

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 CHAIN-OF-CONTEXT LEARNING: DYNAMIC CON- STRAINT UNDERSTANDING FOR MULTI-TASK VRPs

Anonymous authors

Paper under double-blind review

## ABSTRACT

Multi-task Vehicle Routing Problems (VRPs) aim to minimize routing costs while satisfying diverse constraints. Existing solvers typically adopt a unified reinforcement learning (RL) framework to learn generalizable patterns across tasks. However, they often overlook the constraint and node dynamics during the decision process, making the model fail to accurately react to the current context. To address this limitation, we propose *Chain-of-Context Learning* (CCL), a novel framework that progressively captures the evolving context to guide fine-grained node adaptation. Specifically, CCL constructs step-wise contextual information via a Relevance-Guided Context Reformulation (RGCR) module, which adaptively prioritizes salient constraints. This context then guides node updates through a Trajectory-Shared Node Re-embedding (TSNR) module, which aggregates shared node features from all trajectories' contexts and uses them to update inputs for the next step. By modeling evolving preferences of the RL agent, CCL captures step-by-step dependencies in sequential decision-making. We evaluate CCL on 48 diverse VRP variants, including 16 in-distribution and 32 out-of-distribution (with unseen constraints) tasks. Experimental results show that CCL performs favorably against the state-of-the-art baselines, achieving the best performance on all in-distribution tasks and the majority of out-of-distribution tasks.

## 1 INTRODUCTION

The vehicle routing problem (VRP) seeks to determine optimal routes for a fleet of vehicles to serve a set of customers while satisfying operational constraints such as vehicle capacity. Efficiently solving VRPs can significantly reduce transportation costs and improve service quality, making it a critical task in logistics and supply chain management (Toth and Vigo, 2014; Konstantakopoulos et al., 2022; Garaix et al., 2010; Dondo et al., 2011). Traditional approaches (Perron and Furnon; Lin and Kernighan, 1973; Vidal et al., 2020) often rely on heuristic-based solvers, such as LKH (Lin and Kernighan, 1973) and HGS (Vidal et al., 2020). While effective in certain settings, these methods are computationally intensive and typically require extensive hand-crafted rules to adapt to different problem variants. Recently, neural networks have emerged as a promising alternative due to their flexibility and ability to learn generalizable policies (Joshi et al., 2019; Kool et al., 2018; Kwon et al., 2020; Wu et al., 2021; Ma et al., 2023; Sun and Yang, 2023; Bengio et al., 2021; Bogyrbayeva et al., 2024; Hottung and Tierney, 2020; Hottung et al., 2021; Xin et al., 2021; Chalumeau et al., 2023; Ma et al., 2023; Chen et al., 2023a). These neural solvers are trained offline using historical or synthetically generated instances, enabling fast inference at test time for a given VRP variant. However, real-world VRPs often involve more complex and diverse constraints beyond vehicle capacity, leading to multi-task VRPs, where each task involves a different combination of constraints. This makes the neural VRP solvers for a specific single task less effective due to the massive yet necessary re-training or fine-tuning.

In multi-task VRPs, the commonly studied constraints include *backhaul demands* (B) (Zong et al., 2022; Kong et al., 2024), *open routes* (O) (Tyasnurita et al., 2024; Bezerra et al., 2023), *route duration limits* (L) (Oliveira et al., 2025), *customer time windows* (TW) (Zhang et al., 2022; Lin et al., 2021), *mixed backhaul* (MB) (Wang et al., 2024), and *multi-depot settings* (MD) (Karakatić and Podgorelec, 2015). To tackle the multi-task scenario, a number of neural models (Liu et al., 2024; Zhou et al., 2024a; Berto et al., 2024a; Li et al., 2025) have been developed using a unified reinforcement learning (RL) framework, which encodes both constraint information and node attributes into static

embeddings. The decoding stage follows a Markov Decision Process (MDP). For a given VRP task, the model combines a global context, such as current time or remaining vehicle capacity, with these static node embeddings to select the next node. Since node priorities change across decoding, static node embeddings, which remain fixed across decoding steps, cannot reflect this dynamic property. While the context is updated, such a misaligned context-node pair may lead to inaccurate state estimation, thereby misjudging the next decision.

To overcome this limitation, we argue that *constraint requirements should be explicitly integrated into the step-wise context and used to adaptively refine node-level representations*. In single-task VRPs, dynamic decoding mechanisms, such as the removal of visited nodes (Xin et al., 2020), have been used to reflect evolving routing decisions. While conceptually related, extending such a mechanism to multi-task settings introduces three unique challenges: (1) The importance of each constraint may vary across decoding steps, *e.g.*, the open route constraint becomes more critical as a vehicle’s sub-route nears completion. Applying uniform attention across all constraints at each step, such as the one in (Li et al., 2025), limits the model’s ability to focus on the most important ones. Moreover, performing RL-based node refinement into VRPs poses issues with efficiency and sequential dependencies. On the one hand, (2) multi-trajectories involve different contexts at each step, and re-embedding the nodes for each context (*e.g.*, (Xin et al., 2020)) causes a heavy computational burden. On the other hand, (3) multi-task VRP solvers (Li et al., 2025; Berto et al., 2024a; Zhou et al., 2024a) typically refine the node representations at step- $i$  using only the initial (step-0) embeddings and the current context. A misaligned state may fail to capture the status of the current decoding step, thereby limiting the model’s ability to accurately represent the Markov property, which is essential for coherent sequential decision-making.

To address these challenges, we propose *Chain-of-Context Learning* (CCL), a novel framework for constraint-aware, step-wise reasoning in multi-task VRPs. Specifically, to tackle *Challenge (1)*, CCL constructs step-wise contextual information using a Relevance-Guided Context Reformulation (RGCR) module. RGCR combines constraint-specific attributes (*e.g.*, remaining capacity for B and current time for TW), and adaptively emphasizes each constraint according to its similarity to the current node embedding. To address *Challenge (2)*, we design a Trajectory-Shared Node Re-embedding (TSNR) module, which enables efficient refinement of node features. TSNR employs shared node embeddings as queries and uses multi-trajectory contexts as keys and values in a multi-head attention mechanism, avoiding redundant re-embedding for each trajectory. To resolve *Challenge (3)*, TSNR updates node embeddings in the environment and feeds them as queries to the next decoding step. This design allows CCL to capture sequential dependencies and model the evolution of node importance over time.

We evaluate CCL on the combinations of six core constraints (B, O, L, TW, MB, MD), resulting in 16 in-distribution and 32 out-of-distribution multi-task VRP variants. Our contributions are summarized as follows: (1) *Conceptually*, we correct a misalignment in prior VRP formulation, by learning step-wise context and node status for a more accurate state. (2) *Methodologically*, we propose RGCR to integrate constraint requirements into the step context, along with TSNR to facilitate effective refinement and capture sequential dependencies. (3) *Experimentally*, our method achieves superior results on all seen (in-distribution) tasks and the majority of unseen (out-of-distribution) tasks.

## 2 PRELIMINARIES

**Problem Definition.** The classical vehicle routing problem (VRP) aims to determine a set of sub-routes that minimize total travel cost while satisfying customer demands. In each sub-route, a vehicle departs from the depot, delivers goods to a subset of customers, and returns to the depot, subject to the following standard constraints: (1) each sub-route starts and ends at the depot; (2) each customer is visited exactly once; and (3) the total demand on each sub-route does not exceed the vehicle’s capacity. Formally, the problem is defined on a graph where the set of nodes  $\mathbf{V} = \{v_0, v_1, \dots, v_N\}$  represents the depot ( $v_0$ ) and  $N$  customer locations. Each customer node  $v_i$  is associated with a demand value  $\delta_i \in [0, Q]$ , where  $Q$  denotes the vehicle’s capacity.

Following (Berto et al., 2024b), we extend this classical setting by considering six additional constraints commonly studied in multi-task VRPs: (1) **Open Routes (O)**: In problems like OVRP, this constraint is denoted by a binary flag  $o \in \{0, 1\}$ , which defines whether a route must return to the depot. When  $o = 1$ , vehicles are not required to return to the depot after completing their route. (2)

108 **Duration Limits (L):** In problems like VRPL, this constraint enforces a maximum route length  $l$  to  
 109 promote workload balancing across sub-routes. (3) **Backhaul Demands (B):** In problems like VRPB,  
 110 customer nodes are classified into linehaul and backhaul types. The vehicle must first complete all  
 111 linehaul deliveries (goods from depot to customers) before collecting backhaul items (goods from  
 112 customers to depot). Each customer has two types of demand:  $\delta_i^l$  for linehaul and  $\delta_i^b$  for backhaul,  
 113 with  $\delta_i^l, \delta_i^b \in [0, Q]$ . (4) **Mixed Backhaul (MB):** In problems like VRPMB, this constraint relaxes the  
 114 linehaul-before-backhaul requirement, allowing both types of customers to appear in any order along  
 115 the route, while still respecting the capacity constraint. (5) **Time Windows (TW):** In problems like  
 116 VRPTW, each customer  $v_i$  is associated with an early time  $t_i^e$ , a late time  $t_i^l$ , and a service duration  $t_i^s$ .  
 117 Vehicles must arrive before  $t_i^l$  and wait if they arrive earlier than  $t_i^e$ , ensuring service occurs within the  
 118 specified window. (6) **Multi-Depot (MD):** In problems like MDVRP, this constraint allows multiple  
 119 depot nodes instead of a single depot. Vehicles may begin their routes from any depot in the set,  
 120 introducing additional complexity in depot assignment.

121 **Markov Decision Process for Multi-Task VRPs.** The multi-task VRP solver acts as a single agent,  
 122 using the encoder-decoder architecture as its policy network. The policy generates a node sequence  
 123 autoregressively, using a Markov Decision Process (MDP) environment

$$124 \quad \mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}). \quad (1)$$

126 (1) **State ( $\mathcal{S}$ )** consists of node embeddings and context embeddings. During decoding, following (Liu  
 127 et al., 2024; Zhou et al., 2024a; Berto et al., 2024a;b; Li et al., 2025), the model explores from diverse  
 128 starting points, forming multiple trajectories in parallel. Each trajectory maintains its own context  
 129 (e.g., current time and used capacity), while all trajectories share the same set of node embeddings.

130 (2) **Action ( $\mathcal{A}$ )** corresponds to selecting the next node to visit. The policy network takes the current  
 131 state as input and generates a trajectory-specific probability distribution over feasible nodes, allowing  
 132 each trajectory to independently select its next action based on the predicted probabilities.

134 (3) **Transition ( $\mathcal{P}$ )** updates the environment after a node is selected. This modifies the environmental  
 135 routing information, such as the vehicle’s current position and remaining capacity. The updated  
 136 environment then defines the next context embedding and continues the decision process.

