COAgents: Multi-Agent Framework to Learn and Navigate Routing Problems Search Space

Oleksandr Yakovenko[1], Mahdi Mostajabdaveh[1], Cheikh Ahmed[1], Abdullah Ali Sivas[1], Xiaorui Li[1], Zirui Zhou[1], and Mao Kun[2]

[1] Huawei Technologies Canada, 4321 Still Creek Dr, Burnaby, BC, Canada V5C 6S7
[2] Huawei Technologies, China

Abstract. Although Vehicle Routing Problems (VRP) are essential to many real-world systems, they remain computationally intractable at scale due to their combinatorial complexity. Traditional heuristics rely on handcrafted rules for local improvements and occasional *jumps* to escape local minima, but often struggle to generalize across diverse instances. We introduce **COAgents**, a cooperative multi-agent framework that models the search process as a graph: nodes represent solutions, and edges correspond to either local refinements or large perturbations for diversification (i.e., jumps). A *Partial Search Graph* (PSG) is dynamically constructed during search, enabling COAgents to train a Node Selection Agent and a Move Selection Agent to guide intensification, and a Jump Agent to trigger well-timed explorations of new regions. Unlike end-to-end learning approaches, COAgents cleanly separates problem-agnostic search control from compact domain-specific encoding, facilitating adaptability across tasks. Extensive experiments on the CVRP and VRPTW benchmarks show that COAgents remains competitive with several learn-to-search baselines on CVRP and sets a new state of the art among learning-based methods on the more challenging VRPTW instances, reducing the gap to the best-known solutions by 14% at $N = 100$ and 44% at $N = 50$ relative to the strongest neural solver (POMO), and by 21% and 40% respectively relative to ALNS.
Code is available at https://github.com/mahdims/COAgents.

Keywords: Combinatorial Optimization · AI Agents · Partial Search Graph · CVRP · VRPTW.

1 Introduction

Combinatorial optimization problems are central to many real-world decision-making scenarios, including finance, e-commerce, logistics, and manufacturing [27]. Despite breakthroughs in generic solvers, the inherently discrete nature of these problems means that even state-of-the-art solvers often fall short on practical-sized instances, particularly in real-world large-scale logistics and routing applications [26]. Consequently, there is a demand for fast, reliable heuristic methods that can produce high-quality solutions within limited computation

times. Numerous variants of the Vehicle Routing Problem (VRP) offer a solid yet diverse playground for prototyping and testing such OR methods.

Many traditional heuristics for VRPs are built on iterative local search: at each step, they (i) select a current solution, (ii) apply one or more neighborhood moves, and (iii) accept or reject the resulting solution via a heuristic rule. To avoid getting stuck, the VRP solvers augment this process with diversification, randomized constructions, or large perturbations that "jump" to new regions. The local search algorithms are repeatedly answering three core questions: 1) **Which solution** becomes the next incumbent? 2) **Which operator** should be applied next? 3) **When and how** should the search be redirected to escape a local minimum?

Despite their widespread success, classic methods rely on handcrafted, problem-specific rules for each decision, demanding extensive domain expertise and costly trial-and-error tuning [15]. Even minor shifts in problem formulation or input data can invalidate the tuned parameters, forcing expert intervention and comprehensive re-engineering. Moreover, static rule sets cannot accumulate or adapt from past search experience, nor adjust online to evolving instance distributions. This brittleness and maintenance burden motivate our shift to the **COAgents** framework, where learned agents replace fixed heuristics to deliver adaptive, data-driven decision policies with minimal manual tuning.

COAgents is a general multi-agent framework that leverages search history to orchestrate local improvement heuristics via three learned agents, the *Node Selection Agent* (NSA), the *Move Selection Agent* (MSA), and the *Jump Agent* (JA), operating over a *Partial Search Graph* (PSG). A PSG is the visited subgraph of the entire search space, with nodes representing explored solutions and edges representing moves or jumps, thereby encoding the search history. Starting from an initial solution, NSA selects a candidate node and MSA predicts the most promising improvement heuristic, driving local exploration. When repeated failed attempts signal stagnation, JA generates a new solution from scratch based on the search history, steering the search into previously unexplored or unreachable regions. This jump transformation is not a standard operator but a learned restart that escapes local minima and resumes downhill search from a fresh point. Control then returns to NSA and MSA for renewed local search. Figure 1 outlines this top-level loop: NSA, MSA, move application, PSG update, and conditional invocation of JA. The loop is executed until the computational budget (for example, time limits or number of explored nodes) is reached.

Contributions COAgents is the first multi-agent framework that models and learns a combinatorial search space of VRPs via collaborative decision-making over a Partial Search Graph. Our key contributions are: (1) **Novel learned agents.** We introduce two new agents, the Node Selection Agent, which prioritizes promising regions of the search, and the Jump Agent, which predicts learned restarts to escape stagnation—alongside a learned Move Selection Agent, collectively replacing handcrafted rules. (2) **Unified, weight-shared architecture.** Agents share a common neural backbone, yielding data-efficient training, easy transfer to new problem variants, and reduced expert tuning. (3) **Partial**

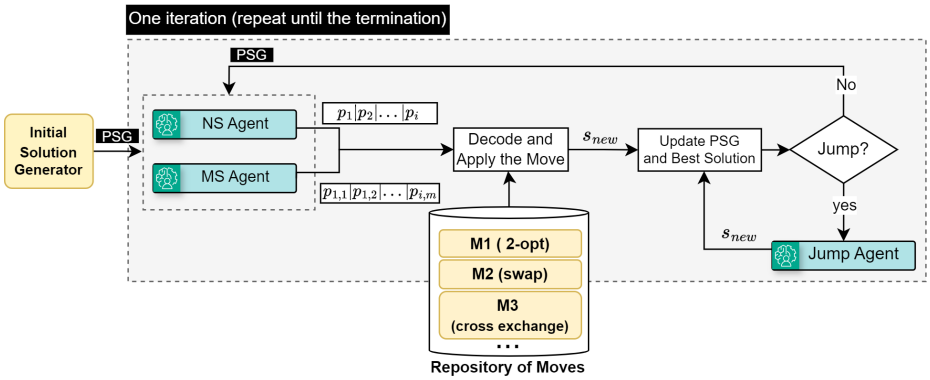

Fig. 1: The COAgents which defines how our three agents are collaborating to orchestrate the search.

Search Graph representation. We formalize the PSG as a general, history-aware state encoding that drives agent decisions. (4) **Solver- and problem-adaptive framework.** COAgents wraps around a existing local search routines and neural solvers to enhance them with adaptive decision policies that seamlessly adapts to different variants of VRP problems. Empirically, COAgents establishes a new state of the art among learning-based methods on VRPTW, reducing the gap to the best-known solutions by up to 44% over the strongest neural baseline.

2 Related Works

Hyper-heuristics are automated methods for selecting or generating heuristics to solve combinatorial problems, first introduced in the 1990s [6]. Interest in hyper-heuristics has grown due to their potential to reduce development overhead and advances in machine learning [7]. They are broadly classified into *selection* and *generation* approaches [2]. Selection hyper-heuristics choose from a set of low-level heuristics to apply to the current solution and then decide whether to accept the new solution [31]. Generation hyper-heuristics, by contrast, create new heuristics by combining building blocks or identifying patterns in existing ones [37]. COAgents differs from these methods as it is a multi-agent framework that not only selects heuristics but also makes decisions on solution selection and diversification jumps.

