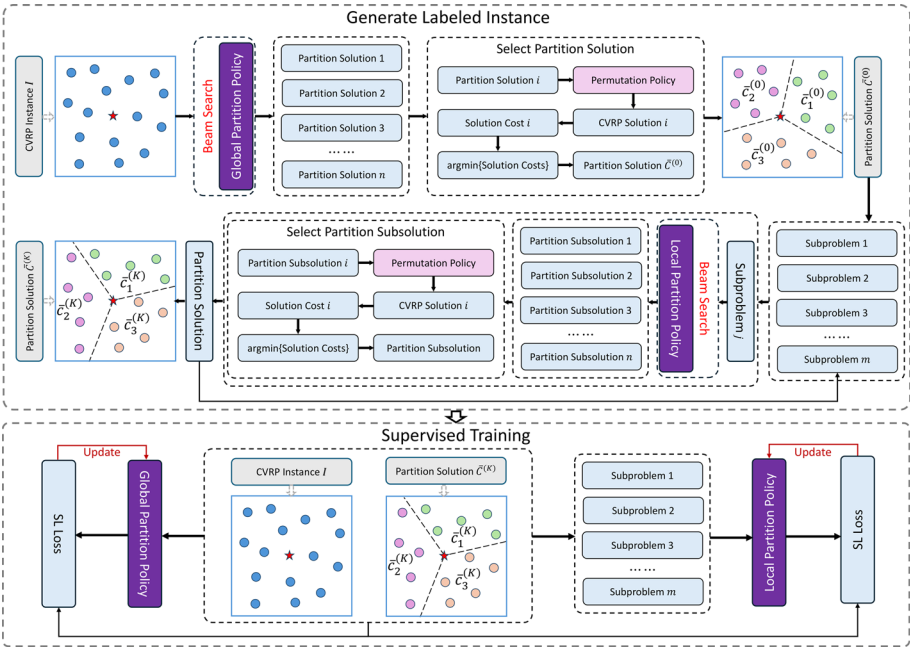

Fig. 3 This figure illustrates the working pipeline of the SL-driven HLGP framework. At the global partition level, a set of candidate partition solutions is generated via beam search, guided by the global partition policy. Each candidate is then evaluated based on the cost of the resulting CVRP solution, computed with the aid of a pretrained permutation policy. Among these, the partition solution with the lowest cost is selected and denoted as $\bar{\mathcal{C}}^{(0)}$. At each local partition level, the partition solution $\bar{\mathcal{C}}^{(k-1)}$ (starting with $k = 1$) is used to construct a set of subproblems. Each subproblem is independently solved using beam search guided by the local partition policy, and the best subsolution is selected accordingly. These subsolutions for the subproblems are then aggregated to form an updated partition solution $\bar{\mathcal{C}}^{(k)}$. This procedure is repeated for K local partition levels, ultimately yielding the partition solution $\bar{\mathcal{C}}^{(K)}$, which serves as the supervision label for training both the global and local partition policies on the original problem instance I

output the probability distribution over node sequences. Subsequently, the log-probability of the node sequence corresponding to $\bar{\mathcal{C}}$ is maximized to update θ_G. This log-probability of the node sequence can be represented as a sum of conditional log-probabilities: $\log \prod_{t=1}^{N_{\text{sol}}} \pi_{\theta_G}(\bar{\mathcal{C}}[t]|\bar{\mathcal{C}}[0:t-1], I) = \sum_{t=1}^{N_{\text{sol}}} \log \pi_{\theta_G}(\bar{\mathcal{C}}[t]|\bar{\mathcal{C}}[0:t-1], I)$. This formulation naturally prompts us to consider $(\bar{\mathcal{C}}[0:t-1], I)$ as an individual training sample, with its corresponding label being the next node $\bar{\mathcal{C}}[t]$. In this context, at each time step $t \geq 1$, the global partition policy addresses a subproblem $I(t)$, which is defined based on the partial partition solution $\bar{\mathcal{C}}[0:t-1]$ and the original problem instance I. This definition is consistent with the notion of subproblems used in the RL setting. Hence, instead of generating the entire node sequence for behavioral imitation, the labeled instance $(I(t), \bar{\mathcal{C}}[t])$ is fed to the global partition policy to imitate one-step behavior at each time step $t \geq 1$. In practice, we adopt a variant Transformer model used in BQ [38] as the backbone for π_{θ_G} and π_{θ_L}, which aligns with the analysis above. Therefore, we underscore the importance of viewing the subproblems encountered during training as individual training instances within both the contexts of RL and SL training paradigms.

4.4 Time complexity analysis

Let N_v denote the number of customer nodes, N_c the number of subgraphs at each level, and N_v/N_c the average number of nodes per subgraph. In the RL-driven HLGP framework, during the global partition process, the first generated subgraph is retained, while the remaining nodes form a new individual problem instance. This problem instance is then solved again by the global partition policy. The procedure continues iteratively until no nodes remain. In addition, each encountered subproblem is evaluated using the local permutation policy. Since the nodes within each subgraph are permuted in parallel, the overall time complexity of the global partition process is $O(((3 + N_c)/2)N_v)$. At each local partition level, every subproblem merges two neighboring subgraphs. These merged subproblems are then independently divided by the local partition policy and the resulting partition solutions are evaluated by the local permutation policy. The resulting time complexity at each local partition level is $O(3N_v/2)$. Therefore, with K local partition levels, the total time complexity of the RL-driven HLGP is $O(((3 + N_c + 3K)/2)N_v)$.

Let N_{cand} denote the number of candidate solutions maintained by the beam search in the SL-driven HLGP framework. During the global partition process, at each decoding step, the beam search selects the N_{cand} partial partition solutions with the highest probabilities. Therefore, generating N_{cand} complete partition solutions under the guidance of the global partition policy incurs a time complexity of $O(N_{\text{cand}}N_v)$. Each resulting partition solution is then independently evaluated using the local permutation policy. Consequently, the overall time complexity of the global partition stage is $O((1 + 1/N_c)N_{\text{cand}}N_v)$. At each local partition level, beam search is again applied using the local partition policy to solve each subproblem. Furthermore, every partition solution generated for a subproblem is evaluated by the local permutation policy. This leads to a time complexity of $O(3N_{\text{cand}}N_v/2)$. Therefore, with K local partition levels, the total time complexity of the SL-driven HLGP framework is $O((1 + 1/N_c + 3K/2)N_{\text{cand}}N_v)$.

5 Experiments

In this section, we detail the training and evaluation procedures for the proposed HLGP methods, including the problem settings, hyperparameter configurations, and practical training protocols. Following this, we present performance comparisons against baseline methods, along with a hyperparameter study and an ablation study to examine the impact of key components and parameters within the HLGP framework. Additionally, visualization results are provided in Appendix B.

5.1 Training settings

During the training phase for both the RL-driven and SL-driven HLGP frameworks, we strictly adhere to the problem setting used in GLOP [31]. Under this setting, each CVRP instance consists of 1,000 customer nodes uniformly distributed within a unit square area. The vehicle capacity, denoted by D, is fixed at 200. The maximum permissible number of returns to the depot by vehicles is accordingly calculated as $\lceil \sum_{i=1}^{N_v} d_i/D \rceil + 1$, where N_v represents the number of customer nodes and d_i signifies the demand associated with each customer node. Each demand d_i is independently drawn from the discrete uniform distribution over the set $\{1, \ldots, 9\}$.

The RL-driven HLGP framework adopts the same isomorphic GNN architecture [44] as employed in GLOP [31] to implement both the global and local partition policies. For each CVRP (sub-)problem instance, a corresponding partition heatmap is generated by the associated partition policy, which further aids in progressively decoding a feasible partition (sub-)solution that complies with the predefined CVRP constraints. Subsequently, the pretrained local permutation policy from GLOP [31] is applied to reorder the nodes within each subgraph of the feasible partition (sub-)solution. These resulting CVRP subtours collectively form a CVRP (sub-)solution. Accordingly, the cost of the CVRP (sub-)solution directly reflects the evaluation of the corresponding feasible partition (sub-)solution.

The training phase of the RL-driven HLGP spans 20 epochs, each consisting of 256 iterations. In each iteration, 5 CVRP instances are randomly generated to train both the global and local partition policies. For a detailed description of the training procedure, please refer to Algorithm 1 in Appendix C. Following the training configurations in GLOP [31], 20 solutions are simultaneously generated for both the global and local partition policies to calculate their respective baselines, thereby reducing gradient variance. The Adam optimizer is employed with an initial learning rate of 3×10^{-4}, and a cosine annealing scheduler is used to gradually decay the learning rate over the course of training. Additionally, gradient clipping is applied to ensure the maximum L2 norm of gradients does not exceed 1. The RL-driven HLGP is trained on an NVIDIA RTX 3090 GPU and an INTEL(R) XEON(R) GOLD 5218R CPU@2.10GHz.

The SL-driven HLGP framework leverages the same network architecture, a variant of the Transformer model, as used in BQ [38] to implement both the global and local partition policies, which incrementally generate partition solutions. Although BQ [38] was originally designed to serve as a neural solver for producing CVRP solutions directly, it can be repurposed to generate partition solutions. This adaptability arises from the fact that a partition solution can be represented as a node list, irrespective of the specific decoding order. This justifies that any neural solver crafted for generating CVRP solutions can be repurposed as the partition policy. Subsequently, the same pretrained local permutation policy utilized in the RL-driven HLGP is applied to solve the resulting node ordering problems. As in the RL-driven case, the cost of the final CVRP solution corresponds to the evaluation of the generated partition solution.

The training phase of the SL-driven HLGP framework consists of two stages. Notably, since both the global and local partition policies are initialized randomly, incorporating beam search within our proposed multi-level HL framework to generate partition solutions as supervision signals for large-scale instances does not guarantee sufficiently high-quality labels for effective training. Additionally, there are no optimal solvers specifically tailored for the partition task in the context of CVRP. However, as previously mentioned, a CVRP solution can be regarded as a partition solution, regardless of the node ordering, implying that the optimal CVRP solution is equivalent to the optimal partition solution. Thus, in the first stage, we employ a curriculum learning strategy to train both the global and local partition policies on CVRP instances, each comprising 100 nodes. The labels for these small-scale instances are generated using the procedure employed in BQ [38], and we follow the training configurations outlined in BQ [38] during this stage. In the second stage, we train the partition policies on CVRP instances of 1,000 nodes. A detailed description of the training procedure in this stage is included in Algorithm 2 of Appendix C. This stage comprises 20 epochs, each containing 500 iterations. Within each epoch, 100 problem instances of 1000 nodes are randomly sampled. The corresponding labels are generated by executing the involved policies using beam search at each level of the proposed multi-level

HL framework (as illustrated in Fig. 3), with a beam size of 16. These labeled instances form the training dataset. During each iteration, mini-batches of data are sampled from this training dataset to train the global partition policy, with the batch size fixed at 50. Additionally, we employ the Adam optimizer with an initial learning rate of 1×10^{-5}. The learning rate decays by a factor of 0.9 every 5 update steps. We constrain the maximum L2 norm of gradients to be less than 1. The coefficients for the L2 regularization terms in the loss function are set to $\lambda_G = \lambda_L = 1 \times 10^{-6}$, which are consistent with standard practices in Transformer-based neural solvers [15, 16, 34, 42]. The SL-driven HLGP model is trained on 4 NVIDIA RTX 3090 GPUs and an INTEL(R) XEON(R) GOLD 5218R CPU@2.10GHz.

5.2 Evaluation settings

During the evaluation phase, our focus is on comparing the generalization capabilities of our proposed models against prior baseline methods. To this end, we utilize diverse datasets that vary in both scale and node distribution. Specifically, each evaluation dataset contains 1,000 (1K), 2,000 (2K), 5,000 (5K), 7,000 (7K), or 1,0000 (10K) customer nodes. These datasets have been widely used to investigate generalization performance across varying scales [31, 35, 38, 39]. The distribution of customer nodes follows one of four patterns: Uniform (U), Gaussian (G), explosion (E) or rotation (R), which are commonly adopted to evaluate model generalization performance across diverse spatial distributions [42, 45]. With the exception of the dataset containing 1,000 customer nodes, for which the vehicle capacity is set to 200, all other datasets use a capacity of 300. Each dataset consists of 128 problem instances. The generation of these instances follows the methodologies outlined in [13, 42]. In addition, for comparative evaluation, we report three metrics: the average cost over the dataset (Avg.), the standard deviation of solution costs (Std.), and the average inference time required to solve a single problem instance within the dataset (Time). For both the RL-driven and SL-driven HLGP frameworks, the number of levels, denoted as K is set as 5. The beam size in SL-driven HLGP during evaluation is set as 16, 16, 8, 4, and 4 for the CVRP datasets with node counts of 1,000, 2,000, 5,000, 7,000, and 10,000, respectively. Please note that the implementation code is publicly available.