137 (4) **Reward ( $\mathcal{R}$ )** is defined as the negative total route length. After all nodes are visited, each trajectory  
 138 computes its own negative route length as the reward. These rewards, together with the action  
 139 log-probabilities produced by the policy network, are aggregated to form a single training objective.  
 140 The policy network parameters  $\theta$  are then updated using the REINFORCE gradient (Williams, 1992):

$$141 \quad \nabla_{\theta} J(\theta) = \frac{1}{N} \sum_{i=1}^N (R_i - b) \nabla_{\theta} \log \pi_{\theta}(a_i | s_i), \quad (2)$$

144 where  $i$  is the index of the trajectory and  $\pi_{\theta}(a_i | s_i)$  denotes the probability assigned to action  $a_i$   
 145 conditioned on state  $s_i$ .  $b$  is a shared baseline used to reduce gradient variance, computed as the  
 146 average reward over all trajectories.

### 148 3 METHODOLOGY

150 Existing works only update the context embeddings while keeping node embeddings fixed. As  
 151 described in Section 2, the current state should include both candidate node embeddings and context  
 152 embeddings. In our method, we treat context and node status as a pair, ensuring that both reflect the  
 153 status of the current decoding step. During environment updates, we update both simultaneously to  
 154 maintain alignment between context and node information.

#### 156 3.1 OVERVIEW OF CHAIN-OF-CONTEXT LEARNING (CCL)

158 Fig. 1 (a) illustrates the training workflow of our proposed *Chain-of-Context Learning* (CCL).  
 159 It adopts the classic encoder-decoder paradigm, with Relevance-Guided Context Reformulation  
 160 (RGCR) and Trajectory-Shared Node Re-embedding (TSNR) integrated into the decoding stage.  
 161 During encoding, each VRP instance-comprising constraints, depot, and node features-is embedded  
 162 using a transformer encoder. Instances from 16 tasks, derived from the four constraints (B, L, O,

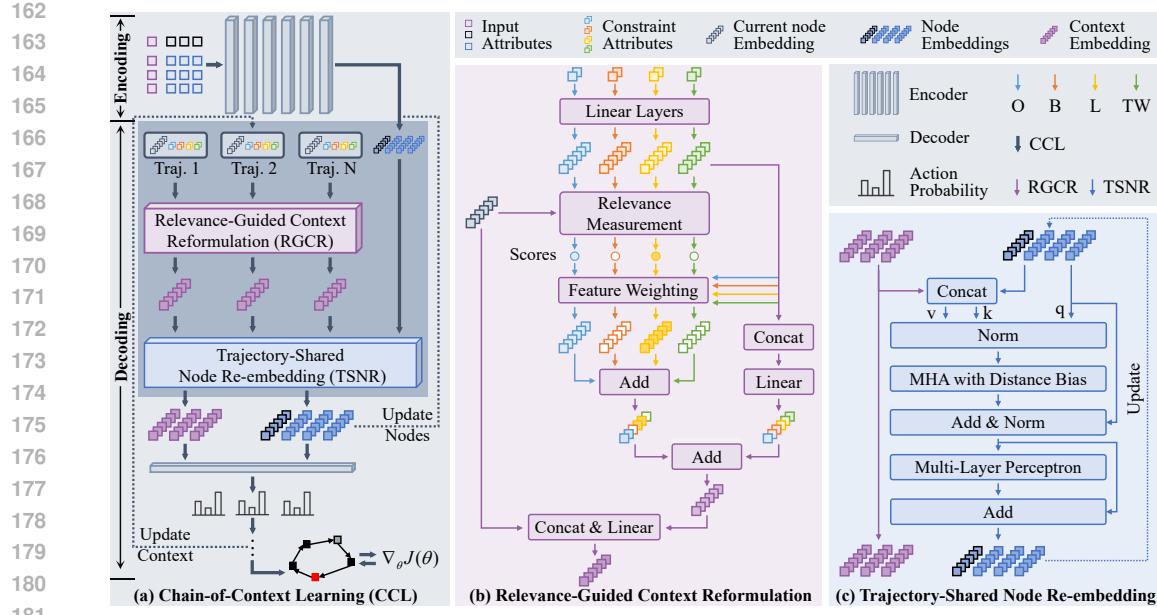


Figure 1: (a) CCL enables fine-grained constraint understanding by integrating RGCR and TSNR into the decoding stage. (b) and (c) illustrate the internal architectures of RGCR and TSNR, respectively.

and TW), are combined into a single batch for multi-task learning. In the RL-based decoding stage, CCL employs a lightweight architecture to make decisions, with multiple trajectories explored in parallel from diverse starting points. At each decision step, RGCR aggregates the constraint-specific attributes and current node embedding to generate a context embedding. After collecting the context embeddings from all trajectories, TSNR refines the historical node embeddings by jointly processing them with the multi-trajectory contexts. These refined node embeddings are passed to the next step, progressively influencing context construction and forming a Chain-of-Context across decoding steps. The constructed context and refined node features are used together to make the decision, with all components jointly optimized using an RL objective. The inference procedure is similar to the training setup, except it is extended to evaluate generalization on two additional constraints, *i.e.*, MB and MD, which are held out during training for zero-shot evaluation.

### 3.2 ENCODER

In the encoding stage, as shown in Fig. 1 (a), the inputs includes the constraint flag  $\tilde{\mathbf{h}}$  and the node attributes  $\mathbf{h} = \{\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_N\}$ . These attributes are embeded through a transformer-based encoder  $\mathcal{E}(\cdot)$ , resulting in node embeddings  $\mathbf{H} \in \mathbb{R}^{(N+1) \times D}$ :

$$\mathbf{H} = \mathcal{E}(\tilde{\mathbf{h}}, \mathbf{h}). \quad (3)$$

Following (Li et al., 2025), the constraint label  $\tilde{\mathbf{h}} \in \mathbb{R}^4$  is a one-hot vector to indicate the presence of 4 constraints (*i.e.*, B, O, L, TW). The depot attribute  $\mathbf{h}_0 = \{c_0^x, c_0^y, o, l\} \in \mathbb{R}^4$  includes the depot coordinates, and labels of O and L.  $\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_N\} \in \mathbb{R}^{N \times 7}$  are customer features, with each node  $\mathbf{h}_i = \{[c_i^x, c_i^y], [\delta_i^l, \delta_i^b], [t_i^e, t_i^l, t_i^s]\}$  specifying the coordinates, demands, and time windows. For simplicity, the encoder’s input processing and architecture are provided in Appendix B.1.

### 3.3 RELEVANCE-GUIDED CONTEXT REFORMULATION (RGCR)

In multi-constraint scenarios, RGCR automatically learns the relative importance of constraints at each step, enabling the model to focus on the most critical ones. In Fig. 1(b), RGCR undertakes three steps to formulate context embedding: (1) generating embedding for each constraint, (2) computing the correlation between each constraint embedding and the current node embedding, and (3) adaptively aggregating constraint embeddings based on correlation scores.

In the constraint embedding formulation, we first extract the corresponding attributes and then project them through separate linear layers. For the  $i$ -th trajectory at decoding step  $j$ , the current node index is denoted as  $\tau_{i,j}$ . The attributes for each constraint are summarized as follows:

$$\begin{aligned}\mathbf{c}_{i,j}^B &= \{\delta_{\tau_{i,j}}^l, \delta_{\tau_{i,j}}^b, c_{i,j}\}, & \mathbf{c}_{i,j}^L &= \{c_{\tau_{i,j}}^x, c_{\tau_{i,j}}^y, d_{i,j}\}, \\ \mathbf{c}_{i,j}^O &= \{c_{\tau_{i,j}}^x, c_{\tau_{i,j}}^y, d_{i,j}'\}, & \mathbf{c}_{i,j}^{TW} &= \{t_{\tau_{i,j}}^e, t_{\tau_{i,j}}^l, t_{\tau_{i,j}}^s, t_{i,j}\},\end{aligned}\quad (4)$$

where  $\delta^l, \delta^b$  denote the linehaul and backhaul demands, and  $c_{i,j}$  is the remaining vehicle capacity. The coordinates  $c^x, c^y$  specify node locations in the two-dimensional space,  $d$  is the remaining distance of the current sub-route, and  $d'$  is the total distance traveled. Moreover,  $t^e, t^l, t^s, t$  represent the earliest, latest, service times, and current time, respectively. These attributes are separately fed to linear layers for producing constraint embeddings, denoted as:

$$\mathbf{C}_{i,j}^k = \mathcal{H}(\mathbf{c}_{i,j}^k), \quad (5)$$

where  $\mathbf{C}_{i,j}^k \in \mathbb{R}^D$ ,  $k \in \{B, L, O, TW\}$  is the constraint type, and  $\mathcal{H}(\cdot)$  denotes the linear layer used for projection. In correlation computing, these constraint embeddings interact with the current node embedding to produce correlation scores, denoted as:

$$s_{i,j}^k = \mathbf{H}_{\tau_{i,j}} \cdot \mathbf{C}_{i,j}^k, \quad (6)$$

where  $\mathbf{H}_{\tau_{i,j}} \in \mathbb{R}^D$  is the current node embedding, and  $\cdot$  denotes the dot product used for calculating the correlation scores (or similarities). In constraint aggregating, the unified constraint embedding is obtained by adding the original and enhanced constraint embedding, denoted as  $\mathbf{S}_{i,j} = \tilde{\mathbf{S}}_{i,j} + \bar{\mathbf{S}}_{i,j}$ . The original part is defined as the concatenation of the four constraint embeddings from Eq. (5):

$$\tilde{\mathbf{S}}_{i,j} = \mathcal{H}(\text{Concat}(\mathbf{C}_{i,j}^B, \mathbf{C}_{i,j}^L, \mathbf{C}_{i,j}^O, \mathbf{C}_{i,j}^{TW})), \quad (7)$$

where  $\text{Concat}(\cdot)$  denotes concatenation along the feature dimension, resulting in a concatenated embedding of size  $N \times 4D$ .  $\mathcal{H}(\cdot)$  is a linear layer that projects the  $4D$  input back to  $D$ , resulting in  $\tilde{\mathbf{S}}_{i,j} \in \mathbb{R}^D$ . For the enhanced part, we apply a weighted sum over the constraint embeddings:

$$\bar{\mathbf{S}}_{i,j} = s_{i,j}^B \mathbf{C}_{i,j}^B + s_{i,j}^L \mathbf{C}_{i,j}^L + s_{i,j}^O \mathbf{C}_{i,j}^O + s_{i,j}^{TW} \mathbf{C}_{i,j}^{TW}. \quad (8)$$

The final context embedding is aggregated from the unified constraint and current node embeddings:

$$\tilde{\mathbf{C}}_{i,j} = \mathcal{H}(\text{Concat}(\mathbf{S}_{i,j}, \mathbf{H}_{\tau_{i,j}})). \quad (9)$$