Learn-to-Search methods. [23] introduced a learn-to-search (L2S) framework for VRPs that iteratively improves an initial solution by selecting customized operators through a reinforcement learning (RL) controller. Building on this idea, [3] proposed a deep RL approach that iteratively modifies local solution components until convergence, successfully addressing problems such as the Capacitated Vehicle Routing Problem (CVRP), scheduling, and Traveling Salesperson Problem (TSP). More recently, [21] formulated operator selection in hyper-heuristics as a Multi-Armed Bandit (MAB) problem, leveraging Thompson Sampling and EXP3 to adaptively handle adversarial and non-stationary

settings. These online MAB algorithms dynamically tune parameters during runtime, and were tested on the Vehicle Routing Problem with Time Windows (VRPTW). COAgents differs fundamentally from these methods by going beyond operator selection. It introduces a multi-agent framework that simultaneously decides which solution to expand, which operator to apply, and when and where to diversify, while explicitly modeling the search space as a graph for richer context.

ALNS operator selection. Recent studies have applied Deep Reinforcement Learning (DRL) to improve operator selection in Adaptive Large Neighborhood Search (ALNS). [11] and [14] used multi-layer perceptron architectures, relying only on high-level search features (e.g., iteration count, temperature, and optimality gap) and ignoring solution-specific details. This omission may limit operator-selection effectiveness. To address this, [13] use a Graph Neural Network (GNN) that embeds the current solution graph to guide operator selection from a large pool (28 destroy and 7 repair operators), achieving consistent gains across five routing problems. Building on these ideas, [30] introduced DR-ALNS, a DRL framework that jointly learns operator selection and dynamically tunes algorithm parameters. Evaluated on the orienteering problem, DR-ALNS outperformed both traditional and Bayesian-tuned ALNS of [22]. Unlike these approaches, COAgents explicitly models the entire search trajectory as a PSG, enabling richer context for its solution selection, operator selection, and jump agents.

3 Problem Statement

We define the *combinatorial solution space* of a VRP variant as a directed graph $G_P^{CO}(\mathcal{S}, E)$ where each node $s \in \mathcal{S}$ represents a distinct solution to the VRP problem P and E is a collection of *neighbourhood moves* for the problem P. An edge $e_{ij} \in E$ connects two nodes s^i and s^j if there exists a move that converts s^i into s^j. An example of a move that corresponds to an edge is the 2-opt heuristic for the Traveling Salesman (Sub-)Problem.

Any global optimum s^* of the problem P is a node in the graph $G_P^{CO}(\mathcal{S}, E)$. However, there is no guarantee that an arbitrary solution s can be transformed into s^* via a sequence of moves. Equivalently, $G_P^{CO}(\mathcal{S}, E)$ may be a disconnected graph. Additionally, for each local optimal solution s_i^*, there exists an induced subgraph $G_P^{CO}(s_i^*; \mathcal{S}, E)$ such that if s is a node of $G_P^{CO}(s_i^*; \mathcal{S}, E)$ then sequences of moves from s are likely to lead to s_i^*. We call these induced subgraphs *basins of attraction*. Hence, the graph traversal is biased, and neighborhood moves may lead to a local optimum even when a path to a global optimum exists.

In practical scenarios, only a subset of all possible moves are available and/or cheap. Therefore, we introduce the *search space* as the subgraph $G_P^{SS}(\mathcal{S}, E_{\mathcal{M}}) \subset G_P^{CO}(\mathcal{S}, E)$, where $\mathcal{M}$ denotes the set of available moves, and $E_{\mathcal{M}}$ is the corresponding set of edges. From now on, we refer to these spaces as G^{SS} and G^{CO}, respectively. Observe that edges of G^{SS} are a subset of edges of G^{CO} and the optimal solution s^* is a node in G^{SS}.

Now, we cast the problem of finding an optimal solution $s^* \in \mathcal{S}$ of a given VRP problem P to

- finding a set of edges E_J, not necessarily corresponding to any legitimate move, such that $G_P^{SS}(\mathcal{M}; \mathcal{S}, E_{\mathcal{M}} \bigcup E_J))$ is connected,
- and finding the shortest path between an initial solution $s^{(0)} \in \mathcal{S}$ and $s^* \in \mathcal{S}$ over the graph $G_P^{SS}(\mathcal{M}; \mathcal{S}, E_{\mathcal{M}} \bigcup E_J))$.

We name the set E_J *jumps*. Note that jumps do not have to improve the objective value; hence, they are not affected by basins of attraction. See fig. 2 for an illustration.

Explicit representation of G^{SS} is not available due to space and time constraints. Instead, we will aim to train our model to implicitly construct the search space during inference. We achieve this by constructing a set of *Partial Search Graphs* and using this set as an additional input to the model. A partial search graph is an *explored* subspace of the entire search space. Nodes represent solutions encountered during the search process. Edges represent either the moves or the jumps. As a result, a PSG captures a trajectory of the search within the search space G^{SS}, showing the parts of the space that have been visited and how they are connected. fig. 3 demonstrates how a PSG evolves between iterations.

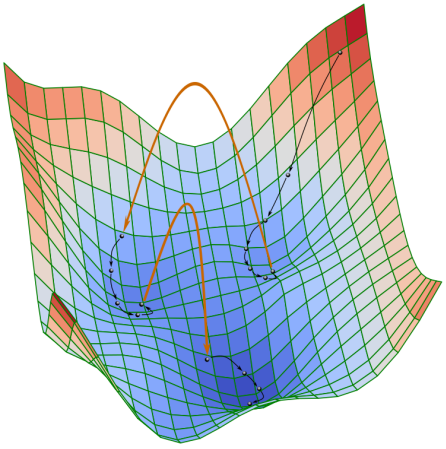

Fig. 2: *Moves* (black) and *jumps* (orange) work together in the search of the global optimum.

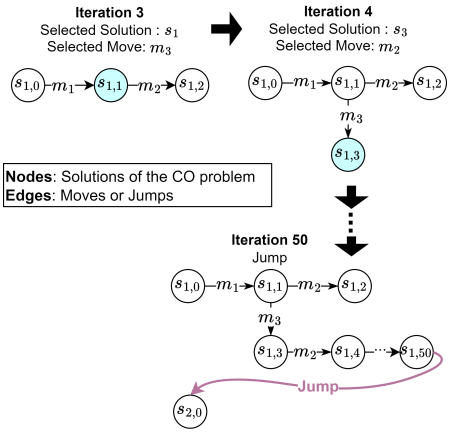

Fig. 3: Evolution of a PSG during the algorithm. s denotes solutions and m are moves.

In this study, we propose a method that learns how to guide the exploration of the search space from previous experiences and how to generate *jumps* during exploration. We will use PSGs to identify patterns in solution exploration and to guide further exploration, improving search efficiency by focusing on unexplored or promising regions. We discuss our approach in the *Methods* section.

4 Solution Method: COAgents

COAgents consists of three agents—the NSA, MSA, and JA, which interact as shown in Fig. 1. Each agent is implemented as a graph-neural-network–based policy trained to maximize long-term progress toward the optimum. In this section, we discuss the data representation, model architecture, and training procedures for these three agents.

4.1 Search History Data Representation

Navigating the non-convex combinatorial solution space of VRPs is a challenging task. To enhance agents' understanding of the solution space, it is crucial to integrate information about the problem, the solution, and the exploration history. While problem and solution encodings are inherently problem-specific, the history exhibits a consistent structure across many variants of combinatorial optimization problems. Next, we discuss the problem-independent representation of the exploration history in detail.

Partial Search Graph The presented framework is built upon the PSG data structure, which represents the search history. As described above, the PSG is essentially a directed, disjointed graph. Each node in the PSG corresponds to a single solution s_i of a combinatorial optimization problem, and each directed edge $(s_i \to s_j)$ indicates that s_j was reached by applying a neighborhood move operator to s_i. The PSG naturally decomposes into connected components—which we call *samples*—each of which explores the neighborhood of a local optimum s_k^*. In turn, each sample is a flattened collection of embeddings of individual solutions s_i together with their connectivity information.