During the evaluation phase, we compare our proposed RL-driven and SL-driven HLGP models against a broad range of existing methods. Classic heuristic methods include OR-Tools [51], LKH3 [9] and HGS [10]. Among learning-based constructive methods, AM [13], POMO [15], and Sym-POMO [16] serve as baselines trained using RL. AMDKD [45] and Omni-POMO [42] specifically target generalization issues. ELG-POMO [34] and INViT [35] harness local topological features. BQ [38] and LEHD [39] are trained using SL to improve their generalization capabilities. Within the realm of iterative methods, L2I [17], NLNS [19], and DACT [20] integrate RL-based policies with local operators to iteratively refine solutions. In the divide-and-conquer paradigm, RBG [27] and L2D [26] employ heuristic partition rules, while TAM [30] and GLOP [31] utilize neural partition policies. We follow the official implementations of these methods and the instructions provided in prior works [30, 31, 35] that apply these methods to replicate the experimental results. For RBG and TAM, we directly use the reported results in previous studies [27, 30, 31]. To maintain a fair comparison, we only include those baseline methods that either have official code available for reproducibility or have been extensively benchmarked in the literature. Further implementation details of these baseline methods are deferred to Appendix A.

5.3 Performance comparisons

In Table 1, we compare our proposed RL-driven and SL-driven HLGP models with a variety of existing methods on cross-size datasets. The RL-driven HLGP model consistently outperforms its baseline, GLOP-G, by achieving lower average solution costs across datasets of varying problem scales. When compared to the state-of-the-art learning-based method, LEHD, the RL-driven HLGP shows only a slight performance drop, notably on the CVRP1K dataset. Across other cross-size datasets, the RL-driven HLGP model consistently delivers superior solutions and is notably more efficient than LEHD. Moreover, the standard deviation of its solution costs is closer to that of the optimal heuristic method, HGS, indicating enhanced stability and robustness compared to its baseline. Notably, on the large-scale evaluation setting with 10,000 nodes (i.e., CVRP10K), the RL-driven HLGP model achieves a improvement of 10.62% in terms of the average cost compared to its baseline. The SL-driven HLGP model also demonstrates consistent improvements over its baseline, BQ, across datasets of varying scales, achieving lower average solution costs. Notably, on the in-scale dataset (i.e., CVRP1K), the SL-driven HLGP model not only outperforms prior learning-based methods but also surpasses the optimal heuristic method, HGS, while achieving a lower average inference time of 8.31s per instance. For large-scale problem instances ranging from 2,000 to 10,000 customer nodes, the SL-driven HLGP model exhibits only marginal performance degradation compared to HGS, yet maintains a substantially lower average inference time. In comparison to all other learning-based baselines, the SL-driven HLGP model consistently demonstrates its superiority in terms of average cost while maintaining efficiency. In terms of the standard deviation, the SL-driven HLGP model achieves lower values on CVRP5K, CVRP7K and CVRP10K, and slightly higher values on CVRP1K and CVRP 2K. On the CVRP10K dataset, the SL-driven HLGP model achieves a 14.57% improvement in average cost compared to its baseline. Moreover, it stands out as the only method capable of generating high-quality solutions within 4 minutes per instance on this large-scale setting.

Table 2 displays a comparison of our proposed methods with various existing methods on both cross-distribution datasets (i.e., CVRP1K+G, CVRP1K+E and CVRP1K+R) and cross-size-and-distribution datasets (i.e., CVRP2K+G, CVRP5K+E and CVRP7K+R). On cross-distribution datasets, the RL-driven HLGP model demonstrates the consistent improvement over its baseline method, GLOP-G, in terms of the average solution cost, while achieving a standard deviation closer to that of the optimal heuristic method, HGS. When compared to BQ and LEHD, the RL-driven HLGP model exhibits a minor performance decline on cross-distribution datasets, yet benefits from the lower average inference time. In contrast, on the cross-size-and-distribution datasets, the RL-driven HLGP model consistently showcases superior generalization by efficiently producing improved solutions compared to prior learning-based methods. Compared to all previous learning-based methods, the SL-driven HLGP model consistently outperforms across both cross-size and cross-size-and-distribution datasets. Additionally, the standard deviation of its solution costs is significantly closer to that of the optimal heuristic method, HGS, than those of other learning-based approaches that perform well across diverse datasets, such as BQ, LEHD, and L2D. Moreover, the SL-driven HLGP model achieves performance levels comparable to HGS while offering significantly greater computational efficiency.

Furthermore, we observe that, compared with LKH3, except for slight performance decreases of 0.03 and 0.25 on CVRP1K+G and CVRP2K+G, respectively, the SL-driven

Table 1 Comparative results on uniformly distributed CVRP instances. OOM stands for out-of-memory. The symbol∗indicates that the results are obtained from the original papers. The notation↓indicates that a lower value is better

Methods	CVRP1K			CVRP2K			CVRP5K			CVRP7K			CVRP10K		
	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓
OR-Tools	47.19	3.89	90.00s	64.71	5.03	3.00m	136.05	12.95	4.00m	180.70	18.20	5.00m	245.55	24.64	6.00m
LKH3	42.17	0.80	14.18s	58.06	1.06	31.92s	126.59	2.81	6.80m	172.80	4.04	18.21m	240.23	5.42	41.29m
HGS	41.12	0.77	4.57m	56.24	1.07	12.68m	122.84	2.87	18.80m	165.37	3.95	20.15m	226.59	5.29	24.64m
AM (ICLR'19)	59.18	2.81	8.84s		OOM			OOM			OOM			OOM	
POMO (NeurIPS'20)	100.99	7.43	4.63s	255.13	30.02	39.35s		OOM			OOM			OOM	
Sym-POMO (NeurIPS'22)	98.04	2.86	5.71s	157.89	2.96	45.23s		OOM			OOM			OOM	
AMDKD (NeurIPS'22)	84.16	0.98	4.27s	188.75	4.39	34.39s		OOM			OOM			OOM	
Omni-POMO (ICML'23)	47.80	0.77	4.45s	74.01	1.05	35.86s		OOM			OOM			OOM	
ELG-POMO (IJCAI'24)	46.41	0.40	9.53s	66.07	0.55	67.19s		OOM			OOM			OOM	
INViT (ICML'24)	46.56	0.81	17.08s	64.94	1.09	36.67s	139.45	2.86	141.07s	186.57	3.93	4.81m	254.17	5.39	6.96m
LEHD (NeurIPS'23)	42.80	0.82	40.25s	60.48	1.12	72.48s	136.80	2.86	3.22m	188.11	4.00	6.52m	266.06	5.56	11.92m
BQ (NeurIPS'23)	43.12	0.80	4.75s	60.95	1.10	15.66s	137.14	3.00	79.89s	188.78	4.23	1.88m	265.81	5.97	3.30m
L2I (ICLR'20)	49.79	1.10	18.60s	95.58	5.44	44.88s	262.84	9.99	2.64m	506.73	25.25	3.61m	1263.23	4.00	4.07m
NLNS (ECAI'20)	53.51	0.83	12.08s	81.54	1.12	12.15s	180.84	2.99	12.62s	243.50	4.17	13.16s	331.27	5.53	13.97s
DACT (NeurIPS'21)	50.43	0.35	75.47s	71.17	0.51	5.40m		OOM			OOM			OOM	
L2D (NeurIPS'21)	46.45	0.87	4.67s	64.04	1.21	7.54s	135.09	3.02	16.11s	182.21	4.13	24.37s	246.38	5.55	27.59s
RBG* (KDD'22)	74.00	-	13.00s	137.60	-	42.00s		-			-			-	
TAM-AM* (ICLR'23)	50.06	0.98	0.76s	74.31	1.42	2.20s	172.22	-	11.78s	233.44	-	26.47s		-	
TAM-LKH* (ICLR'23)	46.34	0.84	1.82s	64.78	1.18	5.63s	144.64	-	17.19s	196.91	-	33.21s		-	
GLOP-G (AAAI'24)	47.21	0.90	0.73s	63.60	1.41	1.74s	141.67	3.67	2.37s	191.75	4.99	3.50s	266.96	6.46	13.74s
GLOP-LKH (AAAI'24)	46.03	0.99	0.78s	63.10	1.43	1.83s	140.51	3.72	4.31s	191.45	5.06	7.34s	267.50	6.50	16.47s
RL-driven HLGP	43.78	0.85	3.72s	59.58	1.17	10.03s	128.12	3.06	82.59s	173.71	4.39	1.96m	238.62	6.03	5.13m
SL-driven HLGP	**41.95**	0.78	8.31s	**57.67**	1.08	32.40s	**124.13**	2.79	97.27s	**166.73**	3.91	2.15m	**227.07**	5.25	3.39m
SL-driven HLGP-LKH	41.93	0.78	8.67s	57.52	1.08	33.11s	123.95	2.79	71.94s	166.49	3.90	86.66s	226.80	5.22	3.71m

Table 2 Comparative results on various distributed CVRP instances. "G" denotes the Gaussian distribution. "E" denotes the Explosion distribution. "R" denotes the Rotation distribution. The notation↓indicates that a lower value is better

Methods	CVRP1K+G			CVRP1K+E			CVRP1K+R			CVRP2K+G			CVRP5K+E			CVRP7K+R		
	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓	Avg.↓	Std.↓	Time↓
OR-Tools	35.30	6.01	90.00s	41.24	7.24	90.00s	41.31	6.42	90.00s	45.98	8.19	3.00m	107.70	21.46	4.00m	161.10	32.75	5.00m
LKH3	32.52	1.21	37.35s	38.01	1.48	15.55s	37.50	1.33	15.35s	42.60	1.62	64.06s	103.45	4.39	6.67m	156.04	6.81	25.69m
HGS	31.84	1.19	15.57m	37.13	1.46	6.52m	36.62	1.32	7.78m	41.64	1.61	19.80m	101.16	4.40	16.53m	151.04	6.72	21.04m
AM (ICLR'19)	93.62	20.23	9.32s	58.99	4.79	8.74s	60.80	5.42	8.72s		OOM			OOM			OOM	
POMO (NeurIPS'20)	56.83	2.72	4.78s	74.88	4.84	4.54s	75.26	5.52	4.41s	101.75	7.32	38.31s		OOM			OOM	
Sym-POMO (NeurIPS'22)	97.59	5.35	5.57s	95.08	5.11	5.65s	106.88	6.11	5.53s	134.32	5.20	40.56s		OOM			OOM	
AMDKD (NeurIPS'22)	58.71	1.98	4.14s	71.10	2.04	4.17s	73.32	2.00	4.11s	108.11	3.82	32.96s		OOM			OOM	
Omni-POMO (ICML'23)	35.47	1.20	4.30s	41.80	1.47	4.30s	41.30	1.34	4.28s	51.02	1.76	35.31s		OOM			OOM	
ELG-POMO (IJCAI'24)	36.49	0.63	9.86s	41.64	0.75	9.67s	41.31	0.69	9.48s	49.34	0.86	68.73s		OOM			OOM	
INViT (ICML'24)	35.67	1.28	19.80s	42.11	1.53	19.80s	41.22	1.37	19.74s	47.31	1.75	46.92s	113.26	4.79	3.17m	169.38	7.47	4.66m
LEHD (NeurIPS'23)	33.99	1.23	36.27s	38.96	1.50	36.19s	38.44	1.36	36.16s	47.48	1.65	64.80s	116.70	4.38	2.85m	176.14	6.91	5.93m
BQ (NeurIPS'23)	34.71	1.25	3.88s	39.64	1.50	3.90s	39.17	1.39	3.91s	47.74	1.67	11.34s	120.23	4.75	72.12s	181.04	7.59	76.47s
L2I (ICLR'20)	37.42	1.28	13.56s	44.05	1.58	14.15s	43.56	1.41	14.01s	63.33	3.26	26.14s	204.51	10.31	2.10m	348.70	17.47	4.37m
NLNS (ECAI'20)	41.31	1.27	12.15s	46.52	1.51	12.15s	47.44	1.36	12.16s	60.38	1.89	12.22s	142.87	4.65	12.73s	221.69	6.51	13.02s
DACT (NeurIPS'21)	37.03	0.57	68.63s	43.10	0.66	67.92s	43.50	0.57	67.98s	49.30	0.83	4.52m		OOM			OOM	
L2D (NeurIPS'21)	35.26	1.24	2.60s	41.09	1.50	2.52s	40.40	1.37	2.63s	46.29	1.69	4.24s	108.95	4.63	10.15s	162.90	6.99	19.04s
GLOP-G (AAAI'24)	39.20	1.40	0.44s	43.44	1.63	0.43s	43.46	1.46	0.41s	50.55	1.97	1.88s	117.65	4.80	7.07s	178.37	6.84	7.98s
GLOP-LKH (AAAI'24)	38.71	1.42	1.22s	42.83	1.67	0.93s	42.80	1.49	0.77s	50.42	1.98	3.84s	117.04	4.83	9.85s	178.08	6.90	11.35s
RL-driven HLGP	34.58	1.26	3.47s	39.85	1.54	3.43s	39.36	1.38	3.47s	44.80	1.70	10.37s	106.27	4.52	70.80s	160.73	7.20	3.04m
SL-driven HLGP	**32.55**	1.21	7.21s	**37.96**	1.48	7.55s	**37.40**	1.34	7.41s	**42.85**	1.65	30.61s	**102.27**	4.47	84.12s	**152.47**	6.91	98.49s
SL-driven HLGP-LKH	32.54	1.21	8.79s	37.96	1.47	8.68s	37.39	1.34	8.70s	42.77	1.64	33.16s	102.02	4.47	72.04s	152.33	6.89	87.36s