### 3.4 TRAJECTORY-SHARED NODE RE-EMBEDDING (TSNR)

To capture node-specific states influenced by the current context, we aggregate contextual semantics from other nodes and multi-trajectory contexts into the node embeddings. As illustrated in Fig. 1 (c), this is achieved via a multi-head attention mechanism, where node embeddings serve as queries, and the unified node-context information acts as keys and values. Formally, at step  $j$ , we denote the context embedding for  $N$  trajectories as  $\tilde{\mathbf{C}}_j = \text{Concat}(\tilde{\mathbf{C}}_{1,j}, \tilde{\mathbf{C}}_{2,j}, \dots, \tilde{\mathbf{C}}_{N,j})$ , where  $\tilde{\mathbf{C}}_j \in \mathbb{R}^{N \times D}$ . By using the last step node  $\mathbf{H}_{j-1} \in \mathbb{R}^{(N+1) \times D}$ , the query, keys, and values are represented as

$$\mathbf{q}_j = \mathcal{H}(\text{Norm}(\mathbf{H}_{j-1})), \quad \mathbf{k}_j, \mathbf{v}_j = \mathcal{H}(\text{Norm}(\text{Concat}(\mathbf{H}_{j-1}, \tilde{\mathbf{C}}_j))), \quad (10)$$

where  $\mathbf{q}_j \in \mathbb{R}^{N \times D}$  and  $\mathbf{k}_j, \mathbf{v}_j \in \mathbb{R}^{(N+1) \times D}$ .  $\text{Norm}(\cdot)$  denotes the Root Mean Square (RMS) normalization layer (Zhang and Sennrich, 2019). For simplicity, we use the same notation  $\mathcal{H}(\cdot)$  to denote the module that produces  $\mathbf{k}_j$  and  $\mathbf{v}_j$ . To calculate attention weights, we further incorporate a distance-based bias to prevent the model from overfitting to TW. This bias term, denoted as  $\mathbf{B}_j = \text{Concat}(\mathbf{d}^{n-n}, \mathbf{d}_j^{c-n})$ , consists of two parts: the node-node and node-context distance:

$$\begin{aligned}\mathbf{d}^{n-n} &= \{d_{m,n} | m, n \in \{0, 1, \dots, N\}\}, \\ \mathbf{d}_j^{c-n} &= \{d_{m,n} | m \in \{\tau_{1,j}, \tau_{2,j}, \dots, \tau_{N,j}\}, n \in \{0, 1, \dots, N\}\},\end{aligned}\quad (11)$$

where  $\mathbf{d}^{n-n} \in \mathbb{R}^{(N+1) \times (N+1)}$ ,  $\mathbf{d}_j^{c-n} \in \mathbb{R}^{N \times (N+1)}$  and each element of it takes the form  $d_{m,n} = \|\mathbf{c}_m - \mathbf{c}_n\|_2$  with  $\mathbf{c} = \{c^x, c^y\}$  denoting Euclidean coordinates. For the node-context part, we

270 extract the coordinates of the current node for each trajectory (indexed by  $\{\tau_{1,j}, \tau_{2,j}, \dots, \tau_{N,j}\}$ ) and  
 271 compute their distances to all candidate nodes. The attention weights are subsequently computed as  
 272

$$273 \quad \mathbf{A}_j = \text{Softmax}(\mathbf{q}_j \mathbf{k}_j^\top / \sqrt{D} + \mathbf{B}_j), \quad (12)$$

274 where  $\mathbf{A}_j, \mathbf{B}_j \in \mathbb{R}^{(N+1) \times (N+1+N)}$ , and  $\text{Softmax}(\cdot)$  is the softmax operation. The re-embedded  
 275 node representations are computed as follows:

$$276 \quad \tilde{\mathbf{H}}_j = \mathbf{q}_j + \mathbf{A}_j \mathbf{v}_j, \quad \mathbf{H}_j = \tilde{\mathbf{H}}_j + \text{MLP}(\text{Norm}(\tilde{\mathbf{H}}_j)). \quad (13)$$

277 We preserve the updated node embeddings  $\mathbf{H}_j$  from the current step and use them as input queries  
 278 for the next step, with update frequency controlled by probabilities  $P_{tr}$  (training) and  $P_{ts}$  (testing).  
 279

### 280 3.5 STEP-WISE DECISION AND TRAINING OBJECTIVE

282 Once the context embedding  $\tilde{\mathbf{C}}_j \in \mathbb{R}^{N \times D}$  and current node embeddings  $\mathbf{H}_j \in \mathbb{R}^{(N+1) \times D}$  are  
 283 obtained, we use them to predict the selection of the next node, and then compute the RL objective  
 284 function to optimize model parameters. In the step-wise decision stage, we employ a classic decoder  
 285 (shown in Appendix B.2) to acquire the probability of selecting the next node. This procedure is  
 286 represented as follows:

$$287 \quad \mathbf{P}_j = \mathcal{D}(\tilde{\mathbf{C}}_j, \mathbf{H}_j, \mathbf{M}_j), \quad (14)$$

288 where  $\mathbf{P}_j \in \mathbb{R}^{N \times (N+1)}$ ,  $\mathcal{D}(\cdot)$  denotes the decoder, while  $\mathbf{M}_j$  is a mask that prevents revisiting  
 289 previously selected nodes. If all constraints are satisfied, the node with the highest probability  
 290 is selected as the next node to visit. Otherwise, the depot is selected. After one interaction, the  
 291 model generates  $N$  solution trajectories, each denoted as  $\tau_i = \{\tau_{i,1}, \tau_{i,2}, \dots, \tau_{i,N'}\}$ , where  $i \in$   
 292  $\{1, 2, \dots, N\}$  and  $N'$  is the total number of decision steps. **The RL objective is then computed using  
 293 the reward of each trajectory and the log-probabilities of selected nodes**, as illustrated in Eq. 2.

## 294 4 EXPERIMENTS

### 297 4.1 DATASETS AND EVALUATION METRICS

299 We evaluate CCL on 48 VRP variants. Following (Berto et al., 2024b), node locations are sampled  
 300 uniformly from the 2D Euclidean space  $[0, 1]^2$ . Each vehicle starts at the depot with a capacity  
 301  $Q=1$  and a maximum route duration  $l=3$ . Linehaul and backhaul demands are sampled as integers  
 302 from  $[1, 10]$  and scaled by a factor of  $30 + N/5$ , where  $N$  is the number of customers. In backhaul  
 303 settings, 20% of the customers are designated as backhaul, and the remaining 80% as linehaul. For  
 304 time window tasks, early arrival times, service durations, and time window lengths are independently  
 305 sampled from  $[0.0126, 4.25]$ ,  $[0, 0.15]$ , and  $[1.8, 2.0]$ , respectively. Late times are computed as the  
 306 sum of early times and window lengths. The training set consists of 100,000 instances uniformly  
 307 distributed across 16 variants. The best model checkpoint is selected based on validation performance  
 308 on CVRP (Capacitated VRP), using a held-out set of 128 instances. The test set comprises 48 variants,  
 309 each containing 1,000 instances. We benchmark CCL against state-of-the-art (SOTA) baselines under  
 310 two settings:  $N=50$  and  $N=100$ . We evaluate performance using three standard VRP metrics: total  
 311 routing length ("Obj."), performance gap ("Gap") to the strong baseline HGS-PyVRP (Wouda et al.,  
 312 2024), and inference time. All metrics are computed over 1,000 test instances, with "Obj." and "Gap"  
 313 reported as averages and inference time as total runtime.

### 314 4.2 IMPLEMENTATION DETAILS

315 Our method is implemented in PyTorch (Paszke et al., 2019). All experiments are conducted on  
 316 a machine with an AMD EPYC 7702P 24-core CPU and a single NVIDIA RTX L40S GPU. We  
 317 use a batch size of 256 during training. The model adopts a 6-layer Transformer encoder, with both  
 318 encoder and decoder sharing the same architecture: embedding dimension  $D=128$ , 8 attention heads,  
 319 and a hidden dimension of 512. During decoding, node refinement is applied probabilistically. For  
 320 instances with  $N=50$ , the refinement probability is 0.75 during training and 1.0 during testing. For  
 321  $N=100$ , the respective probabilities are 0.25 and 0.5. The model is optimized using Adam with a  
 322 learning rate of  $3 \times 10^{-4}$  and a weight decay of  $1 \times 10^{-6}$ . A multi-step learning rate scheduler is  
 323 used with milestones at epochs 270 and 295, a decay factor of 0.1, and gradient clipping set to 1.  
 Training is conducted for a total of 300 epochs. **We will make our code publicly available.**

324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
Table 1: Performance on 16 seen in-distribution tasks. \* denotes the strong baseline used to compute the gap. Best neural approach is highlighted in **bold**; best existing SOTA is underlined.