Node Features Each solution embedding is represented by a fixed-size vector $\mathbf{u_i}$ which encodes global solution-specific attributes, including objective value, feasibility, and graph-level aggregates such as the gap relative to its parent in PSG, the number of derived solutions and their objectives, etc. Additionally, each solution is accompanied by a dynamically sized matrix $\mathbf{V_i}$ (which varies across batches) that encapsulates its internal structure. Both u_i and V_i are constructed by concatenating the *structural* and *positional* embeddings of a CO solution s_i. The detailed computation of these embeddings is discussed next, while the complete set of per-problem features utilized in our experiments is provided in **Appendix A**

Input Embeddings Our agents operate on node embeddings of the PSG introduced above. Since all PSG edges (moves and jumps) are provided explicitly, we compute each node's positional embedding u_{pos} via a simple random-walk encoder [28]. The structural embedding of the local and global features is then obtained by

$$\begin{aligned} u_{\text{str}} &= \text{Linear}(x_u) \ \in \mathbb{R}^{d_u}, \\ V_{\text{str}} &= \text{Linear}(x_V) \ \in \mathbb{R}^{d_V}. \end{aligned} \tag{1}$$

Each solution also carries a problem-specific positional embedding V_{pos}; details for its construction in each domain are given in **Appendix B**. Finally, we concatenate structural and positional embeddings:

$$u_{inp} = u_{\text{str}} \frown u_{\text{pos}}, \quad V_{inp} = V_{\text{str}} \frown V_{\text{pos}}. \tag{2}$$

where $u_{inp} \in \mathbb{R}^{d_u + d_{\text{pos}}}$ and $V_{inp} \in \mathbb{R}^{d_V + d'_{\text{pos}}}$.

4.2 Agent Architectures

All the agents are composed of the same core blocks, with only minor differences in functionality. We employ a combination of Gated Graph Convolution (GGCN), Transformer, and problem-specific modules as the universal core block of the agents' networks. GGCN layers propagate and filter messages among each node's immediate neighbors in PSG to capture local structural patterns, while Transformer layers apply global self-attention across all nodes to model long-range dependencies. Given GGCN's graph-specialized capabilities, we opted for a standard Transformer architecture rather than a graph attention mechanism. Figure 4 illustrates the general architecture of agents.

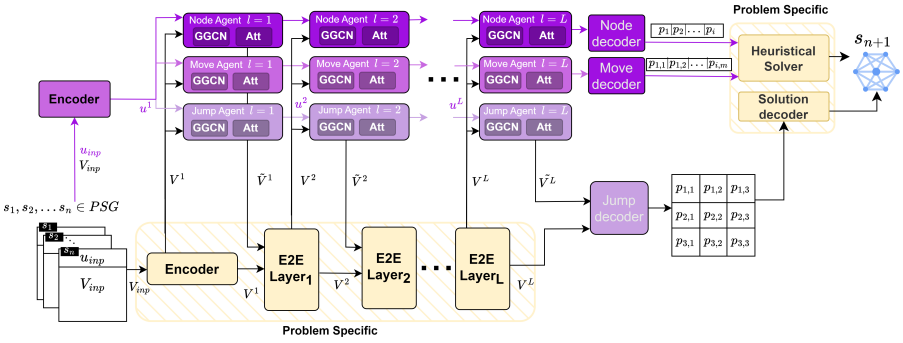

Fig. 4: Agents model architecture

Our network comprises a stack of interleaved *CoreBlocks*, each consisting of a Gated Graph Convolutional Network (GGCN), followed by a Transformer layer, and finally an augmented problem-specific layer. Let $u_i^{(l-1)} \in \mathbb{R}^{d_u}$ be the global embedding of node i and let $v_{ik}^{(l-1)} \in \mathbb{R}^{d_v}$ denote the k-th row of its internal structure matrix $V_{ik}^{(l-1)} = [v_{0,0}^{l-1}, v_{0,1}^{l-1}, \ldots, v_{0,k}^{l-1}, v_{1,0}^{l-1}, \ldots, v_{M,N}^{l-1}]$ where N is the size of the problem and M is the amount of trial solutions in the scope of an agent. The CoreBlock transformation is then:

$$\begin{aligned}
(\hat{u}_i^{(l)}, \hat{V}_i^{(l)}) &= \text{GGCN}(u_i^{(l-1)}, V_i^{(l-1)}, E), \\
(\tilde{u}_i^{(l)}, \tilde{V}_i^{(l)}) &= \text{Transformer}(\hat{u}_i^{(l)}, \hat{V}_i^{(l)}), \\
\bar{V}_i^{(l)} &= \text{E2E}(V_i^{(l-1)}, \tilde{V}_i^{(l)}), \\
u_i^{(l)} &= u_i^{(l-1)} + \tilde{u}_i^{(l)}, \quad V_i^{(l)} = V_i^{(l-1)} + \bar{V}_i^{(l)}.
\end{aligned} \tag{3}$$

4.3 Core Block of COAgents

CoreBlock: Gated Graph Convolution Network. We process the immediate neighborhood of each node in the PSG with the Gated Graph Convolutional Network (GGCN) [1, 29]. The GGCN part learns to gate and propagate information along edges of the PSG. It transforms the input representation (details provided in **Appendix C1**) into $\hat{u}^l$ and $\hat{V}^l$ (the first line of eq. (3)) using a variant of message passing mechanism.

CoreBlock: Transformer. To capture the long–range dependencies across the PSG, we apply a standard Transformer self-attention block (the second line in eq. (3)). The M-by-M attention matrix of the transformer encodes (as it is described in **Appendix C2**) pairwise attention weights across all PSG nodes. Each of the attention weights uniformly scales all the internal features $\tilde{V}_i$ of a PSG node i. Since the inner dimension N (e.g., VRP customers) can be arbitrarily large, we delegate the row-wise attention over $\tilde{V}_i$ to problem-specific modules due to computational overhead. Approximate Transformers implementations, such as Linformer [34] or Performer [5], can mitigate this computational burden to near-linear complexity. Their application, however, would introduce additional errors in a problem-dependent manner, making seamless integration into our generic framework challenging without further problem-specific tuning.

CoreBlock: Problem-Specific End-to-End Modules. The problem-specific layers neither possess any intrinsic knowledge about our framework nor operate at the PSG level. Consequently, they can be trivially outsourced from existing literature and updated to enhance framework efficiency as domain-specific SOTA evolves. The E2E module focuses exclusively on the internal structure of the combinatorial problem $\tilde{V}$ (as it is shown in the last line of eq. (3)), refining it independently from the rest of the framework by leveraging both previous and current local embeddings. It is the framework's responsibility to adapt to the problem-specific layer and inject meaningful correcting dependencies directly into the latent space of an E2E module. **Appendix C3** lists the architectures that we used for each problem type.

4.4 Decoders

Node & Move Selection Decoders Both the Node Selection Agent and Move Selection Agent use the same decoder head. Given each node's final embeddings (u_i, V_i), we first aggregate:

$$z_i \;=\; \frac{1}{N}\sum_{k=1}^{N} \mathrm{GELU}\big(G\,(u_i \frown v_{ik})\big),$$

where $G \in \mathbb{R}^{d_z \times (d_u + d_v)}$. Then, letting $U \in \mathbb{R}^{1 \times d_z}$, $W \in \mathbb{R}^{1 \times (d_z + d_e)}$, and $\{e_m\}_{m=1}^{M} \subset \mathbb{R}^{d_e}$ be move embeddings. Denoting σ as the sigmoid function, we compute

$$p_i \;=\; \sigma\big(U\,z_i\big), \qquad p_{i,m} \;=\; \sigma\big(W\,(z_i \frown e_m)\big),$$

Jump Decoder The Jump Decoder decodes the final local embedding $V_i^{(L)} \in \mathbb{R}^{N \times d_v}$ of a selected solution into a stochastic adjacency matrix $Y \in \mathbb{R}^{N \times N}$. Concretely:

$$Y_i = \left(\frac{V_i^{(L)} W_Q W_K^\top V_i^{(L)\top}}{\sqrt{d_k}}\right)$$

We then mask self-decisions by setting $(Y_i)_{kk} = -\infty$ for all k, and apply a row-wise softmax:

$$P_i = \text{softmax}(Y_i) \ \in [0,1]^{N \times N}.$$

Each row of P_i is a probability distribution over decisions (e.g., edges or arcs) for one component of the solution. Sampling from P_i yields a completely new solution in one shot. In our implementation, we apply this decoder to the local optimal solution from the previous jump; alternatively, it can be run on any or all PSG nodes to generate diverse candidates.