HLGP delivers superior performance across all other settings. Moreover, the performance improvement achieved by our method becomes more pronounced as the graph size increases. More importantly, the SL-driven HLGP is able to produce high-quality solutions with significantly lower inference time compared to LKH3. For instance, on CVRP10K, our method requires 3.39 minutes per instance and achieves an average cost of 227.07, whereas LKH3 requires 41.29 minutes per instance while yielding a higher average cost of 240.23. Therefore, the advantage of HLGP lies in its ability to deliver competitive or superior solution quality with substantially lower inference time compared to LKH3, making it particularly suitable for large-scale instances. Additionally, we integrate LKH3 as the local permutation policy within the SL-driven HLGP framework, with the results reported in the last rows of Tables 1 and 2, referred to as SL-driven HLGP-LKH. We observe that, in nearly all dataset settings, SL-driven HLGP-LKH continues to outperform standalone LKH3, achieving both lower average cost and reduced inference time. Moreover, compared with GLOP-LKH, which employs the same LKH3 solver as the local permutation policy but achieves inferior performance relative to standalone LKH3, the SL-driven HLGP-LKH produces more refined and effective partition solutions. As a result, when combined with LKH3, it delivers superior overall performance.

To present the relative performance of the SL-driven HLGP model in terms of solution quality and runtime, we compare it with several representative baselines, namely HGS, BQ, and GLOP. The relative performance of the SL-driven HLGP model against each selected baseline is reported in Table 3. We use the performance gap in both solution quality and runtime with respect to each baseline as the evaluation metrics. As shown in Table 3, the SL-driven HLGP achieves solution quality that is close to HGS in terms of average cost, while generating high-quality solutions more rapidly. Compared with both BQ and GLOP, the SL-driven HLGP consistently produces better solutions. Although the runtime of the SL-driven HLGP is higher than that of BQ and GLOP, it still maintains a favorable trade-off between solution quality and computational cost relative to prior methods.

5.4 Training stability

In the literature on learning-based methods for VRPs, it is standard practice to evaluate the stability and robustness of a neural solver by reporting the standard deviation across problem instances within a test dataset. In our work, we follow the same protocol and evaluate

Table 3 Relative performance comparisons between SL-driven HLGP and representative baselines

Datasets	HLGP vs HGS		HLGP vs BQ		HLGP vs GLOP	
	$\Delta_{\text{Avg.}}\downarrow$	$\Delta_{\text{Time.}}\downarrow$	$\Delta_{\text{Avg.}}\downarrow$	$\Delta_{\text{Time.}}\downarrow$	$\Delta_{\text{Avg.}}\downarrow$	$\Delta_{\text{Time.}}\downarrow$
CVRP1K	0.83	-4.43m	-1.17	3.56s	-5.26	7.58s
CVRP2K	1.43	-12.14m	-3.28	16.74s	-5.93	30.66s
CVRP5K	1.29	-17.18m	-13.01	17.38s	-17.54	94.90s
CVRP7K	1.36	-18.00m	-22.05	16.20s	-25.02	2.09m
CVRP10K	0.48	-21.25m	-38.74	5.40s	-39.89	3.16m
CVRP1K+G	0.71	-15.45m	-2.16	3.33s	-6.65	6.77s
CVRP1K+E	0.83	-6.39m	-1.68	3.65s	-5.48	7.12s
CVRP1K+R	0.78	-7.66m	-1.77	3.50s	-6.06	7.00s
CVRP2K+G	1.21	-19.29m	-4.89	19.27s	-7.70	28.73s
CVRP5K+E	1.11	-15.13m	-17.96	12.00s	-15.38	77.05s
CVRP7K+R	1.43	-19.40m	-28.57	22.02s	-25.90	90.51s

the stability of both the HLGP methods and the prior baselines using the standard deviation across problem instances, as reported in Tables 1 and 2. To further demonstrate the training stability of the HLGP methods and eliminate the possibility that a particular random initialization may be favorable to our approach, we additionally conduct five independent training runs with different random seeds. The corresponding results are reported in Table 4. Across all datasets, including both in-distribution and out-of-distribution settings, the performance variation across runs remains limited and the standard deviations are small. These results indicate that the training of the HLGP methods is not sensitive to random initialization.

5.5 Training overhead

We report the training time of both the RL-driven and SL-driven HLGP models, as well as the dataset generation time for the SL-driven HLGP model, in Table 5. The reported training times for both variants include the training of the global and local partition policies. The local permutation policy, however, is pretrained and directly adopted from the publicly available code repository of GLOP [31]. For the SL-driven HLGP model, the dataset generation time includes both the solution generation using LKH3 and the solution generation via beam search. As shown in Table 5, the RL-driven HLGP model requires less training time than the SL-driven variant. Notably, the SL-driven HLGP model incurs additional computational overhead, as the generation of the supervised learning dataset requires more than twice the training time.

5.6 Computation time overhead analysis

In this section, we analyze the time overhead contributions of the global partition process and the local partition process for both the RL-driven HLGP and SL-driven HLGP frameworks, as illustrated in Table 6. For the RL-driven HLGP, it is evident that the local partition process contributes significantly more to the average time overhead compared to the global partition process. This disparity arises primarily due to the local partition process typically involving multiple levels of recursive partitioning. Conversely, the SL-driven HLGP exhibits comparable average time overheads between the global and local partition processes. This similarity can be attributed to the design of the global partition process in the SL-driven framework. In the global partition process of the SL-driven framework, the global partition policy requires to solve a sequence of subproblems $\{I(t)\}_{t=0}^{N_{\text{sol}}}$. As the number of encountered subproblems grows linearly with the graph size, the time overhead associated with the global partition process increases proportionally. In contrast, the RL-driven HLGP framework adopts a more efficient strategy for the global partition process, where a subproblem is formed and passed to the policy network when a new subgraph is constructed. This selective invocation substantially reduces the frequency of global partition policy calls. As a result, the time overhead of the global partition process in the RL-driven HLGP framework is significantly lower than that of the local partition process.

5.7 Hyperparameter studies

We begin by examining the influence of the number of levels, denoted as K, on the performance of both the RL-driven and SL-driven HLGP frameworks. The left portion of Table 7 pres-

Table 4 Performance of the HLGP methods across five independent training runs

	CVRP1K	CVRP2K	CVRP5K	CVRP7K	CVRP10K	CVRP1K+G	CVRP1K+E	CVRP1K+R	CVRP2K+G	CVRP5K+E	CVRP7K+R
RL-driven HLGP											
Avg.↓	43.89	59.70	127.64	171.86	236.54	34.31	39.85	39.26	44.57	105.79	158.26
Std.↓	0.09	0.10	0.38	0.70	1.29	0.05	0.09	0.09	0.11	0.21	0.74
SL-driven HLGP											
Avg.↓	41.97	57.69	124.28	166.71	227.25	32.56	37.97	37.41	42.83	102.31	152.73
Std.↓	0.02	0.02	0.06	0.02	0.12	0.02	0.02	0.02	0.03	0.04	0.07

Table 5 Comparison of training overheads for the HLGP models

RL-driven HLGP	SL-driven HLGP	
Training Time	Training Time	Dataset Generation Time
10.46h	12.50h	24.95h

Table 6 Time overheads of both the RL-driven and SL-driven HLGP frameworks. For the columns grouped under the Local Partition Process, each value is calculated as the difference between the corresponding value under the Global Partition Process and that under the Global Partition Process. These values quantify the improvements in average cost and standard deviation, as well as the additional time overhead introduced specifically by the local partition process

RL-driven HLGP	Global Partition Process			Overall Partition Process			Local Partition Process		
	Avg.	Std.	Time	Avg.	Std.	Time	$\Delta_{\text{Avg.}\downarrow}$	$\Delta_{\text{Std.}}$	$\Delta_{\text{Time}\downarrow}$
CVRP1K+G	37.98	1.35	0.58s	34.58	1.26	3.47s	-3.40	-0.09	2.89s
CVRP2K+U	63.55	1.19	2.37s	59.58	1.17	10.03s	-3.97	-0.02	7.66s
SL-driven HLGP	Global Partition Process			Overall Partition Process			Local Partition Process		
	Avg.	Std.	Time	Avg.	Std.	Time	$\Delta_{\text{Avg.}\downarrow}$	$\Delta_{\text{Std.}}$	$\Delta_{\text{Time}\downarrow}$
CVRP1K+G	32.73	1.21	3.69s	32.55	1.21	7.21s	-0.18	0	3.52s
CVRP2K+U	57.77	1.08	15.11s	57.67	1.08	32.40s	-0.10	0	17.29s

ents the results for the RL-driven HLGP framework, evaluated on the CVRP1K dataset with Gaussian-distributed nodes (CVRP1K+G) and the CVRP2K dataset with uniformly distributed nodes (CVRP2K+U), with K ranging from 0 to 10. These results exhibit that the average cost converges to 34.51 at $K = 7$ for CVRP1K+G and to 59.50 at $K = 8$ for CVRP2K+U. As expected, the computational time increases with the number of levels. The right portion of Table 7 illustrates the performance trends of the SL-driven HLGP framework on the CVRP1K+G and CVRP2K+U datasets as K varies. For CVRP1K+G, the average cost stabilizes at 32.55 when $K = 5$, while for CVRP2K+U, convergence is observed at 57.66 when $K = 6$. Again, the time overhead increases with higher values of K. In practice, to trade off the performance against efficiency, we select $K = 5$ for both RL-driven and SL-driven HLGP.

Moreover, as shown in Table 7, both the RL-driven HLGP and the SL-driven HLGP models exhibit only marginal changes in the objective value, measured by the average cost, once $K \geq 5$. This observation indicates that the primary contribution of the local partition policy to cost reduction is realized within the first five partition levels, after which the performance gradually stabilizes. For the RL-driven HLGP model, the global partition policy tends to generate relatively coarse partition solutions, as the subsequent local partition levels lead to a considerable reduction in cost. In contrast, the global partition policy in the SL-driven HLGP model already produces high-quality partition solutions. As a result, the local partition policy mainly performs finer-grained refinement of node clustering. It is important to emphasize that this does not imply the local partition policy in the SL-driven HLGP model is negligible. For real-world NP-hard problems, even small improvements in solution quality can translate into significant financial savings.