Methods	N = 50			N = 100			Methods	N = 50			N = 100			
	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓		Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	
CVRP	HGS-PyVRP	10.372	*	10.4m	15.628	*	20.8m	HGS-PyVRP	16.031	*	10.4m	25.423	*	20.8m
	MTPOMO	10.520	1.423%	2s	15.941	2.030%	8s	MTPOMO	16.419	2.423%	2s	26.433	3.962%	9s
	MVMoE	10.499	1.229%	3s	15.888	1.693%	11s	MVMoE	16.400	2.298%	3s	26.390	3.789%	11s
	RF-TE	10.502	1.257%	2s	15.860	1.524%	8s	RF-TE	16.341	1.933%	2s	26.228	3.154%	8s
	CaDA	10.505	1.287%	2s	15.843	1.412%	8s	CaDA	16.312	1.745%	1s	26.169	2.925%	9s
	CaDA <sup>†</sup>	10.471	0.959%	3s	<u>15.790</u>	<u>1.070%</u>	13s	CaDA <sup>†</sup>	16.299	1.670%	3s	<u>26.105</u>	2.668%	14s
OVRP	CCL	10.473	0.977%	5s	15.823	1.287%	19s	CCL	16.190	0.979%	5s	25.913	1.908%	21s
	CCL <sup>†</sup>	<b>10.463</b>	<b>0.881%</b>	6s	<b>15.787</b>	<b>1.058%</b>	24s	CCL <sup>†</sup>	<b>16.177</b>	<b>0.907%</b>	7s	<b>25.862</b>	<b>1.706%</b>	24s
	HGS-PyVRP	6.507	*	10.4m	9.725	*	20.8m	HGS-PyVRP	10.510	*	10.4m	16.926	*	20.8m
	MTPOMO	6.717	3.194%	2s	10.216	5.028%	8s	MTPOMO	10.676	1.558%	2s	17.442	3.022%	9s
	MVMoE	6.705	3.003%	3s	10.177	4.617%	11s	MVMoE	10.674	1.541%	3s	17.416	2.870%	12s
	RF-TE	6.682	2.658%	2s	10.115	3.996%	8s	RF-TE	10.645	1.264%	2s	17.328	2.352%	9s
OVRPB	CaDA	6.677	2.585%	1s	10.095	3.786%	8s	CaDA	10.630	1.122%	1s	17.283	2.086%	9s
	CaDA <sup>†</sup>	<u>6.652</u>	<u>2.212%</u>	3s	<u>10.060</u>	<u>3.425%</u>	13s	CaDA <sup>†</sup>	<u>10.621</u>	<u>1.030%</u>	3s	<u>17.246</u>	<u>1.868%</u>	14s
	CCL	6.636	1.957%	5s	10.068	3.511%	20s	CCL	10.569	0.543%	6s	17.123	1.142%	21s
	CCL <sup>†</sup>	<b>6.610</b>	<b>1.566%</b>	6s	<b>10.012</b>	<b>2.936%</b>	25s	CCL <sup>†</sup>	<b>10.564</b>	<b>0.506%</b>	7s	<b>17.104</b>	<b>1.033%</b>	26s
	HGS-PyVRP	6.898	*	10.4m	10.335	*	20.8m	HGS-PyVRP	11.669	*	10.4m	19.156	*	20.8m
	MTPOMO	7.105	2.973%	2s	10.882	5.264%	8s	MTPOMO	11.823	1.307%	3s	19.656	2.592%	9s
OVRPB	MVMoE	7.089	2.744%	3s	10.841	4.869%	11s	MVMoE	11.816	1.245%	4s	19.637	2.499%	13s
	RF-TE	7.065	2.385%	2s	10.774	4.233%	8s	RF-TE	11.790	1.027%	2s	19.555	2.062%	9s
	CaDA	7.064	2.377%	1s	10.739	3.890%	8s	CaDA	11.775	0.898%	2s	19.495	1.754%	9s
	CaDA <sup>†</sup>	<u>7.032</u>	<u>1.916%</u>	3s	<u>10.682</u>	<u>3.329%</u>	13s	CaDA <sup>†</sup>	<u>11.768</u>	<u>0.843%</u>	3s	<u>19.469</u>	<u>1.617%</u>	15s
	CCL	7.008	1.568%	5s	10.666	3.179%	19s	CCL	11.721	0.436%	6s	19.348	0.985%	21s
	CCL <sup>†</sup>	<b>6.992</b>	<b>1.344%</b>	6s	<b>10.624</b>	<b>2.775%</b>	25s	CCL <sup>†</sup>	<b>11.718</b>	<b>0.416%</b>	7s	<b>19.329</b>	<b>0.888%</b>	27s
OVRPBL	HGS-PyVRP	6.899	*	10.4m	10.335	*	20.8m	HGS-PyVRP	11.668	*	10.4m	19.156	*	20.8m
	MTPOMO	7.112	3.053%	2s	10.888	5.318%	8s	MTPOMO	11.823	1.307%	3s	19.656	2.592%	9s
	MVMoE	7.094	2.799%	3s	10.847	4.929%	11s	MVMoE	11.816	1.249%	4s	19.640	2.514%	12s
	RF-TE	7.068	2.417%	2s	10.778	4.266%	8s	RF-TE	11.789	1.017%	2s	19.554	2.061%	9s
	CaDA	7.062	2.339%	1s	10.741	3.900%	8s	CaDA	11.777	0.914%	2s	19.497	1.762%	9s
	CaDA <sup>†</sup>	<u>7.034</u>	<u>1.935%</u>	3s	<u>10.686</u>	<u>3.368%</u>	13s	CaDA <sup>†</sup>	<u>11.769</u>	<u>0.848%</u>	3s	<u>19.467</u>	<u>1.602%</u>	15s
OVRPL	CCL	7.009	1.569%	5s	10.681	3.323%	20s	CCL	11.721	0.442%	6s	19.346	0.977%	22s
	CCL <sup>†</sup>	<b>6.992</b>	<b>1.335%</b>	6s	<b>10.609</b>	<b>2.631%</b>	23s	CCL <sup>†</sup>	<b>11.718</b>	<b>0.414%</b>	7s	<b>19.334</b>	<b>0.915%</b>	27s
	HGS-PyVRP	6.507	*	10.4m	9.724	*	20.8m	HGS-PyVRP	10.510	*	10.4m	16.926	*	20.8m
	MTPOMO	6.720	3.248%	2s	10.224	5.112%	8s	MTPOMO	10.677	1.572%	2s	17.442	3.020%	9s
	MVMoE	6.706	3.028%	3s	10.184	4.693%	11s	MVMoE	10.677	1.564%	3s	17.418	2.880%	12s
	RF-TE	6.683	2.680%	2s	10.121	4.054%	8s	RF-TE	10.646	1.267%	2s	17.328	2.352%	9s
OVRPL	CaDA	6.680	2.623%	1s	10.093	3.773%	8s	CaDA	10.631	1.133%	1s	17.280	2.073%	9s
	CaDA <sup>†</sup>	<u>6.652</u>	<u>2.200%</u>	2s	<u>10.060</u>	<u>3.432%</u>	13s	CaDA <sup>†</sup>	<u>10.621</u>	<u>1.033%</u>	3s	<u>17.244</u>	<u>1.861%</u>	14s
	CCL	6.637	1.968%	5s	10.067	3.495%	20s	CCL	10.569	0.546%	6s	17.123	1.143%	22s
	CCL <sup>†</sup>	<b>6.610</b>	<b>1.569%</b>	6s	<b>10.000</b>	<b>2.811%</b>	24s	CCL <sup>†</sup>	<b>10.564</b>	<b>0.501%</b>	7s	<b>17.109</b>	<b>1.063%</b>	26s
	HGS-PyVRP	9.687	*	10.4m	14.377	*	20.8m	HGS-PyVRP	18.292	*	10.4m	29.467	*	20.8m
	MTPOMO	10.036	3.596%	2s	15.102	5.052%	8s	MTPOMO	18.649	1.938%	2s	30.478	3.426%	9s
VRPB	MVMoE	10.007	3.292%	3s	15.023	4.505%	10s	MVMoE	18.632	1.841%	3s	30.437	3.284%	12s
	RF-TE	9.979	3.000%	2s	14.935	3.906%	8s	RF-TE	18.573	1.517%	2s	30.249	2.641%	9s
	CaDA	9.979	3.010%	1s	14.910	3.721%	8s	CaDA	18.543	1.361%	1s	30.174	2.390%	9s
	CaDA <sup>†</sup>	9.922	2.405%	2s	14.838	3.222%	13s	CaDA <sup>†</sup>	18.528	1.276%	3s	30.113	2.183%	14s
	CCL	9.916	2.352%	5s	14.882	3.526%	19s	CCL	18.430	0.738%	6s	29.911	1.494%	21s
	CCL <sup>†</sup>	<b>9.875</b>	<b>1.921%</b>	6s	<b>14.780</b>	<b>2.808%</b>	22s	CCL <sup>†</sup>	<b>18.419</b>	<b>0.678%</b>	7s	<b>29.871</b>	<b>1.357%</b>	26s
VRPL	HGS-PyVRP	10.186	*	10.4m	14.779	*	20.8m	HGS-PyVRP	18.361	*	10.4m	29.026	*	20.8m
	MTPOMO	10.679	4.760%	2s	15.718	6.294%	8s	MTPOMO	19.001	2.199%	3s	30.948	3.794%	9s
	MVMoE	10.639	4.384%	3s	15.642	5.771%	11s	MVMoE	18.983	2.097%	3s	30.892	3.609%	12s
	RF-TE	10.569	3.713%	2s	15.523	5.008%	8s	RF-TE	18.910	1.713%	2s	30.705	2.978%	9s
	CaDA	10.576	3.776%	1s	15.490	4.771%	8s	CaDA	18.894	1.623%	1s	30.620	2.700%	9s
	CaDA <sup>†</sup>	<u>10.503</u>	<u>3.064%</u>	3s	<u>15.389</u>	<u>4.093%</u>	13s	CaDA <sup>†</sup>	<u>18.878</u>	<u>1.540%</u>	3s	<u>30.570</u>	<u>2.531%</u>	15s
VRPL	CCL	10.484	2.883%	5s	15.407	4.219%	19s	CCL	18.773	0.976%	6s	30.366	1.842%	21s
	CCL <sup>†</sup>	<b>10.440</b>	<b>2.450%</b>	6s	<b>15.297</b>	<b>3.472%</b>	24s	CCL <sup>†</sup>	<b>18.758</b>	<b>0.899%</b>	7s	<b>30.323</b>	<b>1.697%</b>	25s
	HGS-PyVRP	10.587	*	10.4m	15.766	*	20.8m	HGS-PyVRP	16.356	*	10.4m	25.757	*	20.8m
	MTPOMO	10.775	1.733%	2s	16.157	2.483%	8s	MTPOMO	16.832	2.877%	2s	26.913	4.455%	9s
	MVMoE	10.753	1.525%	3s	16.099	2.113%	11s	MVMoE	16.817	2.783%	3s	26.866	4.272%	12s
	RF-TE	10.747	1.485%	2s	16.057	1.858%	8s	RF-TE	16.728	2.248%	2s	26.706	3.645%	9s
VRPLTW	CaDA	10.749	1.505%	1s	16.036	1.725%	8s	CaDA	16.709	2.130%	1s	26.631	3.358%	9s
	CaDA <sup>†</sup>	<u>10.707</u>	<u>1.112%</u>	2s	<u>15.984</u>	<u>1.400%</u>	13s	CaDA <sup>†</sup>	<u>16.692</u>	<u>2.034%</u>	3s	<u>26.556</u>	<u>3.065%</u>	14s
	CCL	10.710	1.145%	5s	16.009	1.561%	19s	CCL	16.579	1.333%	6s	26.366	2.321%	20s
	CCL <sup>†</sup>	<b>10.698</b>	<b>1.027%</b>	6s	<b>15.960</b>	<b>1.245%</b>	23s	CCL <sup>†</sup>	<b>16.555</b>	<b>1.192%</b>	7s	<b>26.324</b>	<b>2.157%</b>	24s

### 4.3 COMPARISON WITH THE STATE-OF-THE-ARTS

**Baselines.** We compare CCL with state-of-the-art multi-task VRP solvers, including MTPOMO (Liu et al., 2024), MVMoE (Zhou et al., 2024a), RouteFinder (RF-TE) (Berto et al., 2024b), and CaDA (Li et al., 2025). Among these, RF-TE and CaDA have reported the strongest performance, and we include them in both in-distribution (Table 1) and out-of-distribution (Table 2) evaluations. To ensure a fair comparison, we reimplement CaDA in the RouteFinder framework (Berto et al., 2024b), which also serves as the basis for RF-TE and our CCL. As shown in Appendix C.1.1, our reproduction closely matches the performance reported in the original paper (Li et al., 2025). To further enhance performance, we integrate a context-aware module, ReLD (Huang et al., 2025), into both CaDA and CCL, denoted as CaDA<sup>†</sup> and CCL<sup>†</sup>, respectively.