4.5 Training

Let $\mathcal{D} = \{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$ be our dataset of PSGs. Each PSG $\mathcal{G}$ has a set of solutions $\mathcal{S}$ and, for each $s \in \mathcal{S}$, a set of candidate moves $\mathcal{M}$.

We treat *node and move selection* as two binary classification tasks trained jointly. For each solution $s \in \mathcal{S}$, we assign $y_s \in \{0,1\}$, which is 1 if s is closest to the optimal solution in and 0 otherwise. For each move $m \in \mathcal{M}$, we assign $y_{s,m} \in \{0,1\}$, which will be 1 if applying m to s yields the best next candidate in $\mathcal{S}$, and 0 otherwise. Given the predicted probabilities p_s and $p_{s,m}$ (see section *Decoder*), we minimize the joint binary cross-entropy:

$$\begin{aligned} \mathcal{L}_{\text{select}} = &- \sum_i \Big[y_i \log p_i + (1 - y_i) \log(1 - p_i) \Big] \\ &- \sum_{i,m} \Big[y_{i,m} \log p_{i,m} + (1 - y_{i,m}) \log(1 - p_{i,m}) \Big]. \end{aligned} \tag{4}$$

At inference, we sample the node index $i \sim \{p_i\}_i$ and then sample the move $m \sim \{p_{i,m}\}_m$ according to these distributions.

The *Jump Agent* is trained separately. For each CO instance, we precompute a set of K target solutions $\{\delta^{(1)}, \ldots, \delta^{(K)}\}$, comprising the optimal solution and near-optimal neighbors that can be converted to optimal with given heuristics.

Let $\mathcal{I}$ be the set of all atomic decisions (e.g., edges or arcs). The Jump Decoder outputs probabilities $P_i \in [0,1]$ for each $i \in \mathcal{I}$. For each $\delta^{(k)}$, we compute labels $y^{(k)} \in \{0,1\}^{|\mathcal{I}|}$. Then, we pick the closest reference via $k^* = \arg\min_k \frac{1}{2} \sum_{i \in \mathcal{I}} \left| y_i^{(k)} - P_i \right|$, set $y^* = y^{(k^*)}$, and minimize

$$\mathcal{L}_{\text{jump}} = - \sum_{i \in \mathcal{I}} \big[y_i^* \log P_i + (1 - y_i^*) \log(1 - P_i) \big]. \tag{5}$$

At inference, decisions i are sampled according to P to form a complete solution in one shot.

Training stability. Across training, the node and move selection agents exhibited stable convergence because each sampled subgraph contains a single positive move label. The jump agent was more sensitive, since multiple structurally different solutions can be near-optimal and therefore valid jump targets. To reduce label ambiguity, we train the jump model against a set of near-optimal references and select the closest target during training. We did not observe a collapse in the reported runs.

Computational overhead. Let K denote the number of retained nodes per PSG subgraph, $|E_{\text{PSG}}|$ the number of PSG edges, N the problem size (e.g., number of customers), $|\mathcal{M}|$ the number of candidate moves, and d the hidden dimension. Each GGCN layer scales as $\mathcal{O}(|E_{\text{PSG}}|\, d)$, the PSG-level Transformer as $\mathcal{O}(K^2 d)$, the node and move decoders as $\mathcal{O}(Kd + K|\mathcal{M}|d)$, and the jump decoder as $\mathcal{O}(N^2 d)$ per invocation. The dominant per-iteration costs are therefore PSG-level self-attention, quadratic in K, and jump decoding, quadratic in N. Because the PSG grows unboundedly with search length, we cap each subgraph at $K \leq 64$ nodes, evicting the oldest entry once the limit is exceeded, while leaving the number of subgraphs in the history pool unrestricted. This bounds the $\mathcal{O}(K^2 d)$ attention term to a small constant and preserves long-horizon historical context, keeping per-iteration latency low. The $\mathcal{O}(N^2 d)$ jump-decoder cost remains problem-dependent and is the only term that grows with instance size.

5 Experimental Setup and Results

In this section, we evaluate our COAgents framework on two variants of VRP: (1) the Capacitated Vehicle Routing Problem and (2) the Vehicle Routing Problem with Time Windows. Detailed problem definitions are given in **Appendix D**, and the training data generation procedure is described in **Appendix E**. All methods are implemented in Python 3.11 using PyTorch 2.4.1 and trained on a machine with six NVIDIA P100-PCIE 16 GB GPUs under CUDA 12.2. Full reproducibility details, including framework configuration, training data, and per-agent hyperparameters, are provided in **Appendix F**. Runtimes in Tables 1–2 report total wall-clock time over the full test set (following prior work), whereas Table 3 reports average per-instance inference time.

5.1 Experimental Test on the VRPTW

To train our agents, we generate 10K VRPTW instances with 100 customers using a random instance generator. Table 1 compares COAgents against state-of-the-art solvers including heuristics (LKH [8], HGS [33], OR-Tools) and L2C models (MVMoE [38], POMO [20]). We evaluate on the 1K test instances introduced by [38]. All baseline results are taken from [38] unless otherwise specified. COAgents achieves the best performance across all neural solvers, OR-Tools, and ALNS, reducing the optimality gap by 14% ($N = 100$) and 44% ($N = 50$) relative to the strongest neural solver (POMO), and by 21% and 40%, respectively, relative to ALNS.

Table 1: VRPTW: Average objective (Obj.), gap vs. HGS, and runtime on 1k test instances. (H: heuristic, C: learn-to-construct, S: learn-to-search.) Times are total wall-clock time over the full test set.

Method	**Type**	$N = 50$			$N = 100$		
		Obj.	Gap↓	Time↓	Obj.	Gap↓	Time↓
HGS	H	14.51	*	8.4m	24.34	*	19.6m
LKH3	H	14.61	0.66%	5.5m	24.72	1.58%	7.8m
OR-Tools	H	14.92	2.69%	10.4m	25.89	6.30%	20.8m
ALNS (1k)	H	14.97	2.79%	13.4h	25.49	4.68%	7.3d
POMO	C	14.94	2.99%	3s	25.37	4.31%	11s
POMO-MTL	C	15.03	3.64%	3s	25.61	5.31%	11s
MVMoE/4E	C	15.00	3.41%	4s	25.51	4.90%	12s
MVMoE/4E-L	C	15.01	3.50%	3s	25.52	4.93%	11s
COAgents	S	**14.77**	**1.67%**	3.5h	**25.26**	**3.69%**	5.3h

5.2 Experimental Test on CVRP

We used the 10K CVRP instances with 100 customers introduced by [25] to train our agents. Table 2 benchmarks our COAgents against the main families of neural VRP solvers on 10k test instances introduced in [25]: 1. *L2P (learning-to-predict):* CVAE-Opt-DE [9], DPDP [19] (state-of-the-art) 2. *L2C (learning-to-construct):* AM+LCP [17], POMO [20], Sym-NCO [18], POMO+EAS(+SGBS) [10, 4] 3. *L2S (learning-to-search):* NLNS [12], NCE [16], Wu et al. [36], DACT [24] (state-of-the-art). Unless stated otherwise, all baseline numbers are taken from [25]. For a fair comparison, we evaluate inference on a single GPU/CPU using a neural processing unit comparable to an A100 GPU [25]. Following prior work [10, 24, 25], we report: (i) average tour cost, (ii) optimality gap relative to the state-of-the-art solver HGS [33], and (iii) total wall-clock time.