We then proceed to investigate the impact of the coefficients associated with the entropy terms, denoted as λ_G and λ_L, in the loss function of the RL-driven HLGP framework. Since λ_G directly affects the behavior of the global partition policy, we analyze its impact across four datasets with graph sizes ranging from 1,000 to 7,000 nodes, as shown in the left portion of Table 8. The results indicate that setting $\lambda_G = 0.1$ consistently yields the best performance across all datasets. The right portion of Table 8 displays the performance of the RL-driven HLGP as λ_L varies. The configuration with $\lambda_L = 0.005$ achieves superior

Table 7 Hyperparameter study for the number of levels, denoted as K, in both the RL-driven HLGP and the SL-driven HLGP frameworks

	RL-driven HLGP						SL-driven HLGP					
	CVRP1K+G			CVRP2K+U			CVRP1K+G			CVRP2K+U		
	Avg.	Std.	Time	Avg.	Std.	Time	Avg.	Std.	Time	Avg.	Std.	Time
$K = 0$	37.98	1.35	0.58s	63.55	1.19	2.37s	32.73	1.21	3.69s	57.77	1.08	15.11s
$K = 1$	35.13	1.26	1.18s	60.32	1.18	3.94s	32.65	1.21	4.38s	57.73	1.80	19.54s
$K = 2$	34.84	1.26	1.78s	59.85	1.18	5.50s	32.60	1.21	5.06s	57.69	1.08	23.48s
$K = 3$	34.69	1.25	2.37s	59.70	1.17	7.07s	32.58	1.21	5.78s	57.67	1.08	26.96s
$K = 4$	34.63	1.25	2.91s	59.61	1.16	8.51s	32.56	1.21	6.49s	57.67	1.08	29.97s
$K = 5$	34.58	1.26	3.47s	59.58	1.17	10.03s	32.55	1.21	7.21s	57.67	1.08	32.40s
$K = 6$	34.56	1.25	4.10s	59.56	1.17	11.40s	32.55	1.21	7.92s	57.66	1.08	34.96s
$K = 7$	34.51	1.24	4.57s	59.55	1.17	13.00s	32.55	1.21	8.62s	57.66	1.08	37.60s
$K = 8$	34.51	1.24	5.11s	59.50	1.17	14.70s	32.55	1.21	9.34s	57.66	1.08	40.00s
$K = 9$	34.51	1.24	5.77s	59.50	1.17	16.39s	32.55	1.21	10.05s	57.66	1.08	42.37s
$K = 10$	34.51	1.24	6.31s	59.50	1.17	17.66s	32.54	1.21	10.76s	57.66	1.08	44.77s

Table 8 Hyperparameter study of λ_G and λ_L in the RL-driven HLGP framework on uniformly distributed CVRP instances

λ_G	CVRP1K		CVRP2K		CVRP5K		CVRP7K		λ_L	CVRP1K		CVRP2K		CVRP5K		CVRP7K	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.		Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
0.1	46.40	0.91	63.55	1.19	138.37	3.51	187.91	4.99	0.1	43.92	0.82	59.81	1.11	128.31	3.04	173.78	4.36
0.05	46.48	0.82	63.68	1.21	138.68	3.57	188.65	5.09	0.05	43.91	0.82	59.73	1.12	128.16	3.00	173.63	4.35
0.01	46.51	0.92	63.69	1.19	139.02	3.57	189.31	5.13	0.01	43.91	0.81	59.80	1.12	128.52	3.01	174.02	4.32
0.005	46.54	0.92	63.77	1.20	138.74	3.54	188.71	5.05	0.005	43.78	0.85	59.58	1.17	128.12	3.06	173.71	4.39
0.001	46.64	0.93	63.84	1.20	139.49	3.61	190.13	5.18	0.001	44.01	0.81	60.01	1.13	128.38	3.02	173.40	4.22
0.0005	46.54	0.92	63.78	1.20	139.20	3.62	189.80	5.23	0.0005	44.05	0.82	60.14	1.11	128.38	3.03	173.35	4.23
0.0001	46.60	0.92	63.70	1.20	138.89	3.60	189.04	5.19	0.0001	44.47	0.94	60.14	1.25	133.39	3.41	182.30	4.78
0.0	46.55	0.94	63.75	1.21	139.16	3.58	189.30	5.14	0.0	43.94	0.86	59.70	1.17	129.07	3.08	175.10	4.27

results on three out of the four datasets. Based on these findings, we adopt $\lambda_G = 0.1$ and $\lambda_L = 0.005$ as the default settings for the RL-driven HLGP. It is also worth noting that the coefficients for L2 regularization terms incorporated into the loss function of the SL-driven HLGP framework, are commonly set to 1×10^{-6} in Transformer-based neural solvers [15, 16, 34, 42]. To maintain consistency, we apply this setting to both the global and local partition policies in the SL-driven HLGP, i.e., $\lambda_G = \lambda_L = 1 \times 10^{-6}$. This configuration has proven effective in our experiments.

5.8 Ablation studies

In this section, we analyze the contributions of individual modules within the HLGP framework using the CVRP1K+G and CVRP2K+U datasets. The upper portion of Table 9 presents the results for the RL-driven HLGP framework. "Glob." refers to a model where only the global partition policy is utilized, trained on randomly sampled CVRP1K instances. "Glob.+Subp." indicates that the global partition policy is trained on both the random CVRP1K instances and the subproblems encountered during training. In contrast, "Glob.+Loc." and "Glob.+Loc.+Subp." incorporate the local partition policy, which is designed to progressively rectify misclusterings. As shown in Table 9, both the "Subp" and "Loc." modules contribute to performance improvements. However, the "Loc." module demonstrates a more substantial impact on the final performance across both evaluation datasets. Additionally, Fig. 4 presents the training and validation curves of the global partition policy in the RL-driven HLGP framework. It is clear that incorporating the subproblems encountered during training expedites convergence, thereby enhancing training efficiency.

The bottom portion of Table 9 shows the results for the SL-driven HLGP framework. The configuration "SS.+Glob." refers to using only the global partition policy trained on small-scale instances (100 nodes). "SS.+LS.+Glob." extends this by further including training on large-scale instances (1,000 nodes). Similarly, "SS.+Glob.+Loc." and "SS.+LS.+Glob.+Loc." incorporate the local partition policy into the framework. The results indicate that, for the CVRP1K+G dataset, the "LS." and "Loc." modules contribute comparably to performance improvements. However, on the larger-scale CVRP2K+U dataset, the "LS." module demonstrates a more pronounced impact than the "Loc." module. These findings suggest that the proposed multi-level HL framework, when combined with beam search for each policy component, can generate high-quality solutions that serve as reliable supervision signals. This, in turn, enhances the generalization capability of the partition policies across varying problem scales.

We then conduct an ablation analysis of each hierarchical level in HLGP and compare it with non-hierarchical backbones. The partition policies in HLGP are themselves capable of generating feasible solutions for CVRP instances. Under this view, HLGP can be interpreted as enhancing the solution produced by the neural network used in the partition process through the integration of a pretrained permutation policy. Accordingly, the non-hierarchical counterpart of the RL-driven HLGP corresponds to a model that relies solely on the neural network used in the partition process that is trained to directly generate a CVRP solution. For the SL-driven setting, the corresponding non-hierarchical baseline is BQ. To clearly demonstrate the contributions of the global and local partition levels, we report the

Table 9 Ablation study for the RL-driven HLGP and the SL-driven HLGP frameworks

RL-driven HLGP	Glob.		Glob.+Subp.		Glob.+Loc.		Glob.+Loc.+Subp.	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
CVRP1K+G	39.12	1.41	37.98	1.35	35.04	1.28	34.58	1.26
CVRP2K+U	64.38	1.25	63.55	1.19	60.33	1.14	59.58	1.17
SL-driven HLGP	SS.+Glob.		SS.+LS.+Glob.		SS.+Glob.+Loc.		SS.+LS.+Glob.+Loc.	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
CVRP1K+G	33.28	1.26	32.73	1.21	32.88	1.23	32.55	1.21
CVRP2K+U	59.51	1.09	57.77	1.08	59.10	1.09	57.67	1.08

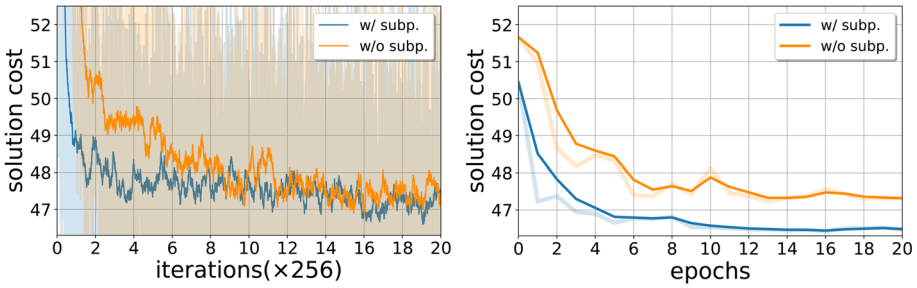

Fig. 4 The training curve (**a**) and the validation curve (**b**) of the global partition policy in the RL-driven HLGP framework

Table 10 Performance comparisons between the HLGP variants and their corresponding non-hierarchical backbone methods

RL-driven HLGP	Non-hierarchical Backbone	Global Partition Level		Local Partition Levels	
	Avg.↓	Avg.↓	$\Delta_{\text{Avg.}}$↓	Avg.↓	$\Delta_{\text{Avg.}}$↓
CVRP1K+G	42.34	37.98	-4.36	34.58	-3.40
CVRP2K+U	76.87	63.55	-13.32	59.58	-3.97
SL-driven HLGP	Non-hierarchical Backbone	Global Partition Level		Local Partition Levels	
	Avg.↓	Avg.↓	$\Delta_{\text{Avg.}}$↓	Avg.↓	$\Delta_{\text{Avg.}}$↓
CVRP1K+G	34.71	32.73	-1.98	32.55	-0.18
CVRP2K+U	60.95	57.77	-3.18	57.67	-0.10

comparative results of the HLGP variants and their corresponding backbone methods in Table 10. As shown in Table 10, for the RL-driven HLGP model, both the global and local partition levels reduce the average cost in the CVRP1K+G setting compared with the non-hierarchical backbone. In the CVRP2K+U setting, the non-hierarchical backbone exhibits poor generalization under scale shifts. Thus, the global partition level significantly reduces the average cost, while the local partition levels further mitigate the compounded errors introduced at earlier stages. For the SL-driven HLGP model, the global partition level contributes more substantially to the reduction in average cost than the local partition levels. As a result, the local partition policy mainly performs finer-grained refinement of node clustering. Importantly, this does not imply that the local partition policy in the SL-driven HLGP model is negligible. In real-world NP-hard problems, even marginal improvements in solution quality can translate into meaningful financial savings.

5.9 Contribution analysis of local partition levels

In this section, we compare the contributions of the first local partition level ($K = 1$) and the subsequent local partition levels ($K > 1$), as summarized in Table 11. This table reports the individual cost reductions and their corresponding contribution percentages, derived from the results presented in Tables 7. According to Table 11, in the RL-driven HLGP model, the subsequent local partition levels still achieve additional cost reductions of 0.55 and 0.74 on the two datasets, accounting for 16.18% and 18.64% of the total cost reduction, respec-

Table 11 Contributions of the first local partition level and the subsequent local partition levels

RL-driven HLGP	First Local Partition ($K = 1$)		Subsequent Local Partitions ($K > 1$)		Total ($K \geq 1$)
	Cost Reduction	Percentage	Cost Reduction	Percentage	Cost Reduction
CVRP1K+G	2.85	83.82%	0.55	16.18%	3.40
CVRP2K+U	3.23	81.36%	0.74	18.64%	3.97
SL-driven HLGP	First Local Partition ($K = 1$)		Subsequent Local Partitions ($K > 1$)		Total ($K \geq 1$)
	Cost Reduction	Percentage	Cost Reduction	Percentage	Cost Reduction
CVRP1K+G	0.08	44.44%	0.10	55.56%	0.18
CVRP2K+U	0.04	40.00%	0.06	60.00%	0.10

tively. These contributions are therefore non-negligible. In contrast, in the SL-driven HLGP model, the subsequent local partition levels contribute even more substantially, accounting for 55.56% and 60% of the total cost reduction on the two datasets, which exceeds the contribution of the first local partition level. These results indicate that a single local partition level is insufficient. Instead, they provide empirical support for our primary argument that the partition task requires both global and local partition processes to mitigate compounding errors in node clustering.