378  
379  
380 Table 2: Generalization on 32 unseen out-of-distribution tasks.  
381  
382  
383  
384  
385  
386  
387

Methods	MDOVRPB, MDOVRPL, MDVRPBL, MDOVRPBL MDCVRP, MDOVRP, MDVRPB, MDVRPL VRPMB, OVRPMB, VRPML, OVRPMLB MDVRPMB, MDOVRPMB, MDVRPML, MDOVRPMLB			MDOVRPBTW, MDOVRPLTW, MDVRPBLTW, MDOVRPBLTW MDCVRPTW, MDOVRPTW, MDVRPBTW, MDVRPLTW VRPMBTW, OVRPMBTW, VRPMLTW, OVRPMLBTW MDVRPMBTW, MDOVRPMBTW, MDVRPMLTW, MDOVRPMLBTW			N = 50			N = 100		
	N = 50			N = 100			N = 50			N = 100		
	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓
RF-TE	9.651	40.472%	1s	14.746	45.724%	9s	14.557	32.586%	2s	24.217	36.564%	10s
CaDA	9.535	38.860%	2s	14.910	47.156%	10s	14.410	30.872%	2s	23.523	32.512%	10s
CaDA <sup>†</sup>	9.285	34.169%	3s	14.395	41.419%	16s	14.830	34.665%	4s	24.328	36.774%	17s
CCL	8.906	29.156%	4s	14.071	38.624%	17s	13.923	26.020%	4s	23.375	31.159%	18s
CCL <sup>†</sup>	<b>8.673</b>	<b>25.781%</b>	5s	<b>13.777</b>	<b>35.413%</b>	22s	<b>13.536</b>	<b>22.422%</b>	5s	24.025	34.163%	25s

388  
389 Table 3: Ablation on key modules within CCL.  
390  
391  
392  
393  
394  
395  
396  
397  
398

Methods	CVRP	OVRP	VRPB	VRPL	OVRPB	OVRPL	VRPBL	OVRPBL	Avg.
CCL <sup>†</sup>	0.881%	<b>1.566%</b>	<b>1.921%</b>	1.027%	<b>1.344%</b>	<b>1.569%</b>	<b>2.450%</b>	<b>1.335%</b>	<b>1.512%</b>
- RGCR	<b>0.874%</b>	1.710%	1.969%	<b>0.993%</b>	1.396%	1.712%	2.486%	1.407%	1.568%
- TSNR	0.961%	2.284%	2.413%	1.131%	1.969%	2.311%	3.001%	1.973%	2.005%
- RGCR - TSNR	1.014%	2.395%	2.416%	1.160%	2.036%	2.411%	3.088%	2.041%	2.070%
Methods	VRPTW	OVRPTW	VRPBTW	VRPLTW	OVRPBTW	OVRPLTW	VRPBLTW	OVRPBLTW	Avg.
CCL <sup>†</sup>	<b>0.907%</b>	<b>0.506%</b>	<b>0.678%</b>	<b>1.192%</b>	<b>0.416%</b>	<b>0.501%</b>	<b>0.899%</b>	<b>0.414%</b>	<b>0.689%</b>
- RGCR	0.938%	0.521%	0.720%	1.235%	0.419%	0.519%	0.926%	0.427%	0.713%
- TSNR	1.539%	0.947%	1.204%	1.857%	0.795%	0.957%	1.409%	0.805%	1.189%
- RGCR - TSNR	1.615%	0.969%	1.266%	1.930%	0.813%	0.958%	1.492%	0.810%	1.232%

400  
401 **In-Distribution Evaluation.** In the Table 1, CCL outperforms CaDA across both  $N=50$  and  $N=100$   
402 settings. Specifically, for  $N=50$ , both CCL and CCL<sup>†</sup> achieve lower performance gaps than CaDA on  
403 all 16 evaluated tasks. For  $N=100$ , the gap relative to the HGS-PyVRP baseline narrows even further.  
404 We also observe complementary strengths between ReLD and CCL. ReLD performs particularly well  
405 on variants without time windows (TW), leveraging its ability to extract globally shared constraint  
406 signals through step-wise context. In contrast, CCL’s dynamic node refinement excels on TW tasks,  
407 offering finer-grained adaptation to local, node-specific constraints. By combining both, the resulting  
408 CCL<sup>†</sup> achieves the best overall performance across all the in-distribution tasks.

409 **Out-of-Distribution Evaluation.** We present the averaged performance for both tasks with or  
410 without TW in Table 2 (detailed results for each task are presented in Appendix C.1.3, where CCL  
411 outperformed CaDA on the majority). Table 2 shows that CCL<sup>†</sup> consistently outperforms other  
412 methods under the  $N=50$  setting. For  $N=100$ , while CCL<sup>†</sup> maintains competitive performance, it  
413 shows a slightly higher gap on TW tasks compared to standalone CCL. We hypothesize that this may  
414 be due to a low test-time update rate, which can cause the model to overfit to static constraint structures  
415 and under-adapt to time-sensitive variations, thus increasing the gap. Nevertheless, either equipped  
416 with ReLD or not, our CCL exhibits superior overall performance to CaDA (and its counterpart).

#### 417 4.4 ABLATION STUDIES

418 **Ablation on Key Modules within CCL.** We conduct ablation experiments to validate the effectiveness  
419 of RGCR and TSNR in CCL<sup>†</sup>. Table 3 reports the results for  $N=50$ . Removing RGCR leads to  
420 a smaller gap increase than TSNR, and even slightly reduces the gap on CVRP and VRPL. It is likely  
421 that the relatively simple constraints of the two VRP tasks make relevance weighting less effective.  
422 Moreover, removing both modules yields the highest gap, highlighting their complementary effectiveness.  
423 We also conduct these ablations on CCL (the variant without ReLD), showing that the main  
424 performance gains come from CCL itself rather than ReLD. Details are presented in Appendix C.3.1.  
425

426 **Ablation on Key Components within RGCR and TSNR.** We further conduct ablation studies using  
427  $N=50$  to evaluate the components within RGCR and TSNR.  
428

429 Regarding RGCR, we first examine direct concatenation of attributes (as in CaDA) and embeddings  
430 (CCL<sup>†</sup>-RGCR in Table 3). We then evaluate three correlation scores, namely random, cosine  
431 similarity, and dot product, as defined in Eq. 6. The left part of Fig. 2 presents the averaged gap  
432 and the corresponding model complexity. Compared with direct concatenation, all three correlation

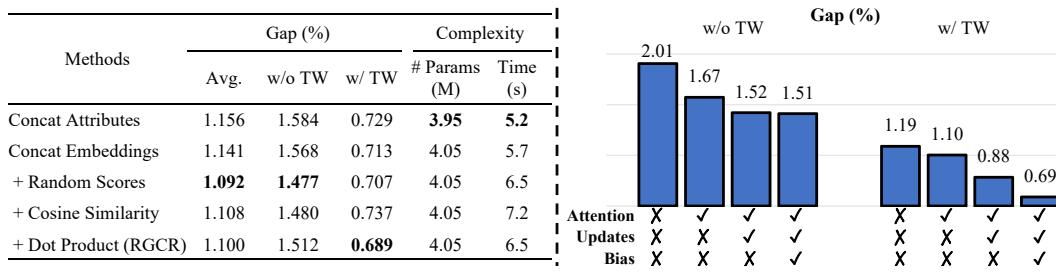


Figure 2: Ablation on key components within RGCR (Left) and TSNR (Right), respectively.

scores reduce the gap in both settings. Notably, the dot product achieves the smallest gap on tasks with TW, demonstrating its superiority in handling complex constraints. More experimental setups and analyses are provided in Appendix C.3.2.

Regarding TSNR, the right part of Fig. 2 shows that combining node-level attention, embedding updates, and the distance bias in Eq. 12 achieves the lowest gaps, indicating that all three elements are essential for improving the overall performance. Moreover, a detailed analysis that node-level attention reduces model complexity compared to the vanilla Transformer is provided in Appendix C.3.3.

## 5 DISCUSSION

We discuss the strengths and limitations of CCL. Its main drawback is the longer inference time required for better performance. However, flexible parameter settings can mitigate this issue, enabling CCL to perform well on large-scale real-world instances.

**Complexity Analysis.** Table 4 compares the model complexity of SOTA methods and our CCL. We further compare CCL with a heavy-decoder variant of the SOTA model, denoted as CaDA<sup>†</sup>-HD. Detailed configurations of this variant are presented in Appendix C.4. All models are trained and evaluated on the 16 VRP variants with  $N=50$  using an L40S GPU. "Time" denotes the total inference time over 1,000 test instances, "# Params" refers to the total number of parameters in the encoder and decoder, and "Memory" indicates the peak memory usage during testing across all 16 variants. Compared to CaDA and CaDA<sup>†</sup>, our CCL and CCL<sup>†</sup> introduce only a moderate increase in memory usage and parameter count, while achieving a substantial performance improvement. The inference time is longer due to additional computation, but the gain in solution quality justifies the cost. In addition, we also apply the step-wise refinement strategy, *i.e.*, ASW-TAM (Xin et al., 2020) to the multi-task setting, where each route is re-embedded individually. However, due to memory constraints, we adopt a much smaller batch size that is only 1/16 of the original one. Results show that the naive refinement strategy leads to significantly higher gaps, longer inference time, and larger memory consumption, which further validates the effectiveness of CCL. Moreover, we observe that CCL achieves a comparable model cost while reducing the gap by 0.25% compared with CaDA<sup>†</sup>-HD. These findings indicate that the effectiveness of CCL stems from its design rather than from an increased network scale.

**Performance-Cost Trade-off.** In Section 3.4,  $P_{tr}$  and  $P_{ts}$  denote the probabilities of updating node embeddings during training and testing, and we assess their impact on model performance and inference efficiency, which also leads to a lightweight version of CCL. We first conduct sensitivity studies using  $N=50$  and evaluating all 20 combinations of  $P_{tr} \in \{0.25, 0.5, 0.75, 1\}$  and  $P_{ts} \in \{0, 0.25, 0.5, 0.75, 1\}$ . Fig. 3 shows

Table 4: Complexity analysis.

Methods	Gap↓ (%)	Memory↓ (GiB)	# Params↓ (M)	Time↓ (s)
CaDA	1.90	6.01	3.37 + 0.1	1.5
CaDA <sup>†</sup>	1.63	7.15	3.37 + 0.3	2.7
<b>CaDA<sup>†</sup>-HD</b>	<b>1.53</b>	<b>7.55</b>	<b>3.37+1.0</b>	<b>5.1</b>
CCL	1.28	8.13	3.39 + 0.45	5.4
CCL <sup>†</sup>	1.10	8.76	3.39 + 0.66	6.5
ASW-TAM	29.93	15.47	3.39 + 0.66	96.1

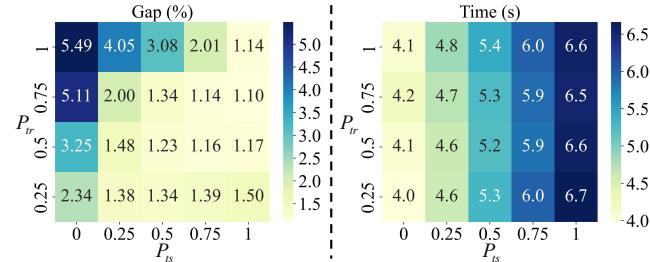


Figure 3: Gap (Left) and inference time (Right) under different training and testing update rates.