5.3 Comparison with ALNS

In this section, we compare against a strong ALNS baseline that includes all local improvement moves available to COAgents, together with 27 additional destroy-and-repair pairs from [13]. This makes the comparison conservative: ALNS has access to a larger pool of handcrafted operators, while COAgents differs mainly in its learned node selection, move selection, and jump policies. Both algorithms are run for 1000 iterations, and we evaluate the quality of their best-found objective values at ten checkpoints during the search on the 1k VRPTW test set from [38]. COAgents consistently outperforms ALNS. Despite using the same local improvement heuristics, COAgents outperforms ALNS by replacing static destroy-and-repair operators with a learned Jump Agent. When combined with the NSA's solution selection and the MSA's operator choice, this learned diversification provides far more robust and effective guidance than ALNS's handcrafted policies.

Table 2: CVRP: Average Obj., gap vs. HGS, and runtime on 10k test instances. (H: heuristic, C: learn-to-construct, S: learn-to-search, P: learn-to-predict, RL: reinforcement learning, SL: supervised learning, UL: unsupervised learning, DE: dynamic encoding, DP: dynamic programming, AS: active search, BS: beam search.). Times are total wall-clock time over the full test set.

Method	**Type**	**Post**	$N = 20$			$N = 50$			$N = 100$		
			Obj.	Gap↓	Time↓	Obj.	Gap↓	Time↓	Obj.	Gap↓	Time↓
HGS [33]	H	–	6.13	*	2.3m	10.37	*	15m	15.56	*	4.2h
LKH3 [8]	H	–	6.14	0.08%	4.7m	10.38	0.09%	35m	15.65	0.54%	9.8h
ALNS(1k) [13]	H	–	6.19	1.09%	1.3d	10.78	3.93%	5.5d	16.54	6.01%	18.3d
CVAE-Opt-DE	P/UL	DE	6.14	0.16%	1.5h	10.40	0.33%	6.1h	–	–	–
DPDP(1M)	P/SL	DP	–	–	–	–	–	–	15.63	0.41%	1.2d
AM+LCP	C/RL	–	6.15	0.33%	23m	10.52	1.48%	52m	16.00	2.81%	2.1h
Sym-NCO	C/RL	–	–	–	–	–	–	–	15.67	0.70%	7.2h
POMO	C/RL	–	6.14	0.09%	1.7m	10.40	0.30%	11m	15.67	0.70%	7.2h
POMO+EAS	C/RL	AS	6.13	0.04%	6.8m	10.38	0.13%	38m	15.61	0.30%	16h
POMO+EAS+SGBS(short)	C/RL	AS+BS	–	–	–	–	–	–	15.59	0.15%	1d
POMO+EAS+SGBS(long)	C/RL	AS+BS	–	–	–	–	–	–	15.58	0.10%	4.1d
NLNS(5k)	S/RL	–	6.18	0.73%	12m	10.51	1.35%	48m	15.92	2.26%	2.4h
NCE(CROSS)	S/SL	–	6.13	0.00%	11h	10.41	0.42%	2.3d	15.81	1.59%	10.4d
Wu et al.	S/RL	–	–	–	–	10.54	1.72%	4.2h	16.17	3.87%	5h
DACT	S/RL	–	6.13	0.01%	3m	10.38	0.16%	16h	15.74	1.11%	1.7d
NeuOpt(D2A=1,1k)	S/RL	–	6.13	0.03%	2m	10.43	0.61%	12m	15.87	1.94%	28m
COAgents(1k)	S/SL	–	6.18	0.34%	22.2h	10.60	1.44%	1.2d	16.05	2.72%	1.9d

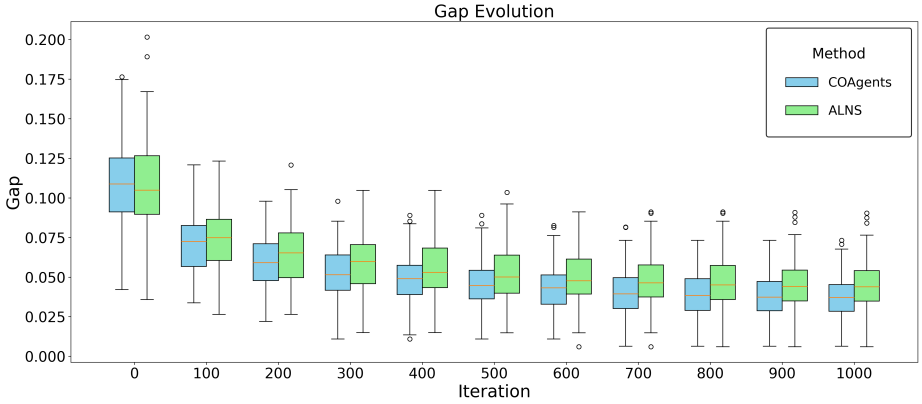

Fig. 5: Comparison of optimality gap for a test set of VRPTW instances.

5.4 Ablation Study

To assess the contribution of each component in COAgents, we conducted an ablation study on VRPTW instances. We evaluated three variants: removing the node selection agent (fallback to hill climbing), removing the move selection agent (fallback to the standard ALNS adaptive mechanism), and removing the jump agent. Table 3 reports the average gap to HGS solutions and runtime. Removing the Jump Agent nearly doubles the gap (from 3.7% to 7.8%), making it the most impactful component of COAgents. Nonetheless, the other agents also contribute to the algorithm's success.

Table 3: Ablation study on VRPTW instances.

Variant	**Avg. Gap ↓**	**Avg. Time, s ↓**
COAgents (full)	**3.691%**	**19**
w/o Node-Selection	3.705%	19
w/o Move-Selection	4.248%	19
w/o Jump Agent	7.762%	52

6 Discussion and Future Directions

Our empirical results demonstrate that COAgents is a compelling alternative to classical heuristics as well as learn-to-construct and learn-to-search methods. On the VRPTW (Table 1), COAgents outperforms all neural baselines, OR-Tools, and ALNS, narrowing the gap to HGS by 14–44% over the strongest neural solver and by 21–40% over ALNS across N=50 and N = 100. On the CVRP (Table 2), it remains competitive with learn-to-search baselines despite relying solely on a compact, problem-specific domain-encoding module. Moreover, in every case COAgents surpasses a pure ALNS implementation, which uses the same local improvement heuristics plus 27 additional destroy–repair pairs, demonstrating that our multi-agent system delivers more robust and effective guidance than ALNS's handcrafted policies.

Strengths. We attribute our superior performance on harder problems, such as the VRPTW, to the PSG, which persistently stores high-quality solutions and feeds them back into our agents. By using this historical information, our agents can drive targeted intensification or diversification more efficiently. In contrast, existing learning-to-search approaches make all their decisions solely based on the current state and their policy weights [13, 21], without ever exploiting accumulated search history. Furthermore, COAgents' modular design isolates domain-specific logic in the E2E blocks, so transferring to a new problem requires only swapping or tuning that module while the GGCN–Transformer core stays unchanged.