5.10 Solution quality–efficiency trade-off

To analyze the trade-off between solution quality and computational efficiency of the HLGP framework, we report results on the CVRP5K+U dataset, which contains large-scale instances with 5,000 nodes per graph, as shown in Table 12. In addition to the average solution cost and the inference time per instance, Table 12 also reports the incremental improvement in solution quality achieved at each level compared with the preceding level, as well as the additional inference time introduced by increasing the number of levels. For the RL-driven HLGP, each additional level incurs approximately 10 seconds of extra inference time per instance. The first local partition level contributes the most significant improvement, reducing the cost by over 8 units. From levels 2 to 5, the improvement becomes marginal, with each level reducing the cost by less than 1 unit, resulting in a total reduction of about 2 units across these levels. In contrast, for the SL-driven HLGP, each additional level increases the inference time by approximately 3 seconds per instance. Similarly, the first local parti-

Table 12 Trade-off between solution quality and computational efficiency

	RL-driven HLGP				SL-driven HLGP			
	CVRP5K+U				CVRP5K+U			
	Avg.	$\Delta_{\text{Avg.}\downarrow}$	Time	$\Delta_{\text{Time}\downarrow}$	Avg.	$\Delta_{\text{Avg.}\downarrow}$	Time	$\Delta_{\text{Time}\downarrow}$
$K = 0$	138.37	–	32.30s	–	124.36	–	81.54s	–
$K = 1$	129.96	-8.41	42.56s	+10.26	124.25	-0.11	84.62s	+3.08
$K = 2$	129.12	-0.84	52.51s	+9.95	124.17	-0.08	87.73s	+3.11
$K = 3$	128.60	-0.52	63.04s	+10.53	124.14	-0.03	90.61s	+2.88
$K = 4$	128.36	-0.24	72.50s	+9.46	124.13	-0.01	94.08s	+3.47
$K = 5$	128.12	-0.24	82.59s	+10.09	124.13	0.00	97.27s	+3.19
$K = 6$	128.12	0.00	93.66s	+11.07	124.11	-0.02	100.15s	+2.88
$K = 7$	128.02	-0.10	103.91s	+10.25	124.10	-0.01	103.45s	+3.30
$K = 8$	128.02	0.00	112.60s	+8.69	124.10	0.00	106.63s	+3.18
$K = 9$	127.95	-0.07	124.00s	+11.40	124.10	0.00	109.85s	+3.22
$K = 10$	127.95	0.00	133.60s	+9.60	124.10	0.00	112.62s	+2.77

tion level provides the largest improvement, reducing the cost by 0.11 units, while levels 2 to 5 collectively contribute an additional 0.12 units of improvement. These results indicate that most of the performance gains are achieved at the early partition levels, while deeper levels primarily provide diminishing returns at the cost of increased runtime. Therefore, in practical applications, the number of levels K can be adjusted to balance solution quality and computational efficiency according to the requirements of the deployment scenario.

6 Conclusion and discussion

6.1 Conclusion

In this work, our primary contributions can be summarized as follows. We propose a novel hierarchical learning-based framework for the graph partition problem to efficiently solve large-scale CVRP instances. Within this framework, the global partition policy and local partition policy synergistically address the partition task, progressively mitigating compounded misclusterings during the partition process. The proposed approach adopts a unified objective function that is compatible with both RL and SL training paradigms. In addition, we analyze the importance of treating subproblems encountered during training as individual instances in both RL and SL settings. Extensive experimental results demonstrate the strong generalization capability of the proposed HLGP framework in efficiently producing low-cost CVRP solutions under both distributional and scale shifts.

In this work, the main findings can be summarized as follows. Empirically, compared with SOTA heuristic methods, the SL-driven HLGP method outperforms LKH3 and OR-Tools while achieving faster inference speed, and exhibits only a marginal performance gap compared with HGS while still maintaining faster inference. Compared with existing learning-based baselines, both the RL-driven and SL-driven methods achieve the best overall performance. In terms of inference time overhead, the local partition process of the RL-driven HLGP contributes significantly more to the average runtime than the global partition process. In contrast, the SL-driven HLGP exhibits comparable time overheads between the global and local partition processes. Furthermore, both the RL-driven HLGP and the SL-driven HLGP models exhibit only marginal changes in the objective value, measured by the average cost, once the number of local partition levels satisfies $K \geq 5$. This observation indicates that most of the cost reduction achieved by the local partition policy occurs within the first five partition levels, after which the performance gradually stabilizes. Therefore, in practical applications, the number of levels K can be adjusted to balance solution quality and computational efficiency according to the requirements of the deployment scenario. Finally, in the RL-driven HLGP framework, the local partition policy contributes more significantly to the overall performance improvement. In contrast, for the SL-driven HLGP framework, training the global partition policy on large-scale instances plays a more critical role in performance improvement, but it incurs substantially higher training overhead due to the need for dataset generation.

In this work, the primary implications can be summarized as follows. For the domain of VRPs, our study introduces a novel hierarchical learning-based graph partition framework that can efficiently generate high-quality solutions for large-scale CVRP instances with rapid inference. For the broader domain of COPs, the proposed HLGP framework demonstrates the effectiveness of combining global and local partition policies to mitigate com-

pounded errors in node clustering. This hierarchical paradigm may inspire future research to adopt similar strategies for COPs that require node clustering as an intermediate step. From a practical perspective, the HLGP framework demonstrates advantages in both solution quality and inference efficiency compared with SOTA heuristic methods such as LKH3 and OR-Tools. Therefore, the proposed approach has strong potential for deployment in real-world scenarios where multiple vehicles are required to serve large numbers of customers. In such applications, the reduction in average routing cost may translate into meaningful operational and financial savings.

6.2 Discussion on extending HLGP to other COPs

We use the Maximum Cut (MaxCut) problem as a case study to illustrate the potential extension of the HLGP framework to other COPs. Before describing how HLGP can be applied to MaxCut, we would like to clarify that HLGP is a unified framework with the potential to generalize across different COPs. Specific components of HLGP may require adaptation to accommodate the structural characteristics and objective functions of each problem. In this response, we outline some potential design choices for these components when extending HLGP to other COPs. A systematic investigation of their effectiveness is beyond the scope of the current paper and is left for future work. After presenting the potential extension of HLGP to MaxCut, we briefly outline the general pipeline for adapting the HLGP framework to other COPs.

In MaxCut, given a graph, the objective is to divide the vertex set into two complementary subsets such that the number of edges between the two subsets is maximized. A feasible solution can be constructed incrementally by sequentially assigning vertices to one subset, while the remaining vertices form the complementary subset. During this construction procedure, vertices are added as long as the number of edges between the selected and unselected vertices increases. In the HLGP framework adapted for MaxCut, the global partition policy sequentially selects vertices into a partial solution, provided that the number of edges between the selected and unselected vertices continues to increase. Based on the resulting selected and unselected vertex sets, subgraphs can be constructed in various ways. One potential design is to define proxy vertices as selected vertices that have direct connections to unselected vertices. For each proxy vertex, a corresponding subgraph can be defined as the set of vertices within an M-hop neighborhood of that proxy node in the original graph. At each local partition level, different strategies may be adopted to select pairs of subgraphs to form subproblems. For example, subgraphs may be selected randomly or based on the connectivity between their proxy vertices. The local partition policy is then applied to each constructed subproblem, yielding a refined set of subgraphs. After completing the partition process, each subgraph can be solved using a local MaxCut solver, and the final solution is consequently derived for the MaxCut.

In this overall design, both the global and local partition policies are capable of producing feasible solutions for the MaxCut. However, the subgraphs and subproblems are derived from the intermediate solutions generated by these partition policies. This design is analogous to the HLGP framework for CVRP, where both the global and local partition policies can generate feasible routing solutions, while the sequential ordering information is subsequently discarded when constructing subgraphs and subproblems for further refinement. Moreover, different COPs require different designs of the local solving component. For MaxCut, this

corresponds to employing a local MaxCut solver for each subgraph, whereas for CVRP, a local permutation policy is required to determine the visiting order within each partition. Thus, the general pipeline for extending HLGP to other COPs can be summarized as follows. First, both the global and local partition policies are designed to generate feasible solutions for the target problem. Next, subgraphs are derived from these feasible solutions, and subproblems are constructed by combining pairs of subgraphs according to a predefined strategy. Finally, after completing the partition process, a problem-specific local solver is applied to each subgraph, and the resulting partial solutions are integrated to obtain the final solution.

Appendix A: Implementation details of baselines

In this section, we provide the implementation details of the baseline methods, including both heuristic and learning-based approaches, selected for comparison with our proposed approaches.

LKH3 [9] We adhere to the settings outlined in Omni-POMO [42] to reproduce the results of LKH3. For solving each CVRP instance with varying scales and distributions, we set the maximum number of trials to 10000.

HGS [10] In order to decrease the runtime, we adjust the number of iterations of HGS for the CVRP datasets with varying scales. Specifically, we set the number of iterations to 20000, 5000, 2000, 1500, and 1000 for the CVRP instances with node counts of 1000, 2000, 5000, 7000, and 10000, respectively.

AM [13] We follow the implementation guidelines of AM as outlined by [31], where the checkpoint trained on CVRP100 is adapted for comparison purposes. In the evaluation stage, to enhance the quality of solutions generated by AM, we employ a sampling decoding strategy with 1280 solutions for each instance. The temperature of the softmax function in the output layer is set to 0.1 to prevent solutions from diverging towards lower-quality outcomes.

POMO [15] To benchmark against POMO, we generalize the checkpoint trained on CVRP100 to generate results across different CVRP datasets. The POMO size is adjusted to align with the number of nodes in each CVRP instance. Furthermore, data augmentation is implemented for each CVRP instance with an augmentation factor of 8.

Sym-POMO [16] We stick to the original implementations of Sym-POMO, where the POMO size is adjusted to match the number of nodes, and each CVRP instance is augmented by a factor of 8. The checkpoint trained on CVRP100 is generalized for comparisons.

AMDKD [45] AMDKD aims to improve the generalization capacity of neural solvers on CVRP datasets with scale and distribution shifts through knowledge distillation. We adhere to the original evaluation settings with the model trained via knowledge distillation on CVRP100 datasets with various distributions. The default POMO size and augmentation factor are employed in our evaluations.

Omni-POMO [42] Omni-POMO aims to transfer neural solvers to problem instances with diverse scales and distributions through meta-learning techniques. Therefore, we strictly follow the evaluation configurations by utilizing the provided model trained via MAML with 250000 epochs to replicate the results of Omni-POMO across various CVRP datasets. The POMO size is configured to match the number of nodes in each CVRP instance. Additionally, data augmentation is applied to every CVRP instance with an augmentation factor of 8.

ELG-POMO [34] ELO-POMO integrates both a global neural solver and a local neural solver to leverage insensitive local topological features, thereby enhancing the overall system's generalization capability. We generalize the checkpoint trained on CVRP100 with 250000 epochs for comparison purposes. Throughout the evaluation phase, we maintain the default values of the POMO size and augmentation factors as recommended in the original implementations.

INViT [35] INViT utilizes multiple encoders to capture the features of subgraphs of varying scales within each CVRP instance, enabling the utilization of local topological features for the generalization improvement. In our study, we generalize the checkpoint trained on CVRP100 to evaluate the generalization ability against other baselines. During the evaluation phase, given that each instance in the evaluation dataset consists of at least 1000 nodes, the beam size is accordingly set to 16 for each evaluation CVRP dataset, as recommended in the original implementation. All other settings strictly conform to the original implementations.

LEHD [39] LEHD incorporates iterative local revision (known as RRC) by utilizing a SL-trained neural solver to improve the generalization capability of the constructive method. We leverage the officially provided checkpoint to compare its generalization ability with other methods. During the evaluation phase, as all baseline methods are evaluated on large-scale instances with a minimum of 1000 nodes, to balance performance and efficiency, the number of RRC is set to 200, 50, 5, and 1 for CVRP datasets with node counts of 1000, 2000, 5000, and 7000. For the CVRP10K dataset, the greedy model is directly deployed for comparison purposes.

BQ [38] BQ enhances generalization ability by leveraging the inherent recursive property of the CVRP instance. We utilize the officially provided checkpoint to generalize its application to various CVRP datasets. During the evaluation phase, to achieve a trade-off between performance and efficiency, the beam size is set to 16, 16, 8, 4, and 4 for the CVRP datasets with node counts of 1000, 2000, 5000, 7000, and 10000, respectively. Other settings remain aligned with the original implementations.

L2I [17] L2I is an iterative approach that merges an RL-based policy with predefined local operators to iteratively enhance a given solution. We directly apply the provided official checkpoint of the RL-based policy and the predefined set of local operators to diverse CVRP datasets for generalization comparisons. Additionally, to prevent unexpected increases in makespan due to excessive iterations, we limit the maximum number of rollout steps for the RL-based policy to 40, 10, 6, 4, and 2 for the CVRP datasets with node counts of 1000, 2000, 5000, 7000, and 10000, respectively.

NLNS [19] NLNS seamlessly integrates heuristic destroy operators and a collection of learning-based repair policies to iteratively enhance a given solution. We closely follow the official implementations of NLNS for generalization comparisons, with the exception of the setting for maximum running time per instance. Since NLNS is intended to generalize to large-scale instances, the maximum running time in our study is extended to approximately 1200ms per instance for various CVRP datasets.