486 the corresponding gap values and inference time. We observe that, for a fixed probability  $P_{tr}$  during  
 487 training, the gap tends to be smaller when the probability  $P_{ts}$  during testing is slightly higher. For  
 488 example, when  $P_{tr}=0.25$  or  $0.5$ , the best performance is achieved at  $P_{ts}=0.5$  and  $0.75$ , respectively.  
 489 This work adopts  $P_{tr} = 0.75$  and  $P_{ts} = 1$ , as this setting yields the lowest gap. Meanwhile, reducing  
 490  $P_{ts}$  leads to shorter inference time, as fewer refinement steps are involved. While this leads to a  
 491 higher gap, it provides a trade-off between solution quality and inference efficiency. Motivated by  
 492 this, we design a lightweight version of CCL $^\dagger$  using  $P_{tr} = 0.25$  and  $P_{ts} = 0.25$ . It achieves an  
 493 average gap of  $1.38\%$  across 16 VRPs with an average inference time of  $4.6\text{s}$ , while the existing  
 494 SOTA CaDA $^\dagger$  attains a gap of  $1.63\%$  in  $2.7\text{s}$  (see Table 4). This enables users to adjust  $P_{ts}$  based on  
 495 the requirements of practical deployment scenarios, and more detailed comparisons between CCL  
 496 and CaDA are provided in Appendix C.4.  
 497

498 **Large-Scale Real-World Practicality.** We evaluate zero-shot  
 499 generalization on 60 real-world VRPTW instances, each with  
 500  $N=600$  (Homberger and Gehring, 1999). The model is trained  
 501 on 16 tasks with  $N=100$ . During inference, we apply an update  
 502 probability of  $P_{ts}=0.1$  to reduce computational cost. Ta-  
 503 ble 5 reports the averaged results across these instances (per-  
 504 instance results in Appendix C.7), showing that CCL achieves  
 505 the lowest average gap while maintaining comparable infer-  
 506 ence time. Appendix C.7 also reports results on VRPTW instances with  $N=100$  (Solomon, 1987),  
 507 where CCL outperforms SOTAs on 24 out of 27. It is further evaluated on CVRP instances with  
 508  $N \in [100, 251]$  (Uchoa et al., 2017), where CCL achieves the best performance on 16 out of 27. These  
 509 findings indicate that our method is well-suited for deployment in real-world scenarios, particularly  
 510 for problems with complex constraints such as time windows.  
 511

## 510 6 CONCLUSIONS

511 Existing neural multi-task VRP methods often neglect the evolving nature of node states during  
 512 decoding, limiting their ability to respond accurately to constraint requirements. To overcome this, we  
 513 proposed Chain-of-Context Learning (CCL), a step-wise framework that updates node embeddings  
 514 based on the current decision context. Through relevance-guided constraint reformulation and  
 515 trajectory-shared re-embedding, CCL captures the agent’s evolving preferences and improves solution  
 516 quality. Experiments on 48 VRP variants show that CCL achieves SOTA performances on all in-  
 517 distribution and most out-of-distribution tasks. One limitation of CCL lies in its slightly longer  
 518 inference time. Although flexible parameter settings can mitigate such issue, a trade-off between  
 519 computation cost and solution quality still remains. In future, we plan to explore more advanced  
 520 techniques to further improve the inference efficiency while preserving superior solution quality.  
 521

522 Table 5: Results on large-scale real-  
 523 world VRPTW instances ( $N=600$ ).  
 524

Methods	Obj. $\downarrow$	Gap $\downarrow$	Time $\downarrow$
RF-TE	29558	145.593%	<b>1.4s</b>
CaDA	22917	88.188%	<b>1.4s</b>
CCL	<b>20633</b>	<b>70.961%</b>	2.0s

540 REFERENCES  
541

542 Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

543 Grigoris D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and  
544 related algorithms for logistics distribution: A literature review and classification. *Operational research*,  
545 pages 1–30, 2022.

546 Thierry Garaix, Christian Artigues, Dominique Feillet, and Didier Josselin. Vehicle routing problems with  
547 alternative paths: An application to on-demand transportation. *European Journal of Operational Research*,  
548 204(1):62–75, 2010.

549 Rodolfo Dondo, Carlos A Méndez, and Jaime Cerdá. The multi-echelon vehicle routing problem with cross  
550 docking in supply chain management. *Computers & Chemical Engineering*, 35(12):3002–3024, 2011.

551 Laurent Perron and Vincent Furnon. Or-tools. URL <https://developers.google.com/optimization/>.

552 Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations  
553 research*, 21(2):498–516, 1973.

554 Thibaut Vidal, Gilbert Laporte, and Piotr Matl. A concise guide to existing and emerging vehicle routing problem  
555 variants. *European Journal of Operational Research*, 286(2):401–416, 2020.

556 Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique  
557 for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

558 Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint  
559 arXiv:1803.08475*, 2018.

560 Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy  
561 optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing  
562 Systems*, 33:21188–21198, 2020.

563 Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for  
564 solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):5057–5069,  
565 2021.

566 Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing  
567 problems with flexible neural k-opt. *Advances in Neural Information Processing Systems*, 36:49555–49578,  
568 2023.

569 Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances  
570 in Neural Information Processing Systems*, 36:3706–3731, 2023.

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

573 Aigerim Bogyrbayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev. Machine  
574 learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation  
575 Systems*, 25(6):4754–4772, 2024.

576 André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem.  
577 In *ECAI 2020*, pages 443–450. IOS Press, 2020.

578 André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization  
579 problems. *arXiv preprint arXiv:2106.05126*, 2021.

580 Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-  
581 kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information  
582 Processing Systems*, 34:7472–7483, 2021.

583 Felix Chalumeau, Shikha Surana, Clément Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and  
584 Tom Barrett. Combinatorial optimization with policy adaptation using latent space search. *Advances in  
585 Neural Information Processing Systems*, 36:7947–7959, 2023.

586 Jinbiao Chen, Jiahai Wang, Zizhen Zhang, Zhiguang Cao, Te Ye, and Siyuan Chen. Efficient meta neural  
587 heuristic for multi-objective combinatorial optimization. *Advances in Neural Information Processing Systems*,  
588 36:56825–56837, 2023a.

594 Zefang Zong, Meng Zheng, Yong Li, and Depeng Jin. Mapdp: Cooperative multi-agent reinforcement learning  
 595 to solve pickup and delivery problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*,  
 596 volume 36, pages 9980–9988, 2022.

597 Detian Kong, Yining Ma, Zhiguang Cao, Tianshu Yu, and Jianhua Xiao. Efficient neural collaborative search  
 598 for pickup and delivery problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):  
 599 11019–11034, 2024.

600 Karas Tyasnurita, Ender Özcan, John H Drake, and Shahriar Asta. Constructing selection hyper-heuristics for  
 601 open vehicle routing with time delay neural networks using multiple experts. *Knowledge-Based Systems*, 295:  
 602 111731, 2024.

603 Sinaide Nunes Bezerra, Sérgio Ricardo de Souza, and Marcone Jamilson Freitas Souza. A general vns for the  
 604 multi-depot open vehicle routing problem with time windows. *Optimization Letters*, 17(9):2033–2063, 2023.

605 Bruno Oliveira, Artur Pessoa, and Marcos Roboredo. Hybrid iterated local search algorithm for the vehicle  
 606 routing problem with lockers. *Journal of Heuristics*, 31(2):22, 2025.

607 Rongkai Zhang, Cong Zhang, Zhiguang Cao, Wen Song, Puay Siew Tan, Jie Zhang, Bihan Wen, and Justin  
 608 Dauwels. Learning to solve multiple-tsp with time window and rejections via deep reinforcement learning.  
 609 *IEEE Transactions on Intelligent Transportation Systems*, 24(1):1325–1336, 2022.

610 Bo Lin, Bissan Ghaddar, and Jatin Nathwani. Deep reinforcement learning for the electric vehicle routing  
 611 problem with time windows. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11528–11538,  
 612 2021.

613 Conghui Wang, Zhiguang Cao, Yaxin Wu, Long Teng, and Guohua Wu. Deep reinforcement learning for  
 614 solving vehicle routing problems with backhauls. *IEEE Transactions on Neural Networks and Learning  
 615 Systems*, 36(3):4779–4793, 2024.

616 Sašo Karakatič and Vili Podgorelec. A survey of genetic algorithms for solving multi depot vehicle routing  
 617 problem. *Applied Soft Computing*, 27:519–532, 2015.

618 Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Multi-task learning for  
 619 routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD  
 620 Conference on Knowledge Discovery and Data Mining*, pages 1898–1908, 2024.

621 Jianan Zhou, Zhiguang Cao, Yaxin Wu, Wen Song, Yining Ma, Jie Zhang, and Xu Chi. MVMoE: Multi-task  
 622 vehicle routing solver with mixture-of-experts. In *Proceedings of the 41st International Conference on  
 623 Machine Learning*, volume 235, pages 61804–61824, 2024a.

624 Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Kevin Tierney,  
 625 and Jinkyoo Park. Routefinder: Towards foundation models for vehicle routing problems. In *ICML 2024  
 626 Workshop on Foundation Models in the Wild*, 2024a. URL <https://openreview.net/forum?id=hCiaiZ6e4G>.

627 Han Li, Fei Liu, Zhi Zheng, Yu Zhang, and Zhenkun Wang. CaDA: Cross-problem routing solver with  
 628 constraint-aware dual-attention. In *Forty-second International Conference on Machine Learning*, 2025. URL  
 629 <https://openreview.net/forum?id=CS4RyQuTig>.

630 Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Step-wise deep learning models for solving routing  
 631 problems. *IEEE Transactions on Industrial Informatics*, 17(7):4861–4871, 2020.

632 Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park,  
 633 Kevin Tierney, and Jinkyoo Park. Routefinder: Towards foundation models for vehicle routing problems.  
 634 *arXiv preprint arXiv:2406.15007*, 2024b.

635 Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning.  
 636 *Machine Learning*, 8:229–256, 1992.

637 Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information  
 638 Processing Systems*, 32, 2019.

639 Niels A Wouda, Leon Lan, and Wouter Kool. Pyvrp: A high-performance vrp solver package. *INFORMS  
 640 Journal on Computing*, 36(4):943–955, 2024.

648 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen,  
 649 Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary  
 650 DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and  
 651 Soumith Chintala. Pytorch: an imperative style, high-performance deep learning library. In *Advances in  
 652 Neural Information Processing Systems*, pages 8024–8035. Curran Associates, Inc., 2019.

653 Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin XU. Rethinking light decoder-based solvers for vehicle  
 654 routing problems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL  
 655 <https://openreview.net/forum?id=4pRwkYpa2u>.

656 Jörg Homberger and Hermann Gehring. Two evolutionary metaheuristics for the vehicle routing problem with  
 657 time windows. *INFOR: Information Systems and Operational Research*, 37(3):297–318, 1999.

658 Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints.  
 659 *Operations research*, 35(2):254–265, 1987.

660 Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New  
 661 benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*,  
 662 257(3):845–858, 2017.

663 Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takáć. Reinforcement learning for  
 664 solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, 31, 2018.

665 Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Advances in  
 666 Neural Information Processing Systems*, volume 34, pages 26198–26211, 2021a.

667 Qingchun Hou, Jingwei Yang, Yiqiang Su, Xiaoqing Wang, and Yuming Deng. Generalize learned heuris-  
 668 tics to solve large-scale vehicle routing problems in real-time. In *International Conference on Learning  
 669 Representations*, 2023.

670 Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. GLOP: Learning global  
 671 partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the  
 672 AAAI Conference on Artificial Intelligence*, 2024.