Limitations. COAgents' runtime is not as competitive as some baseline methods. This is due to two factors: (1) COAgents uses an iterative improvement search that repeatedly generates and modifies solutions, making it slower than

constructive methods (i.e,. L2C) that build a solution only once, and (2) our heuristic implementations are in Python, whereas traditional heuristics such as LKH3 and HGS are implemented in C and C++, respectively. However, our execution time is comparable to L2S methods and, in some cases (e.g., NCE), even faster. The current implementation is evaluated up to $N = 100$. Scaling to substantially larger or strict real-time VRP settings remains open. The main bottlenecks are PSG-level attention, which grows quadratically with the retained PSG size, and the jump decoder, which constructs an $N \times N$ decision matrix. Sparse PSG attention, more aggressive history pruning, batched move evaluation, and lighter jump decoders are promising directions for reducing this overhead.

Disclosure of Interests. Competing Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bresson, X., Laurent, T.: Residual gated graph convnets. arXiv:1711.07553 (2023). https://doi.org/https://doi.org/10.48550/arXiv.1711.07553
2. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A classification of hyper-heuristic approaches: revisited. Handbook of metaheuristics pp. 453–477 (2019)
3. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization. Advances in neural information processing systems **32** (2019)
4. Choo, J., Kwon, Y.D., Kim, J., Jae, J., Hottung, A., Tierney, K., Gwon, Y.: Simulation-guided beam search for neural combinatorial optimization. Advances in Neural Information Processing Systems **35**, 8760–8772 (2022)
5. Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., Weller, A.: Rethinking attention with performers. In: International Conference on Learning Representations (ICLR) (2021), https://arxiv.org/abs/2009.14794
6. Denzinger, J., Fuchs, M., Fuchs, M.: High performance ATP systems by combining several AI methods. In: Proceedings of the 15th International Joint Conference on Artifical Intelligence - Volume 1. p. 102–107. IJCAI'97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997)
7. Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K.: Recent advances in selection hyper-heuristics. European Journal of Operational Research **285**(2), 405–428 (2020)
8. Helsgaun, K.: An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Roskilde: Roskilde University **12**, 966–980 (2017)
9. Hottung, A., Bhandari, B., Tierney, K.: Learning a latent search space for routing problems using variational autoencoders. In: International Conference on Learning Representations (2021)
10. Hottung, A., Kwon, Y.D., Tierney, K.: Efficient active search for combinatorial optimization problems. International Conference on Learning Representations (2021)
11. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. In: Giacomo, G.D., et al. (eds.) ECAI 2020

– 24th European Conference on Artificial Intelligence, Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 443–450. IOS Press (2020). https://doi.org/10.3233/FAIA200124, https://doi.org/10.3233/FAIA200124
12. Hottung, A., Tierney, K.: Neural large neighborhood search for routing problems. Artificial Intelligence **313**, 103786 (2022)
13. Johnn, S.N., Darvariu, V.A., Handl, J., Kalcsics, J.: A graph reinforcement learning framework for neural adaptive large neighbourhood search. Computers & Operations Research **172**, 106791 (2024)
14. Kallestad, J., Hasibi, R., Hemmati, A., Sörensen, K.: A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems. European Journal of Operational Research **309**(1), 446–468 (2023)
15. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. Advances in neural information processing systems **30** (2017)
16. Kim, M., Park, J., Park, J.: Learning to cross exchange to solve min-max vehicle routing problems. In: The Eleventh International Conference on Learning Representations (2023)
17. Kim, M., Park, J., et al.: Learning collaborative policies to solve np-hard routing problems. Advances in Neural Information Processing Systems **34**, 10418–10430 (2021)
18. Kim, M., Park, J., Park, J.: Sym-nco: Leveraging symmetricity for neural combinatorial optimization. Advances in Neural Information Processing Systems **35**, 1936–1949 (2022)
19. Kool, W., van Hoof, H., Gromicho, J., Welling, M.: Deep policy dynamic programming for vehicle routing problems. In: International conference on integration of constraint programming, artificial intelligence, and operations research. pp. 190–213. Springer (2022)
20. Kwon, Y.D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: Pomo: Policy optimization with multiple optima for reinforcement learning. Advances in Neural Information Processing Systems **33**, 21188–21198 (2020)
21. Lagos, F., Pereira, J.: Multi-armed bandit-based hyper-heuristics for combinatorial optimization problems. European Journal of Operational Research **312**(1), 70–91 (2024)
22. Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: Smac3: A versatile bayesian optimization package for hyperparameter optimization. Journal of Machine Learning Research **23**(54), 1–9 (2022), http://jmlr.org/papers/v23/21-0888.html
23. Lu, H., Zhang, X., Yang, S.: A learning-based iterative method for solving vehicle routing problems. In: International conference on learning representations (2019)
24. Ma, Y., Li, J., Cao, Z., Song, W., Zhang, L., Chen, Z., Tang, J.: Learning to iteratively solve routing problems with dual-aspect collaborative transformer. Advances in Neural Information Processing Systems **34**, 11096–11107 (2021)
25. Ma, Y., Cao, Z., Chee, Y.M.: Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. Advances in Neural Information Processing Systems **36**, 49555–49578 (2023)
26. Mostajabdaveh, M., Salman, F.S., Gutjahr, W.J.: A branch-and-price algorithm for fast and equitable last-mile relief aid distribution. European Journal of Operational Research **324**(2), 522–537 (2025)
27. Petropoulos, F., Laporte, G., Aktas, E., Alumur, S.A., Archetti, C., Ayhan, H., Battarra, M., Bennell, J.A., Bourjolly, J.M., Boylan, J.E., Breton, M., Canca,

D., Charlin, L., Chen, B., Cicek, C.T., Jr, L.A.C., Currie, C.S., Demeulemeester, E., Ding, L., Disney, S.M., Ehrgott, M., Eppler, M.J., Erdoğan, G., Fortz, B., Franco, L.A., Frische, J., Greco, S., Gregory, A.J., Hämäläinen, R.P., Herroelen, W., Hewitt, M., Holmström, J., Hooker, J.N., Işık, T., Johnes, J., Kara, B.Y., Özlem Karsu, Kent, K., Köhler, C., Kunc, M., Kuo, Y.H., Letchford, A.N., Leung, J., Li, D., Li, H., Lienert, J., Ljubić, I., Lodi, A., Lozano, S., Lurkin, V., Martello, S., McHale, I.G., Midgley, G., Morecroft, J.D., Mutha, A., Oğuz, C., Petrovic, S., Pferschy, U., Psaraftis, H.N., Rose, S., Saarinen, L., Salhi, S., Song, J.S., Sotiros, D., Stecke, K.E., Strauss, A.K., İstenç Tarhan, Thielen, C., Toth, P., Woensel, T.V., Berghe, G.V., Vasilakis, C., Vaze, V., Vigo, D., Virtanen, K., Wang, X., Weron, R., White, L., Yearworth, M., Yıldırım, E.A., Zaccour, G., Zhao, X.: Operational research: methods and applications. Journal of the Operational Research Society **75**(3), 423–617 (2024)