DACT [20] DACT, as an iterative method, redesigns the solution representation for the RL-based policy to improve the given solution. We thus generalize the official checkpoint trained on CVRP100 to various CVRP datasets for comparisons. Following the guidelines in the official implementation, to tailor DACT to larger CVRP instances, smaller values are recommended for the number of perturbations and the dummy rate. As a result, we set the dummy rate to 0.05 and the number of perturbations to 2. Furthermore, to prevent unexpected increases in makespan resulting from an excessive number of iterations, the maximum rollout steps of the RL-based policy are set to 400.

L2D [26] L2D is replicated by adhering to the publicly available open-source implementation across various CVRP datasets. It is impressive to observe L2D's strong performance on CVRP datasets exhibiting distribution and scale shifts. This success can be attributed to L2D's heavy reliance on near-optimal solutions from numerous neighboring subgraphs, derived from classical heuristic solvers, to guide both clustering subgraphs and resolving subproblems.

GLOP [31] We rigorously follow the official implementations of GLOP to benchmark it against other baseline methods. Moreover, for the selected baseline methods that align with those in GLOP, we rely on GLOP's recommendations to reproduce these methods, thereby ensuring that our replicated results are at least on par with those documented in GLOP.

Appendix B: Visualization results

Figures 5 and 6 illustrate the CVRP solutions generated by the RL-driven and SL-driven HLGP frameworks, respectively, on the CVRP1K datasets with node distributions including Uniform, Gaussian, Explosion, and Rotation. In Fig. 5, we illustrate the impact of training on the encountered subproblems for the global partition policy, as well as the effect of including or excluding the local partition policy in the RL-driven HLGP model. In Fig. 6, we show the differences arising from training on large-scale problem instances and the presence or absence of the local partition policy. Although the CVRP solutions exhibit similar strip-shaped loop pattern within each sub-route, the details of node clustering differ significantly, which substantially affects the final solution costs.

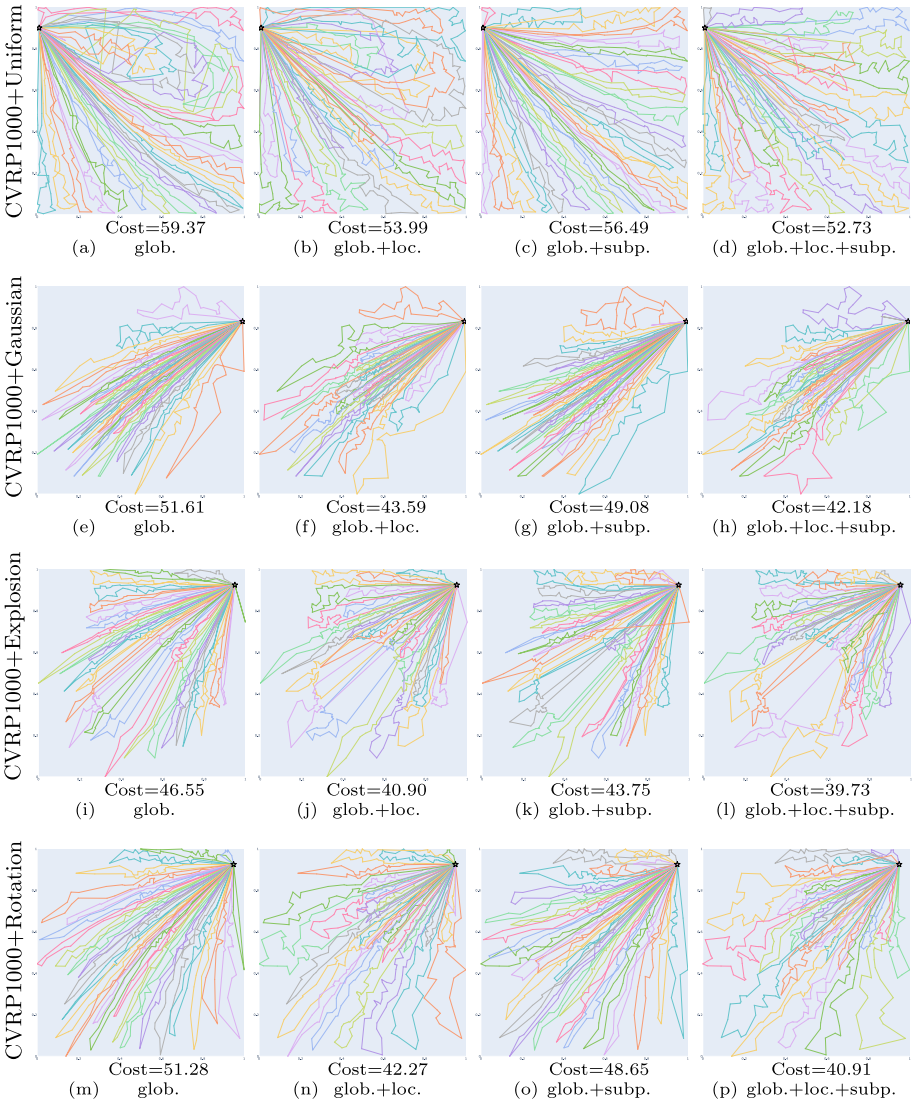

Fig. 5 Visualizations of CVRP Solutions generated by the RL-driven framework

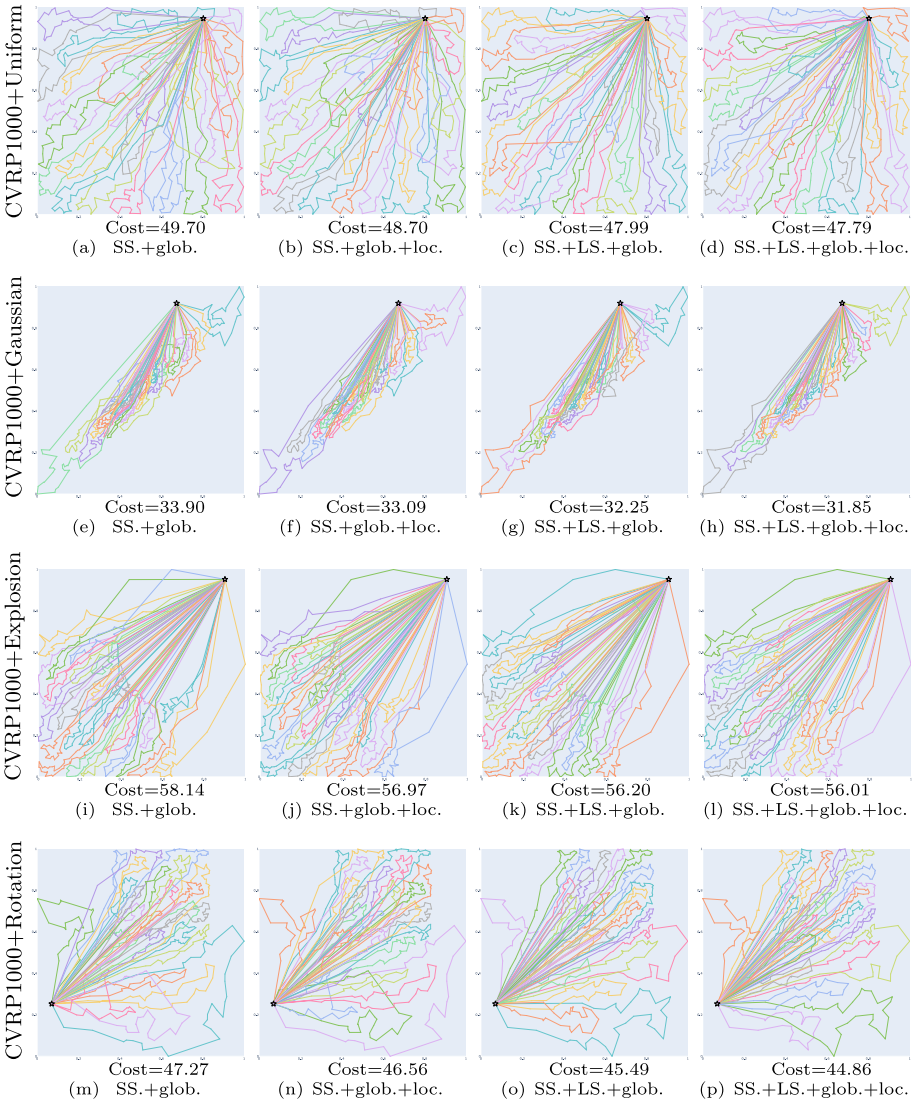

Fig. 6 Visualizations of CVRP Solutions generated by the SL-driven framework

Appendix C: Algorithm Pseudocodes

The algorithm pseudocodes for both the RL-driven and SL-driven HLGP frameworks are illustrated in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 RL-driven HLGP.

Input: Problem instance distribution $p_I(\cdot)$, Permutation policy π_{perm}
Output: Global partition policy π_{θ_G}, Local partition policy π_{θ_L}

for $n = 0$ to N **do**
  Sample an instance $I = (G, D, N_{\max}) \sim p_I(\cdot)$
  Initialize the partition solution $\mathcal{C}^{(0)} = \{\}$
  while $G \neq \emptyset$ **do**
    Sample a partition solution $\hat{\mathcal{C}}^{(0)} = \{\hat{c}_1^{(0)}, \ldots, \hat{c}_{N_c}^{(0)}\} \sim \pi_{\theta_G^n}(\cdot|I)$
    Update $N_{\max} \leftarrow N_{\max} - 1$
    Update $G \leftarrow \cup_{i=2}^{N_c} \hat{c}_i^{(0)}$
    Construct a subproblem $I \leftarrow (G, D, N_{\max})$
    Update the partition solution $\mathcal{C}^{(0)} \leftarrow \mathcal{C}^{(0)} \cup \{\hat{c}_1^{(0)}\}$
    `# train global partition policy`
    CVRP solution $\mathcal{T} \sim \pi_{\text{perm}}(\cdot|\hat{\mathcal{C}}^{(0)})$
    Solution cost $e(\mathcal{T}) = \sum_{\tau \in \mathcal{T}} e(\tau)$
    Update the parameter using RL $\theta_G^n \leftarrow \text{AdamOpt}(\theta_G^n, \nabla_{\theta_G} \hat{J}(\theta_G^n, \theta_L^n))$
  end while
  for $k = 1$ to K **do**
    Construct subproblems $\{I_j^{(k-1)}\}_{j=1}^{\lfloor \frac{N_c}{2} \rfloor}$ from $\mathcal{C}^{(k-1)}$
    for $j = 1$ to $\lfloor \frac{N_c}{2} \rfloor$ **do**
      Sample a partition solution $\mathcal{C}_j^{(k-1)} \sim \pi_{\theta_L^n}(\cdot|I_j^{(k-1)})$
      `# train local partition policy`
      CVRP solution $\mathcal{T}_j \sim \pi_{\text{perm}}(\cdot|\mathcal{C}_j^{(k-1)})$
      Solution cost $e(\mathcal{T}_j) = \sum_{\tau \in \mathcal{T}_j} e(\tau)$
      Update the parameter using RL $\theta_L^n \leftarrow \text{AdamOpt}(\theta_L^n, \nabla_{\theta_L} \hat{J}(\theta_G^n, \theta_L^n))$
    end for
    Construct the partition solution $\mathcal{C}^{(k)}$ from $\{\mathcal{C}_j^{(k-1)}\}_{j=1}^{\lfloor \frac{N_c}{2} \rfloor}$ and $\mathcal{C}^{(k-1)}$
  end for
  $\theta_G^{n+1} \leftarrow \theta_G^n; \theta_L^{n+1} \leftarrow \theta_L^n$
end for

Algorithm 2 SL-driven HLGP.