673 Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning tsp requires  
 674 rethinking generalization. In *International Conference on Principles and Practice of Constraint Programming*,  
 675 2021.

676 Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. Learning  
 677 generalizable models for vehicle routing problems via knowledge distillation. In *Advances in Neural  
 678 Information Processing Systems*, volume 35, pages 31226–31238, 2022.

679 Simon Geisler, Johanna Sommer, Jan Schuchardt, Aleksandar Bojchevski, and Stephan Günnemann. Generaliza-  
 680 tion of neural combinatorial solvers through the lens of adversarial robustness. In *International Conference  
 681 on Learning Representations*, 2022.

682 Jianan Zhou, Yaxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods  
 683 for vehicle routing problems. In *International Conference on Machine Learning*, pages 42769–42789. PMLR,  
 684 2023.

685 Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding  
 686 networks for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*,  
 687 volume 34, 2021.

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

691 Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging symmetricity for neural combinatorial  
 692 optimization. In *Advances in Neural Information Processing Systems*, volume 35, pages 1936–1949, 2022.

693 Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Thomas D Barrett. Winner  
 694 takes it all: Training performant RL populations for combinatorial optimization. In *Advances in Neural  
 695 Information Processing Systems*, 2023.

696 Jinbiao Chen, Zizhen Zhang, Zhiguang Cao, Yaxin Wu, Yining Ma, Te Ye, and Jiahai Wang. Neural multi-  
 697 objective combinatorial optimization with diversity enhancement. In *Advances in Neural Information  
 698 Processing Systems*, volume 36, pages 39176–39188, 2023b.

702 Chengrui Gao, Haopu Shang, Ke Xue, Dong Li, and Chao Qian. Towards generalizable neural solvers for vehicle  
 703 routing problems via ensemble with transferrable local policy. In *IJCAI*, 2024.

704 André Hottung, Mridul Mahajan, and Kevin Tierney. PolyNet: Learning diverse solution strategies for neural  
 705 combinatorial optimization. In *International Conference on Learning Representations*, 2025.

706 Chuanbo Hua, Federico Berto, Jiwoo Son, Seunghyun Kang, Changhyun Kwon, and Jinkyoo Park. CAMP:  
 707 Collaborative Attention Model with Profiles for Vehicle Routing Problems. In *Proceedings of the 2025*  
 708 *International Conference on Autonomous Agents and Multiagent Systems*, 2025a.

709 Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation  
 710 quotienting for efficient neural combinatorial optimization. In *Advances in Neural Information Processing*  
 711 *Systems*, volume 36, 2023.

712 Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy  
 713 decoder: Toward large scale generalization. In *Advances in Neural Information Processing Systems*, volume 36,  
 714 pages 8845–8864, 2023.

715 Fu Luo, Xi Lin, Yaxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting  
 716 neural combinatorial optimization for large-scale vehicle routing problems. In *International Conference on*  
 717 *Learning Representations*, 2025.

718 Jonathan Pirnay and Dominik G. Grimm. Self-improvement for neural combinatorial optimization: Sample  
 719 without replacement, but improvement. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

720 Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. GOAL: A generalist combinatorial optimization agent  
 721 learner. In *International Conference on Learning Representations*, 2025.

722 Zhuoyi Lin, Yaxin Wu, Bangjian Zhou, Zhiguang Cao, Wen Song, Yingqian Zhang, and Senthilnath Jayavelu.  
 723 Cross-problem learning for solving vehicle routing problems. In Kate Larson, editor, *Proceedings of the*  
 724 *Thirty-Third International Joint Conference on Artificial Intelligence*, pages 6958–6966. International Joint  
 725 Conferences on Artificial Intelligence Organization, 2024.

726 Zefang Zong, Xiaochen Wei, Guozhen Zhang, Chen Gao, Huandong Wang, and Yong Li. Unico: Towards a  
 727 unified model for combinatorial optimization problems. *arXiv preprint arXiv:2505.06290*, 2025.

728 Xia Jiang, Yaxin Wu, Yuan Wang, and Yingqian Zhang. Unco: Towards unifying neural combinatorial  
 729 optimization through large language model. *arXiv preprint arXiv:2408.12214*, 2024.

730 Jianan Zhou, Yaxin Wu, Zhiguang Cao, Wen Song, Jie Zhang, and Zhiqi Shen. Collaboration! towards robust  
 731 neural methods for routing problems. *arXiv preprint arXiv:2410.04968*, 2024b.

732 Chuanbo Hua, Federico Berto, Zhikai Zhao, Jiwoo Son, Changhyun Kwon, and Jinkyoo Park. Uspr: Learning a  
 733 unified solver for profiled routing. *arXiv preprint arXiv:2505.05119*, 2025b.

734 Chuanbo Hua, Federico Berto, Jiwoo Son, Seunghyun Kang, Changhyun Kwon, and Jinkyoo Park. Camp:  
 735 Collaborative attention model with profiles for vehicle routing problems. *arXiv preprint arXiv:2501.02977*,  
 736 2025c.

737 Jiwoo Son, Zhikai Zhao, Federico Berto, Chuanbo Hua, Changhyun Kwon, and Jinkyoo Park. Neural combina-  
 738 torial optimization for real-world routing. *arXiv preprint arXiv:2503.16159*, 2025.

739 Zhenwei Wang, Ruibin Bai, and Tiehua Zhang. Towards constraint-based adaptive hypergraph learning for  
 740 solving vehicle routing: An end-to-end solution. *arXiv preprint arXiv:2503.10421*, 2025.

741 LLC Gurobi Optimization. Gurobi optimizer reference manual, 2024. URL <https://www.gurobi.com>.

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756 A RELATED WORK  
757

758 **Neural Solvers for Single-Task VRPs.** A common paradigm in neural solvers for single-task VRPs  
759 is to construct solutions in an autoregressive manner. These methods typically employ an encoder to  
760 embed the VRP instance into node representations, followed by a decoder that sequentially predicts  
761 the probability of selecting the next node. To reduce the computational overhead of reinforcement  
762 learning (RL), most approaches adopt *static node embeddings* during decoding (Joshi et al., 2019;  
763 Nazari et al., 2018; Kool et al., 2018; Kwon et al., 2020; Huang et al., 2025; Li et al., 2021a; Hou  
764 et al., 2023; Ye et al., 2024; Joshi et al., 2021; Bi et al., 2022; Geisler et al., 2022; Zhou et al.,  
765 2023). One influential method in this line is the Attention Model (AM) (Kool et al., 2018), which  
766 uses a Transformer-based policy network to guide node selection. (Kwon et al., 2020) enhances  
767 AM by introducing Policy Optimization with Multiple Optima (POMO), which leverages multiple  
768 solution trajectories and data augmentation to achieve strong performance on TSP and CVRP. POMO  
769 has since become a widely adopted baseline (Kwon et al., 2021; Li et al., 2021b; Kim et al., 2022;  
770 Grinsztajn et al., 2023; Chen et al., 2023b; Gao et al., 2024; Hottung et al., 2025; Hua et al., 2025a; Li  
771 et al., 2021a; Hou et al., 2023; Ye et al., 2024; Joshi et al., 2021; Bi et al., 2022; Geisler et al., 2022;  
772 Zhou et al., 2023). To improve context modeling, ReLD (Huang et al., 2025) proposes an enhanced  
773 decoder architecture incorporating identity mapping and a feed-forward layer to better capture local  
774 and global dependencies. In terms of *dynamic node re-embedding*, (Xin et al., 2020) introduces  
775 a step-wise RL framework that removes visited nodes at each decision step, enabling the model  
776 to represent distinct node states as the context evolves. An alternative direction involves building  
777 heavy decoder-based solvers trained with supervised learning (SL) (Drakulic et al., 2023; Luo et al.,  
778 2023; 2025; Pirnay and Grimm, 2024; Drakulic et al., 2025). While these methods demonstrate  
779 strong empirical performance, their reliance on multi-layered decoder architectures results in high  
780 computational cost, making them unsuitable for RL-based training, which does not need optimal  
solutions as the labels.

781 **Neural Solvers for Multi-Task VRPs.** Multi-task VRPs generalize single-task VRPs by involving  
782 varied combinations of constraints, resulting in multiple task variants within a shared framework.  
783 Recent works train a single model to capture transferable patterns across tasks (Lin et al., 2024; Liu  
784 et al., 2024; Zhou et al., 2024a; Berto et al., 2024a; Zong et al., 2025; Drakulic et al., 2025; Jiang  
785 et al., 2024; Zhou et al., 2024b; Hua et al., 2025b;c; Li et al., 2025; Berto et al., 2024b; Son et al.,  
786 2025; Wang et al., 2025). (Lin et al., 2024) shows that a pre-trained TSP model can be fine-tuned to  
787 handle other VRP variants. To expand constraint coverage, (Liu et al., 2024) and (Zhou et al., 2024a)  
788 introduce models that handle B, L, O, and TW constraints, training on single-constraint tasks with  
789 the goal of generalizing to tasks with mixed attributes. (Berto et al., 2024a) presents a foundation  
790 model trained on 16 variants and fine-tuned on an unseen constraint (MB) across 8 new variants. A  
791 subsequent extension (Berto et al., 2024b) adds MD to the task space, culminating in a benchmark  
792 of 48 variants. Most recently, (Li et al., 2025) proposes Constraint-Aware Dual-Attention (CaDA),  
793 which incorporates constraint prompts and global-sparse attention to enhance encoder performance  
794 in capturing both broad and localized constraint-relevant node information. Despite their progress,  
795 these methods generally struggle to capture the dynamic, fine-grained impact of constraints during  
796 decision-making—particularly when certain nodes become increasingly urgent due to time-sensitive  
797 or context-dependent requirements. In contrast, our work introduces a step-wise, context-aware  
798 refinement mechanism to better model these evolving constraint-driven priorities.