28. Rampášek, L., Galkin, M., Dwivedi, V.P., Luu, A.T., Wolf, G., Beaini, D.: Recipe for a general, powerful, scalable graph transformer. In: Advances in Neural Information Processing Systems. vol. 35, pp. 14501–14515 (2022), https://arxiv.org/abs/2205.12454
29. Rampášek, L., Galkin, M., Dwivedi, V.P., Luu, A.T., Wolf, G., Beaini, D.: Recipe for a general, powerful, scalable graph transformer. In: Proceedings of the 36th International Conference on Neural Information Processing Systems. NIPS '22, Curran Associates Inc., Red Hook, NY, USA (2024)
30. Reijnen, R., Zhang, Y., Lau, H.C., Bukhsh, Z.: Online control of adaptive large neighborhood search using deep reinforcement learning. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 34, pp. 475–483 (2024)
31. Soria-Alcaraz, J.A., Ochoa, G., Sotelo-Figeroa, M.A., Burke, E.K.: A methodology for determining an effective subset of heuristics in selection hyper-heuristics. European Journal of Operational Research **260**(3), 972–983 (2017). https://doi.org/https://doi.org/10.1016/j.ejor.2017.01.042
32. Vidal, T.: Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. Computers & Operations Research **140**, 105643 (2022). https://doi.org/https://doi.org/10.1016/j.cor.2021.105643, https://www.sciencedirect.com/science/article/pii/S030505482100349X
33. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers & operations research **40**(1), 475–489 (2013)
34. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768 (2020), https://arxiv.org/abs/2006.04768
35. Wouda, N.A., Lan, L., Kool, W.: PyVRP: a high-performance VRP solver package. INFORMS Journal on Computing **36**(4), 943–955 (2024). https://doi.org/10.1287/ijoc.2023.0055, https://doi.org/10.1287/ijoc.2023.0055
36. Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A.: Learning improvement heuristics for solving routing problems. IEEE transactions on neural networks and learning systems **33**(9), 5057–5069 (2021)
37. Ye, H., Wang, J., Cao, Z., Berto, F., Hua, C., Kim, H., Park, J., Song, G.: ReEvo: Large language models as hyper-heuristics with reflective evolution. In: Proceedings of the 38th Conference on Neural Information Processing (NeurIPS 2024). pp. 10–15. Vancouver, Canada (2024)

38. Zhou, J., Cao, Z., Wu, Y., Song, W., Ma, Y., Zhang, J., Xu, C.: Mvmoe: Multi-task vehicle routing solver with mixture-of-experts. In: International Conference on Machine Learning (2024)

A Problem-Specific Feature Definitions

Below we summarize the global and local features used for each problem domain in our experiments.

Vehicle Routing Problem variants Global feature vector x_u for each CVRP solution includes:

- Objective value of the solution.
- Number of vehicles.
- Number of customers.
- Vehicle capacity.
- Improvement in objective relative to the parent solution in the PSG.
- Sum of objective values of offspring solutions (graph-level aggregate).
- Sum of squares of objective values of offspring solutions (graph-level aggregate).
- Number of offspring solutions (graph-level aggregate).

Local structural matrix x_V for each VRP solution is a matrix of customer coordinates, demand, time windows (if applicable), and their positions in the route, encoded using cyclic positional encodings [24].

B Solutions Positional Embeddings

Positional Feature Embedding for CVRP and VRPTW. Cyclic positional embeddings [24] were adopted to implicitly encode the edges of VRP solution. For each dimension d with frequency ω_d,

$$v_{pos} = \begin{cases} \sin\Big(\omega_d \cdot \Big|\Big(z(i) \bmod \frac{4\pi}{\omega_d}\Big) - \frac{2\pi}{\omega_d}\Big|\Big), & \text{if } d \text{ is even,} \\ \cos\Big(\omega_d \cdot \Big|\Big(z(i) \bmod \frac{4\pi}{\omega_d}\Big) - \frac{2\pi}{\omega_d}\Big|\Big), & \text{if } d \text{ is odd.} \end{cases} \tag{6}$$

where $z(i)$ represents an evenly spaced position for the nodes over a cycle.

C Architecture of a Core Block

C.1 CoreBlock: GGCN

Define the outgoing neighbor set $\mathcal{N}^+(i)$ and incoming set $\mathcal{N}^-(i)$.

First, we merge global and local features and compute the *aggregated solution embeddings* as follows

$$z_i^{(l)} = \frac{1}{N} \sum_{k=1}^{N} \mathrm{GELU}\big(\mathbf{G}\,(u_i^{(l-1)} \frown v_{ik}^{(l-1)})\big) \quad \in \mathbb{R}^{d_z}$$

where N is the size of a problem and $G \in \mathbb{R}^{d_z \times (d_u + d_v)}$). Next, we compute edge-conditioned gates in both directions:

$$\text{gate}_{ij}^{(l)} = \tanh\Big(\frac{1}{|\mathcal{N}^+(i)|} \sum_{j \in \mathcal{N}^+(i)} \big(\mathbf{W}_K\, z_i^{(l)} + \mathbf{W}_Q\, z_j^{(l)} + \mathbf{W}_E\, e_{ij}\big)\Big),$$

$$\text{gate}_{ji}^{(l)} = \tanh\Big(\frac{1}{|\mathcal{N}^-(i)|} \sum_{j \in \mathcal{N}^-(i)} \big(\mathbf{W}_K\, z_j^{(l)} + \mathbf{W}_Q\, z_i^{(l)} + \mathbf{W}_E\, e_{ji}\big)\Big),$$

where $\mathbf{W}_K, \mathbf{W}_Q, \mathbf{W}_E$ are learnable keys, queries and e_{ij} encodes the heuristic on edge $(i \to j)$.

We then update each hidden feature row via gated message passing:

$$h_{ik}^{(l)} = h_{ik}^{(l-1)} + \frac{1}{|\mathcal{N}^+(i)|} \sum_{j \in \mathcal{N}^+(i)} \text{gate}_{ij}^{(l)}\, h_{jk}^{(l-1)} + \frac{1}{|\mathcal{N}^-(i)|} \sum_{j \in \mathcal{N}^-(i)} \text{gate}_{ji}^{(l)}\, h_{jk}^{(l-1)}.$$

Here $h^{(l-1)} = V^{(l-1)} H$ projects the local matrix via H.

Finally, we fuse these updates back into the node embeddings with residual connections:

$$\hat{u}_i^{(l)} = u_i^{(l-1)} + \mathbf{W}_u \Big(z_i^{(l)} + \frac{1}{N} \sum_{k=1}^{N} h_{ik}^{(l)}\Big),$$

$$\hat{v}_{ik}^{(l)} = v_{ik}^{(l-1)} + \mathbf{W}_v\, h_{ik}^{(l)},$$

where $\mathbf{W}_u$ and $\mathbf{W}_v$ are learnable projections and $\hat{u}$ and $\hat{V} = [\hat{v}_{ik}]$ are output representations of global and local embeddings, respectively.

C.2 CoreBlock: Transformer

The attention and features matrices are treated separately. The attention is computed from global $\hat{u}$ and internal $\hat{V}$ features using condensed solution-wide representations:

$$z_i^{(l)} = \frac{1}{N} \sum_{k=1}^{N} \text{GELU}\big(\mathbf{G}\, (\hat{u}_i^{(l-1)} \frown \hat{v}_{ik}^{(l-1)})\big) \quad \in \mathbb{R}^{d_z}$$

$$A^{(l)} = \text{softmax}\Big(\frac{(Z^{l-1}Q)\,(Z^{(l-1)}K)^\top}{\sqrt{D}}\Big) \quad \in \mathbb{R}^{N \times N},$$

where $Z \in \mathbb{R}^{N \times d_z}$ is the matrix of all z_i in a PSG, $Q \in \mathbb{R}^{D \times d_z}$ is the queries matrix and $K \in \mathbb{R}^{D \times d_z}$ is the matrix of keys.

In contrast, internal features $\hat{V}$ are updated in a more granular manner using the unrolled attention-weighted summation over all M solutions in the PSG:

$$\tilde{V}_{\text{att}}^{(l)} = A^{(l)} \otimes \left(\hat{V}^{(l-1)} W_V\right) \quad \in \mathbb{R}^{M \times N \times d_v},$$

for which $A \in \mathbb{R}^{M \times 1 \times M}$ is broadcasted over N internal dimensions and $W_V \in \mathbb{R}^{1 \times d_v \times d_v}$ is just a learnable linear projector into the layers latent space.