Input: Problem instance distribution p_I, Permutation policy π_{perm}
Output: Global partition policy π_{θ_G}, Local partition policy π_{θ_L}
for $n = 0$ to N **do**
  Sample an instance $I = (G, D, N_{\max}) \sim p_I(\cdot)$
  Generate candidate solutions $\{\bar{\mathcal{C}}^{(0),i}\}_{i=1}^{N_{\text{cand}}} \sim \text{BeamSearch}(I, \pi_{\theta_G^n})$
  for $i = 1$ to N_{cand} **do**
    CVRP solution $\mathcal{T}^i \sim \pi_{\text{perm}}(\cdot|\bar{\mathcal{C}}^{(0),i})$
    Solution cost $e(\mathcal{T}^i) = \sum_{\tau \in \mathcal{T}^i} e(\tau)$
  end for
  Generate the label $\bar{\mathcal{C}}^{(0)} = \arg\min_{\bar{\mathcal{C}}^{(0),i}} \{e(\mathcal{T}^1), \ldots, e(\mathcal{T}^{N_{\text{cand}}})\}$
  for $k = 1$ to K **do**
    Construct subproblems $\{I_j^{(k-1)}\}_{j=1}^{\lfloor \frac{N_c}{2} \rfloor}$ from $\bar{C}^{(k-1)}$
    for $j = 1$ to $\lfloor \frac{N_c}{2} \rfloor$ **do**
      Generate candidate solutions:
      $\{\bar{\mathcal{C}}_j^{(k-1),i}\}_{i=1}^{N_{\text{cand}}} \sim \text{BeamSearch}(I_j^{(k-1)}, \pi_{\theta_L^n})$
      for $i = 1$ to N_{cand} **do**
        CVRP solution $\mathcal{T}_j^i \sim \pi_{\text{perm}}(\cdot|\bar{\mathcal{C}}_j^{(k-1),i})$
        Solution cost $e(\mathcal{T}_j^i) = \sum_{\tau \in \mathcal{T}_j^i} e(\tau)$
      end for
      Generate the label $\bar{\mathcal{C}}_j^{(k-1)} = \arg\min_{\bar{\mathcal{C}}_j^{(k-1),i}} \{e(\mathcal{T}_j^1), \ldots, e(\mathcal{T}_j^{N_{\text{cand}}})\}$
    end for
    Construct the label $\bar{\mathcal{C}}^{(k)}$ from $\{\bar{\mathcal{C}}_j^{(k-1)}\}_{j=1}^{\lfloor \frac{N_c}{2} \rfloor}$ and $\bar{\mathcal{C}}^{(k-1)}$
  end for
  $\bar{\mathcal{C}} \leftarrow \bar{\mathcal{C}}^{(K)}$
  `# train global partition policy`
  Generate labeled instances $\{(I(t), \bar{\mathcal{C}}[t])\}_{t=0}^{N_{\text{sol}}}$
  Update the parameter using SL $\theta_G^n \leftarrow \text{AdamOpt}(\theta_G^n, \nabla_{\theta_G} \hat{J}(\theta_G^n, \theta_L^n))$
  `# train local partition policy`
  Generate labeled instances $\{(I_i, \bar{C}_i)\}_{i=1}^{N_c}$
  Update the parameter using SL $\theta_L^n \leftarrow \text{AdamOpt}(\theta_L^n, \nabla_{\theta_L} \hat{J}(\theta_G^n, \theta_L^n))$
  $\theta_G^{n+1} \leftarrow \theta_G^n; \theta_L^{n+1} \leftarrow \theta_L^n$
end for

Appendix D: Proofs

D.1 Proof of Theorem 1

Proof For a given CVRP instance *I*, a feasible CVRP solution $\mathcal{T}$ comprises N_τ subtours τ_i , where $1 \le i \le N_\tau$. If a partition solution $\mathcal{C} = \{c_1, \ldots, c_{N_c}\}$ corresponds to $\mathcal{T}$, then each subtour τ_i can be obtained by reordering the nodes within the corresponding subgraph c_i , implying that $N_\tau = N_c$. This observation indicates that a given CVRP solution $\mathcal{T}$ can be represented by the pair $(\mathcal{C}, \mathcal{T}) = \{c_1, \ldots, c_{N_c}, \tau_1, \ldots, \tau_{N_c}\}$, where $\mathcal{C}$ captures the partition structure and $\mathcal{T}$ encodes the intra-cluster node orderings. Therefore, we can derive the following expressions for the primitive objective.

$$
\begin{aligned}
&\min_{\pi} \ \mathbb{E}_{\mathcal{T}}[e(\tau)] \\
=&\min_{\pi} \ \sum_{\mathcal{T}} \pi(\mathcal{T}|I) e(\mathcal{T}) \\
=&\min_{\pi} \ \sum_{(\mathcal{C},\mathcal{T})} \pi((\mathcal{C},\mathcal{T})|I) e(\mathcal{T}) \\
=&\min_{\pi_{\text{part}},\pi_{\text{perm}}} \sum_{\mathcal{C}} \pi_{\text{part}}(\mathcal{C}|I) \sum_{\mathcal{T}} \prod_{i=1}^{N_c} \pi_{\text{perm}}(\tau_i|c_i) \sum_{i}^{N_c} e(\tau_i).
\end{aligned} \tag{D1}
$$

We can further simplify the expectation term associated with π_{perm} as follows:

$$
\textstyle\sum_{\mathcal{T}} \prod_{i=1}^{N_c} \pi_{\text{perm}}(\tau_i|c_i) \sum_{i}^{N_c} e(\tau_i) = \sum_{i=1}^{N_c} \sum_{\tau_i} \pi_{\text{perm}}(\tau_i|c_i) e(\tau_i). \tag{D2}
$$

Thus, we can plug (D2) into (D1) to derive the following objective function:

$$
\begin{aligned}
&\min_{\pi_{\text{part}},\pi_{\text{perm}}} \sum_{\mathcal{C}} \pi_{\text{part}}(\mathcal{C}|I) \sum_{\mathcal{T}} \prod_{i=1}^{N_c} \pi_{\text{perm}}(\tau_i|c_i) \sum_{i}^{N_c} e(\tau_i) \\
=&\min_{\pi_{\text{part}},\pi_{\text{perm}}} \sum_{\mathcal{C}} \pi_{\text{part}}(\mathcal{C}|I) \sum_{i=1}^{N_c} \sum_{\tau_i} \pi_{\text{perm}}(\tau_i|c_i) e(\tau_i) \\
=&\min_{\pi_{\text{part}}} \sum_{\mathcal{C}} \pi_{\text{part}}(\mathcal{C}|I) \sum_{i}^{N_c} \min_{\pi_{\text{perm}}} \sum_{\tau_i} \pi_{\text{perm}}(\tau_i|c_i) e(\tau_i)
\end{aligned} \tag{D3}
$$

If we denote the optimal permutation policy as π^*_{perm}, then the primitive objective can be expressed as:

$$
\min_{\pi_{\text{part}}} \ \mathbb{E}_{\mathcal{C}\sim\pi_{\text{part}}}[\sum_{i=1}^{N_c} \mathbb{E}_{\tau_i\sim\pi^*_{\text{perm}}}(e(\tau_i))]. \tag{D4}
$$

□

D.2 Proof of Theorem 2

Proof The objective function of the HLGP framework can be equivalently reformulated as follows:

$$
\begin{aligned}
&\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} \ \mathbb{E}_{\mathcal{C}^{(0)}\sim\pi_{\text{Gpart}}} \mathbb{E}_{\mathcal{C}^{(1)}\sim\pi_{\text{Lpart}}} \cdots \mathbb{E}_{\mathcal{C}^{(K)}\sim\pi_{\text{Lpart}}}[f(\mathcal{C}^{(K)})] \\
=&\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} \ \mathbb{E}_{\mathcal{C}^{(0)}\sim\pi_{\text{Gpart}}} \mathbb{E}_{\mathcal{C}^{(1)}\sim\pi_{\text{Lpart}}} \cdots \mathbb{E}_{\mathcal{C}^{(K)}\sim\pi_{\text{Lpart}}}[f(\mathcal{C}^{(K)}) - f(\mathcal{C}^{(K-1)}) \\
&+ f(\mathcal{C}^{(K-1)}) - f(\mathcal{C}^{(K-2)}) + \cdots + f(\mathcal{C}^{(1)}) - f(\mathcal{C}^{(0)}) + f(\mathcal{C}^{(0)})]
\end{aligned} \tag{D5}
$$

The RHS of (D5) can be further decomposed as follows:

$$\begin{aligned}&\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} \mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}\cdots\mathbb{E}_{\mathcal{C}^{(K)}}[f(\mathcal{C}^{(K)})-f(\mathcal{C}^{(K-1)})]\\&+\mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}\cdots\mathbb{E}_{\mathcal{C}^{(K-1)}}[f(\mathcal{C}^{(K-1)})-f(\mathcal{C}^{(K-2)})+\cdots+f(\mathcal{C}^{(1)})-f(\mathcal{C}^{(0)})+f(\mathcal{C}^{(0)})]\end{aligned} \tag{D6}$$

Due to the decomposition demonstrated in (D6), the original objective can be iteratively simplified as:

$$\begin{aligned}\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} &\mathbb{E}_{\mathcal{C}^{(0)}}[f(\mathcal{G}^{(0)})]+\mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}[f(\mathcal{C}^{(1)})-f(\mathcal{C}^{(0)})]+\cdots\\&+\mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}\cdots\mathbb{E}_{\mathcal{C}^{(K)}}[f(\mathcal{C}^{(K)})-f(\mathcal{C}^{(K-1)})].\end{aligned} \tag{D7}$$

Then for any $k \geq 1$, the evaluation for $\mathcal{C}^{(k)}$ can be written as:

$$\begin{aligned}&f(\mathcal{C}^{(k)})-f(\mathcal{C}^{(k-1)})\\&=\sum_{i=1}^{N_c} g(c_i^{(k)})-\sum_{i=1}^{N_c} g(c_i^{(k-1)})\\&=\sum_{j=1}^{\lfloor\frac{N_c}{2}\rfloor}[g(c^{(k)}_{(k+m-1) \bmod N_c+1})+g(c^{(k)}_{(k+m) \bmod N_c+1})]\\&\quad-\sum_{j=1}^{\lfloor\frac{N_c}{2}\rfloor}[g(c^{(k-1)}_{(k+m-1) \bmod N_c+1})+g(c^{(k-1)}_{(k+m) \bmod N_c+1})],\end{aligned} \tag{D8}$$

where $m = 2(j-1)$. Let us define the function $h(\mathcal{C},k,m) = g(c_{(k+m-1) \bmod N_c+1}) + g(c_{(k+m) \bmod N_c+1})$, then the evaluation, denoted as $f(\mathcal{C}^{(k)}) - f(\mathcal{C}^{(k-1)})$, for $\mathcal{C}^{(k)}$ can ultimately be written as:

$$f(\mathcal{C}^{(k)})-f(\mathcal{C}^{(k-1)})=\sum_{j=1}^{\lfloor\frac{N_c}{2}\rfloor}[h(\mathcal{C}^{(k)},k,m)-h(\mathcal{C}^{(k-1)},k,m)]. \tag{D9}$$

□

D.3 Proof of Theorem 3

Proof Given that the feasible cost function is defined as $f(\mathcal{C}) = -\mathbf{1}(\mathcal{C} = \bar{\mathcal{C}})$, the objective of the HLGP framework can be written as:

$$\begin{aligned}&\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} \mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}\cdots\mathbb{E}_{\mathcal{C}^{(K)}}[\mathbf{1}(\mathcal{C}^{(K)}=\bar{\mathcal{C}})]\\&=\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} \frac{1}{K+1}\mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}\cdots\mathbb{E}_{\mathcal{C}^{(K)}}[\mathbf{1}(\mathcal{C}^{(K)}=\bar{\mathcal{C}})+\ldots+\mathbf{1}(\mathcal{C}^{(0)}=\bar{\mathcal{C}})]\end{aligned} \tag{D10}$$

Subsequently, we can apply the same recursive decomposition, which is used to equivalently transform (D5) into (D7), to (D10), yielding the following equivalent formulation:

$$\begin{aligned}
&\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} \mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}\cdots\mathbb{E}_{\mathcal{C}^{(K)}}[\mathbf{1}(\mathcal{C}^{(K)}=\bar{\mathcal{C}})+\ldots+\mathbf{1}(\mathcal{C}^{(0)}=\bar{\mathcal{C}})] \\
=&\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} \mathbb{E}_{\mathcal{C}^{(0)}}[\mathbf{1}(\mathcal{C}^{(0)}=\bar{\mathcal{C}})]+\mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}[\mathbf{1}(\mathcal{C}^{(1)}=\bar{\mathcal{C}})]+\cdots \\
&+\mathbb{E}_{\mathcal{C}^{(0)}}\mathbb{E}_{\mathcal{C}^{(1)}}\cdots\mathbb{E}_{\mathcal{C}^{(K)}}[\mathbf{1}(\mathcal{C}^{(K)}=\bar{\mathcal{C}})]
\end{aligned} \tag{D11}$$

It is evident that the objective function defined in (D11) attains its minimum when the condition $\mathcal{C}^{(0)}=\mathcal{C}^{(1)}=\cdots=\mathcal{C}^{(K)}=\bar{\mathcal{C}}$ is satisfied. At the global partition level ($k=0$), the global partition policy is trained to maximize the probability of generating the solution $\bar{\mathcal{C}}$, thereby ensuring that $\mathcal{C}^{(0)}=\bar{\mathcal{C}}$. In contrast, to ensure that the probability of generating $\bar{\mathcal{C}}$ is maximized at any local partition level $k\geq 1$ by training the local partition policy, we define a set of subproblems $I_i=(G_i,D,2)$, where $1\leq i\leq N_c$ and $G_i=\bar{c}_i\cup\bar{c}_{i \bmod N_c+1}$. The corresponding label for each subproblem is defined as $\bar{\mathcal{C}}_i=\{\bar{c}_i,\bar{c}_{i \bmod N_c+1}\}$. It is clear that the subproblem set $\{I_i=(G_i,D,2)\}_{i=1}^{N_c}$ covers all possible subproblems that may arise at any local partition level. The local partition policy is therefore trained to maximize the probability of generating $\bar{\mathcal{C}}_i$ given the corresponding subproblem I_i. In practical scenarios, maximizing the log-probability objective can be utilized without compromising the theoretical optimality of the resulting policies. Accordingly, the objective function of the SL-driven HLGP framework is formulated as follows:

$$\min_{\pi_{\text{Gpart}},\pi_{\text{Lpart}}} -\log\pi_{\theta_G}(\bar{\mathcal{C}}|I)-\sum_{i=1}^{N_c}\log\pi_{\theta_L}(\bar{\mathcal{C}}_i|I_i). \tag{D12}$$

□

D.4 Proof of Proposition 1

Proof We aim to prove that, at any partition level $k\geq 0$, the definitions of state and action, as defined in the multi-level MDP framework, are sufficient to preserve the Markovian property at that level. At the global partition level, the partial partition solution evolves from $\mathcal{C}^{(0)}[0:t-1]$ to $\mathcal{C}^{(0)}[0:t]$ by selecting an unvisited node $\mathcal{C}^{(0)}[t]$ at each time step t. It is evident that the dynamics remains Markovian at this level. At the local partition level $k\geq 1$, the state consists of the subproblem sequence and the partial solution of the subproblem $I_{j_t}^{(k-1)}$ currently being addressed at time step t. During solving $I_{j_t}^{(k-1)}$, executing the action $u_t^{(k)}$ extends the partial solution of $I_{j_t}^{(k-1)}$ by incorporating an unvisited node, thereby generating the subsequent partial solution of $I_{j_t}^{(k-1)}$. The remaining subproblems remain unaffected throughout this process. Once $I_{j_t}^{(k-1)}$ is completed, the framework proceeds to the next subproblem in the sequence. As a result, the dynamics at each local partition level preserve the Markov property. □

Author Contributions All authors contributed to the study conception and the algorithm design. Yuxin Pan proposed the initial idea, implemented the algorithm, and drafted the manuscript. Ruohong Liu prepared Tables 1-2 and curated the datasets used in the experiments. Yize Chen, Zhiguang Cao and Fangzhen Lin critically revised the manuscript. All authors reviewed and approved the final version of the manuscript.

Funding Open access funding provided by Hong Kong University of Science and Technology. This research is supported by a generous research grant from Xiaoi Robot Technology Limited, the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031), and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

Data Availability The datasets used in this study have been made publicly available at https://github.com/panyxy/hlgp_cvrp.

Code Availability The source code is available at https://github.com/panyxy/hlgp_cvrp.

Materials Availability Not applicable.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

Ethics approval and consent to participate Not applicable.

Consent for publication Not applicable.

Competing interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Toth, P., & Vigo, D. (2002). *The Vehicle Routing Problem*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.
2. Martin, S., Ouelhadj, D., Beullens, P., Ozcan, E., Juan, A. A., & Burke, E. K. (2016). A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research, 254*(1), 169–178.
3. Zhang, K., He, F., Zhang, Z., Lin, X., & Li, M. (2020). Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies, 121*, Article 102861.
4. Bono, G., Dibangoye, J. S., Simonin, O., Matignon, L., & Pereyron, F. (2020). Solving multi-agent routing problems using deep attention mechanisms. *IEEE Transactions on Intelligent Transportation Systems, 22*(12), 7804–7813.
5. Ren, L., Fan, X., Cui, J., Shen, Z., Lv, Y., & Xiong, G. (2022). A multi-agent reinforcement learning method with route recorders for vehicle routing in supply chain management. *IEEE Transactions on Intelligent Transportation Systems, 23*(9), 16410–16420.

6. Garaix, T., Artigues, C., Feillet, D., & Josselin, D. (2010). Vehicle routing problems with alternative paths: An application to on-demand transportation. *European Journal of Operational Research, 204*(1), 62–75.
7. Cattaruzza, D., Absi, N., Feillet, D., & González-Feliu, J. (2017). Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics, 6*(1), 51–79.
8. Laporte, G., & Nobert, Y. (1983). A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum*, *5*(2), 77–85.
9. Helsgaun, K. (2017). An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University, 12*, 966–980.
10. Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research, 60*(3), 611–624.
11. Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems, 28*, 2692–2700.
12. Nazari, M., Oroojlooy, A., Snyder, L., & Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, *31*, 9861–9871.
13. Kool, W., Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems! In *International Conference on Learning Representations*. https://openreview.net/forum?id=ByxBFsRqYm.
14. Xin, L., Song, W., Cao, Z., & Zhang, J. (2020). Step-wise deep learning models for solving routing problems. *IEEE Transactions on Industrial Informatics, 17*(7), 4861–4871.
15. Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., & Min, S. (2020). POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems, 33*, 21188–21198.
16. Kim, M., Park, J., & Park, J. (2022). Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *Advances in Neural Information Processing Systems, 35*, 1936–1949.
17. Lu, H., Zhang, X., & Yang, S. (2020). A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJe1334YDH.
18. Chen, X., & Tian, Y. (2019) Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems, 32*, 6281–6292.
19. Hottung, A., & Tierney, K. (2020). Neural large neighborhood search for the capacitated vehicle routing problem. In *24th European Conference on Artificial Intelligence*
20. Ma, Y., Li, J., Cao, Z., Song, W., Zhang, L., Chen, Z., & Tang, J. (2021). Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In *Advances in Neural Information Processing Systems, 34*, 11096–11107.
21. Xin, L., Song, W., Cao, Z., & Zhang, J. (2021). NeuroLKH: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem. In *Advances in Neural Information Processing Systems, 34*, 7472–7483.
22. Ma, Y., Cao, Z., & Chee, Y.M. (2023). Learning to search feasible and infeasible regions of routing problems with flexible neural k-Opt. In *Advances in Neural Information Processing Systems, 36*, 49555–49578.
23. Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks, 11*(2), 109–124.
24. Fu, Z.-H., Qiu, K.-B., & Zha, H. (2021). Generalize a small pre-trained model to arbitrarily large TSP instances. In *Proceedings of the AAAI Conference on Artificial Intelligence, 35*, 7474–7482.
25. Kim, M., Park, J., & Kim, J. (2021). Learning collaborative policies to solve NP-hard routing problems. In *Advances in Neural Information Processing Systems, 34*, 10418–10430.
26. Li, S., Yan, Z., & Wu, C. (2021). Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems, 34*, 26198–26211.
27. Zong, Z., Wang, H., Wang, J., Zheng, M., & Li, Y. (2022). RBG: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining,* pp. 4648–4658.
28. Cheng, H., Zheng, H., Cong, Y., Jiang, W., & Pu, S. (2023). Select and optimize: Learning to solve large-scale TSP instances. In *International Conference on Artificial Intelligence and Statistics,* pp. 1219–1231. PMLR.
29. Pan, X., Jin, Y., Ding, Y., Feng, M., Zhao, L., Song, L., & Bian, J. (2023). H-TSP: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence, 37*, 9345–9353.
30. Hou, Q., Yang, J., Su, Y., Wang, X., & Deng, Y. (2023). Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The 11th International Conference on Learning Representations*. https://openreview.net/forum?id=6ZajpxqTlQ.

31. Ye, H., Wang, J., Liang, H., Cao, Z., Li, Y., & Li, F. (2024). GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence, 38*, 20284–20292.
32. Zheng, Z., Zhou, C., Tong, X., Yuan, M., & Wang, Z. (2024). UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. In *Advances in Neural Information Processing Systems, 37*, 6081–6125.
33. Jiang, Y., Cao, Z., Wu, Y., & Zhang, J. (2023). Multi-view graph contrastive learning for solving vehicle routing problems. In *Uncertainty in Artificial Intelligence,* pp. 984–994. PMLR
34. Gao, C., Shang, H., Xue, K., Li, D., & Qian, C. (2024). Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. In *The 33rd International Joint Conference on Artificial Intelligence,* pp. 6914–6922.
35. Fang, H., Song, Z., Weng, P., & Ban, Y. (2024). INViT: A generalizable routing problem solver with invariant nested view transformer. In *International Conference on Machine Learning,* pp. 12973–12992. PMLR.
36. Pateria, S., Subagdja, B., Tan, A.-H., & Quek, C. (2021). Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys, 54*(5), 1–35.
37. Levy, A., Platt, R., & Saenko, K. (2019). Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ryzECoAcY7.
38. Drakulic, D., Michel, S., Mai, F., Sors, A., & Andreoli, J.-M. (2023). BQ-NCO: Bisimulation quotienting for efficient neural combinatorial optimization. In *Advances in Neural Information Processing Systems, 36*, 77416–77429.
39. Luo, F., Lin, X., Liu, F., Zhang, Q., & Wang, Z. (2023). Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *Advances in Neural Information Processing Systems, 36*, 8845–8864.
40. Luo, F., Lin, X., Wu, Y., Wang, Z., Xialiang, T., Yuan, M., & Zhang, Q. (2025). Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *The 13th International Conference on Learning Representations*. https://openreview.net/forum?id=TbTJJNjumY.
41. Son, J., Kim, M., Kim, H., & Park, J. (2023). Meta-SAGE: Scale meta-learning scheduled adaptation with guided exploration for mitigating scale shift on combinatorial optimization. In *International Conference on Machine Learning,* pp. 32194–32210. PMLR.
42. Zhou, J., Wu, Y., Song, W., Cao, Z., & Zhang, J. (2023). Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning,* pp. 42769–42789. PMLR.
43. Manchanda, S., Michel, S., Drakulic, D., & Andreoli, J.-M. (2022). On the generalization of neural combinatorial optimization heuristics. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* pp. 426–442. Springer
44. Qiu, R., Sun, Z., & Yang, Y. (2022). DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems, 35*, 25531–25546.
45. Bi, J., Ma, Y., Wang, J., Cao, Z., Chen, J., Sun, Y., & Chee, Y.M. (2022). Learning generalizable models for vehicle routing problems via knowledge distillation. In *Advances in Neural Information Processing Systems, 35*, 31226–31238.
46. Jiang, Y., Cao, Z., Wu, Y., Song, W., & Zhang, J. (2023). Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In *Advances in Neural Information Processing Systems, 36*, 53112–53125.
47. Grinsztajn, N., Furelos-Blanco, D., Surana, S., Bonnet, C., & Barrett, T. (2023). Winner takes it all: Training performant RL populations for combinatorial optimization. In *Advances in Neural Information Processing Systems, 36*, 48485–48509.
48. Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*, 229–256.
49. Zhang, J., Kim, J., O'Donoghue, B., & Boyd, S. (2021). Sample efficient reinforcement learning with REINFORCE. In *Proceedings of the AAAI Conference on Artificial Intelligence, 35*, 10887–10895.
50. Choo, J., Kwon, Y.-D., Kim, J., Jae, J., Hottung, A., Tierney, K., & Gwon, Y. (2022). Simulation-guided beam search for neural combinatorial optimization. In *Advances in Neural Information Processing Systems, 35*, 8760–8772.
51. Perron, L., & Furnon, V.: OR-Tools. Google. https://developers.google.com/optimization/.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Yuxin Pan[1] · Ruohong Liu[2] · Yize Chen[3] · Zhiguang Cao[4] · Fangzhen Lin[1]

✉ Yuxin Pan
yuxin.pan@connect.ust.hk

Ruohong Liu
ruohong.liu@eng.ox.ac.uk

Yize Chen
yize.chen@ualberta.ca

Zhiguang Cao
zhiguangcao@outlook.com

Fangzhen Lin
flin@cs.ust.hk

[1] Hong Kong University of Science and Technology, Hong Kong, China

[2] University of Oxford, Oxford, UK

[3] University of Alberta, Edmonton, Canada

[4] Singapore Management University, Singapore, Singapore