Eventually, the $\tilde{V}$ is updated in a rather typical manner for transformer layers:

$$\tilde{V}^{(l)} = \hat{V}^{(l-1)} + \tilde{V}_{\text{att}}^{(l)},$$

$$\tilde{V}^{(l)} = \text{LayerNorm}\big(\tilde{V}^{(l)} + \text{FFN}(\tilde{V}^{(l)})\big),$$

where FFN is a two-layer feed-forward network with GELU activation, and LayerNorm denotes layer normalization. This block aggregates information globally — irrespective of graph distance — while residual connections and normalization stabilize training.

C.3 CoreBlock: Problem-Specific End-to-End Modules

For CVRP, we adopt the dual-aspect Transformer architecture from [24], which combines global instance-level representations with a local GCN/GAN module. We use the same for VRPTW.

C.4 Beam search

The beam search algorithm is described as algorithm 1. Constrained beam search operates on the initial set of sequences denoted as S_0. For each item in a sequence, probabilities P are computed using the *Jump* Agent. Within each route, displacement probabilities are averaged to estimate the route's overall optimality. The three least probable routes are identified and discarded. The remaining routes, retained from S_0, serve as the starting point for the constrained beam search.

Beam search begins with an empty set of completed candidates F. At each iteration, every unassigned unique consumer is systematically added to every candidate sequence in the pool S, each of length $|S_0| + iteration$. To account for route splits or merges, the depot is also included among the possible consumers. Such extension generates up to $N_{Beam} \times |s \notin \text{Routes}|$ new candidate sequences of length $|S_0| + iteration + 1$. To maintain the constant beam search width, $N_{Beam} - |F|$ candidates are randomly selected from this expanded pool, using the probability distribution in P as the sampling weight. When any of the new candidate solution S_i incorporates all the consumers, it is transferred from the candidate pool S to the pool of finished solutions F.

The iteration process terminates when the objective value of the worst completed solution, $\max F$, exceeds the best theoretically achievable score among all incomplete candidates, $\min S$. The algorithm returns the solution from F with the most favorable objective value.

Algorithm 1 Constrained Beam Search

```
S_0 {Initial sequence}
Routes ← max P(S_0) {Most probable routes}
for i = 1 to N_Beam do
  S_i ← ∀s ∈ S_0   s ∉ Routes
end for
F ← ∅
while S ≠ ∅ and min S < max F do
  B ← ∅
  for i = 1 to |S| do
    for j = 1 to |Routes| do
      B_ij ← S_i ∪ Routes_j
    end for
  end for
  S ← Sample(B, P, N_Beam − |F|)
  for i = 1 to |S| do
    if ∀r ∈ Routes   r ∈ S_i then
      F ← F ∪ S_i
    end if
  end for
end while

return min F
```

D Benchmark Problems

Capacitated Vehicle Routing Problem (CVRP) is defined as follows. Let $G = (V, E)$ be a complete graph with vertex set $V = \{0\} \cup C$, where 0 denotes the depot and C the set of customers. Each customer $i \in C$ has demand δ_i, and each arc $(i \to j) \in E$ carries weight d_{ij}, the Euclidean distance between i and j. A solution is a collection of routes—each route is a cycle beginning and ending at the depot, visiting its assigned customers exactly once, subject to the vehicle capacity constraint. The objective is to minimize the sum of all route distances.

Vehicle Routing Problem with Time Window (VRPTW) is a common extension of the CVRP. Each customer i has a time window $[e_i, \ell_i]$ that specifies the earliest and latest allowable service times. A vehicle may begin service at i no earlier than e_i and no later than ℓ_i; otherwise, the delivery is infeasible.

E Training Dataset Generation

Node & Move Selection Dataset For each problem instance in training set, we first compute a near-optimal solution s^* using a state-of-the-art solver (e.g., HGS for VRP variants [32, 35]). We then perturb s^* to generate a set $\mathcal{S}^P$ of solutions whose objective values lie within controlled gaps of 0.1 %, 1 %, 2 %, 3 %, 4 %, 5% and 10 % from s^*. To ensure diversity in the perturbed nodes while maintaining the target gap, we add a sufficiently large constant penalty on the

decisions present in s^* (denoted $I(s^*)$) to the objective function, and repeatedly apply random moves from the move set $\mathcal{M}$ until the perturbed solution meets the desired gap threshold. Next, we restore the original objective function and apply random moves to each perturbed solution s^P, keeping only those moves that successfully improve the objective. We repeat this "perturb-and-improve" process many times per instance to collect a large set of training PSGs.

To form training samples, we then randomly sample subgraphs from these PSGs, selecting at most one move pair from each independent PSG. Let $\mathcal{G}^s = \{\mathcal{G}_1^s, \mathcal{G}_2^s, \ldots, \mathcal{G}_k^s\}$ be the set of sampled subgraphs for one problem instance. Within each $\mathcal{G}_i$, we label every candidate move as 0, except for the single move that brings us closest to the original solution s^* that move is labeled 1. Any sample in which multiple moves tie for the best improvement is discarded, ensuring exactly one positive move per subgraph. Fig. 6 illustrate an example on how a training sample is generated using the training PSGs.

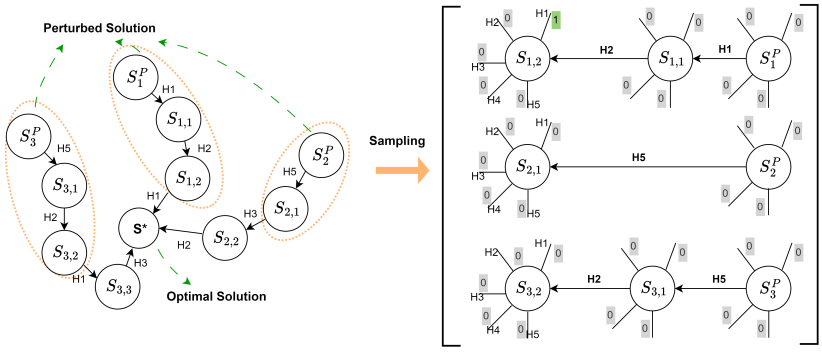

Fig. 6: Node & Move Selection training sample. **Left:** Three PSGs converging from distinct perturbed solutions. **Right:** A training sample with three sub-graphs sampled from the PSGs on the left with labels.

Jump Agent Dataset For our jump model, using the same training instances, we generate multiple trajectories per instance by running ALNS [13] from different random starts until convergence to a local optimum. We record each trajectory as a PSG (a sequence of solutions s_k and moves m). Then, we sample a subgraph $\mathcal{G}$ as in the node–move dataset and pair each solution $s_k \in \mathcal{G}^s$ with the known global optimum as the prediction target.

F Reproducibility

This appendix documents the configuration used to produce all results in the main paper. Framework-level settings shared across agents are listed in Table 4; per-agent training hyperparameters appear in Table 5. Both the Selection Agents and the Jump Agent use a step learning-rate scheduler with step size 100 and $\gamma = 0.998$. All values are held fixed across VRPTW and CVRP.

Table 4: Framework-level configuration shared across all agents.

Component	**Setting**
PSG subgraph cap	64 nodes
Pruning rule	FIFO; discard oldest node on overflow
History pool size	Unbounded
Inference budget	1,000 iterations
VRPTW training set	10K synthetic instances, 100 customers
CVRP training set	10K instances from the NeuOpt benchmark
Hardware	6× NVIDIA P100-PCIE (16 GB)

Table 5: Training hyperparameters for the Node/Move Selection Agents and the Jump Agent.

Hyperparameter	**Selection Agents**	**Jump Agent**
Optimization		
Initial learning rate	10^{-4}	10^{-4}
LR scheduler (step size, γ)	(100, 0.998)	(100, 0.998)
Batch size	48	16
Training iterations	50K	25K
CoreBlock — GGCN		
Hidden dimension	128	128
CoreBlock — Transformer		
Attention heads	16	16
Hidden dimension	256	256
Decoder		
Attention heads	—	4