799 B DETAILS OF MODEL ARCHITECTURE  
800801 B.1 ENCODER ARCHITECTURE  
802

803 As illustrated in Section 4.2, the input attributes includes three parts: the constraint label  $\tilde{\mathbf{h}}_0 \in \mathbb{R}^4$ ,  
804 the depot attribute  $\mathbf{h}_0 \in \mathbb{R}^4$ , and the customer features  $\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_N\} \in \mathbb{R}^{N \times 7}$ . The depot and  
805 customer attributes are separately processed by two linear layers, and the outputs are concatenated to  
806 generate the input node embedding as follows:

$$\mathbf{I} = \text{Concat}(\mathcal{H}(\mathbf{h}_0), \mathcal{H}(\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_N\})), \quad (15)$$

807 where  $\mathbf{I} \in \mathbb{R}^{(N+1) \times D}$ ,  $\text{Concat}(\cdot)$  denotes the concatenate operation, and  $\mathcal{H}(\cdot)$  represents the linear  
808 layer. Similarly, we project the constraint labels into the prompt embedding space and expand them

810 to match the shape of the node embeddings:  
 811

$$812 \quad \mathbf{L} = \text{Expand}(\mathcal{H}(\tilde{\mathbf{h}}_0)), \quad (16)$$

813 where  $\mathbf{L} \in \mathbb{R}^{(N+1) \times D}$ ,  $\mathcal{H}(\cdot)$  is a linear layer to project the feature dimension from 4 to  $D$ , and  
 814  $\text{Expand}(\cdot)$  is the duplication and expansion operation to reshape the feature map from  $\mathbb{R}^{1 \times D}$  to  
 815  $\mathbb{R}^{(N+1) \times D}$ . Subsequently, a unified input embedding is formed by concatenating the projected  
 816 constraint embedding and the node embedding:  
 817

$$818 \quad \tilde{\mathbf{I}} = \mathcal{H}(\text{Concat}(\mathbf{I}, \mathbf{L})). \quad (17)$$

820 The dual-attention encoder processes the original node embeddings  $\mathbf{I}$  and the unified input embeddings  
 821  $\tilde{\mathbf{I}}$  through the sparse and global branches, respectively. Each branch contains a Transformer layer and  
 822 a linear layer for fusion, with the overall computation defined as:  
 823

$$824 \quad \tilde{\mathbf{H}}'^{(i)} = \mathcal{T}_g^{(i)}(\tilde{\mathbf{H}}^{(i-1)}), \quad \mathbf{H}'^{(i)} = \mathcal{T}_s^{(i)}(\mathbf{H}^{(i-1)}), \quad (18)$$

$$825 \quad \tilde{\mathbf{H}}^{(i)} = \tilde{\mathbf{H}}'^{(i)} + \mathcal{H}_g^{(i)}(\mathbf{H}'^{(i)}), \quad \mathbf{H}^{(i)} = \mathbf{H}'^{(i)} + \mathcal{H}_s^{(i)}(\tilde{\mathbf{H}}'^{(i)}),$$

827 where  $\tilde{\mathbf{H}}'$ ,  $\tilde{\mathbf{H}}$ ,  $\mathbf{H}'$ ,  $\mathbf{H} \in \mathbb{R}^{(N+1) \times D}$ , and  $i$  denotes the index of the encoder layer. In the first layer,  
 828 we initialize  $\tilde{\mathbf{H}}^{(1)} = \tilde{\mathbf{I}}$  and  $\mathbf{H}^{(1)} = \mathbf{I}$ .  $\mathcal{T}_g^{(i)}$ ,  $\mathcal{H}_g^{(i)}$  denote the Transformer and linear layers of the global  
 829 branch, respectively, while  $\mathcal{T}_s^{(i)}$ ,  $\mathcal{H}_s^{(i)}$  correspond to those of the sparse branch. The Transformer  
 830 layer employs the pre-norm design from (Berto et al., 2024a), and integrates sparse attention based  
 831 on (Li et al., 2025). The embedding output by the final global layer is passed through a normalization  
 832 layer, and the normalized embeddings are used as the initial node embeddings for decoding:  
 833

$$834 \quad \mathbf{H} = \text{Norm}(\mathbf{H}^{(K)}), \quad (19)$$

835 where  $\mathbf{H} \in \mathbb{R}^{(N+1) \times D}$  and  $K=6$  denotes the number of encoder layers.  
 836

## 837 B.2 CLASSIC DECODER

839 Following RGCR and TSNR, a classic decoder is employed to calculate the action probability  
 840 distribution using a multi-head attention mechanism. At step  $j$ , the context embedding generated by  
 841 RGCR is denoted as  $\tilde{\mathbf{C}}_j \in \mathbb{R}^{N \times D}$ , where  $N$  is the number of trajectories, equal to the number of  
 842 customers. We directly use the context embeddings as the query, *i.e.*,  $\mathbf{q}_j = \tilde{\mathbf{C}}_j$ . In the multi-head  
 843 attention mechanism, the key and value embeddings are derived from the node embeddings produced  
 844 by TSNR, *i.e.*,  $\mathbf{k}_j, \mathbf{v}_j = \mathcal{H}(\mathbf{H}_j)$ , where  $\mathbf{k}_j, \mathbf{v}_j \in \mathbb{R}^{(N+1) \times D}$ . Since each node is visited only once,  
 845 we apply a mask  $\mathbf{M}_j$  to the visited nodes when computing the attention weights:  
 846

$$847 \quad \mathbf{A}_j = \text{Softmax}\left(\frac{\mathbf{q}_j \mathbf{k}_j^\top}{\sqrt{D}} \odot \mathbf{M}_j\right), \quad (20)$$

850 where  $\mathbf{A}_j, \mathbf{M}_j \in \mathbb{R}^{N \times (N+1)}$ ,  $\text{Softmax}(\cdot)$  denotes the Softmax operation, and  $\odot$  ensures that  
 851 multiplication values for visited nodes are set to  $-\infty$ . The context query is computed as  $\tilde{\mathbf{q}}_j =$   
 852  $\mathcal{H}(\mathbf{A}_j \mathbf{v}_j)$ , and the candidate node representations are obtained as  $\tilde{\mathbf{k}}_j = \mathcal{H}(\mathbf{H}_j)$ . Based on these, the  
 853 action probability distribution is derived as follows:  
 854

$$855 \quad \mathbf{D}_j = \frac{\tilde{\mathbf{q}}_j \tilde{\mathbf{k}}_j^\top}{\sqrt{D}}, \quad (21)$$

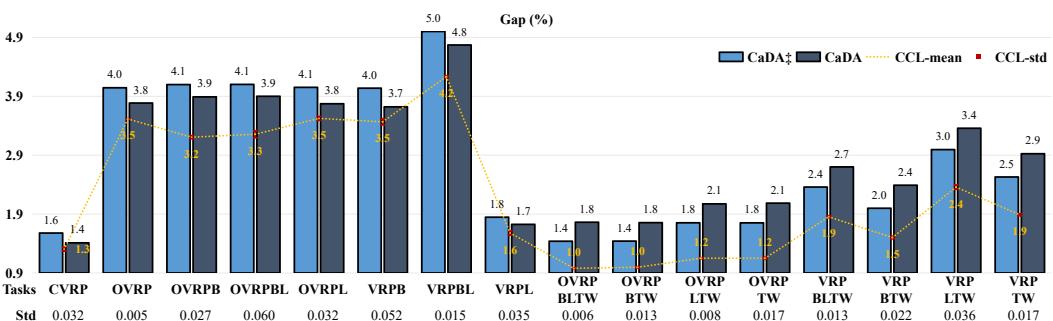
857 where  $\tilde{\mathbf{q}}_j \in \mathbb{R}^{N \times D}$ ,  $\tilde{\mathbf{k}}_j \in \mathbb{R}^{(N+1) \times D}$ , and  $\mathbf{D}_j \in \mathbb{R}^{N \times N+1}$ . To generate solutions, the unnormalized  
 858 log-probability (logit) is calculated as  
 859

$$860 \quad \mathbf{u}_j = \xi \cdot \text{Tanh}(\mathbf{D}_j) \odot \mathbf{M}_j, \quad (22)$$

862 where  $\text{Tanh}(\cdot)$  is the Hyperbolic Tangent operation, and  $\xi=10$  is a predefined clipping hyperparameter.  
 863 The final selection probabilities for each node are computed by applying the Softmax operation:  
 $\mathbf{P}_j = \text{Softmax}(\mathbf{u}_j)$ .  
 864

864  
865 Table 5: Performance on 16 seen in-distribution tasks. \* denotes the strong baseline used to compute  
866 the gap. Best neural approach is highlighted in **bold**; second underlined.

Methods	N = 50			N = 100			Methods	N = 50			N = 100			
	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓		Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	
CVRP	HGS-PyVRP	10.372	*	10.4m	15.628	*	20.8m	HGS-PyVRP	16.031	*	10.4m	25.423	*	20.8m
	CaDA <sup>‡</sup>	<u>10.494</u>	<u>1.182%</u>	2s	15.870	<u>1.578%</u>	8s	CaDA <sup>‡</sup>	<u>16.278</u>	<u>1.536%</u>	2s	<u>26.070</u>	<u>2.530%</u>	8s
	CaDA	10.505	1.287%	2s	<u>15.843</u>	<u>1.412%</u>	8s	CaDA	16.312	1.745%	1s	26.169	2.925%	9s
	CCL	<b>10.473</b>	<b>0.977%</b>	5s	<b>15.823</b>	<b>1.287%</b>	19s	CCL	<b>16.190</b>	<b>0.979%</b>	5s	<b>25.913</b>	<b>1.908%</b>	21s
OVRP	HGS-PyVRP	6.507	*	10.4m	9.725	*	20.8m	HGS-PyVRP	10.510	*	10.4m	16.926	*	20.8m
	CaDA <sup>‡</sup>	<u>6.670</u>	<u>2.468%</u>	2s	10.121	4.045%	8s	CaDA <sup>‡</sup>	<u>10.613</u>	<u>0.957%</u>	2s	<u>17.226</u>	<u>1.751%</u>	9s
	CaDA	6.677	2.585%	1s	<u>10.095</u>	<u>3.786%</u>	8s	CaDA	10.630	1.122%	1s	17.283	2.086%	9s
	CCL	<b>6.636</b>	<b>1.957%</b>	5s	<b>10.068</b>	<b>3.511%</b>	20s	CCL	<b>10.569</b>	<b>0.543%</b>	6s	<b>17.123</b>	<b>1.142%</b>	21s
OVRPB	HGS-PyVRP	6.898	*	10.4m	10.335	*	20.8m	HGS-PyVRP	11.669	*	10.4m	19.156	*	20.8m
	CaDA <sup>‡</sup>	<u>7.049</u>	<u>2.159%</u>	2s	10.762	4.099%	8s	CaDA <sup>‡</sup>	<u>11.761</u>	<u>0.779%</u>	2s	<u>19.436</u>	<u>1.441%</u>	9s
	CaDA	7.064	2.377%	1s	<u>10.739</u>	<u>3.890%</u>	8s	CaDA	11.775	0.898%	2s	19.495	1.754%	9s
	CCL	<b>7.008</b>	<b>1.568%</b>	5s	<b>10.666</b>	<b>3.179%</b>	19s	CCL	<b>11.721</b>	<b>0.436%</b>	6s	<b>19.348</b>	<b>0.985%</b>	21s
OVRPBL	HGS-PyVRP	6.899	*	10.4m	10.335	*	20.8m	HGS-PyVRP	11.668	*	10.4m	19.156	*	20.8m
	CaDA <sup>‡</sup>	<u>7.051</u>	<u>2.166%</u>	2s	10.762	4.102%	8s	CaDA <sup>‡</sup>	<u>11.760</u>	<u>0.771%</u>	2s	<u>19.435</u>	<u>1.439%</u>	9s
	CaDA	7.062	2.339%	1s	<u>10.741</u>	<u>3.900%</u>	8s	CaDA	11.777	0.914%	2s	19.497	1.762%	9s
	CCL	<b>7.009</b>	<b>1.569%</b>	5s	<b>10.681</b>	<b>3.323%</b>	20s	CCL	<b>11.721</b>	<b>0.442%</b>	6s	<b>19.346</b>	<b>0.977%</b>	22s
OVRPL	HGS-PyVRP	6.507	*	10.4m	9.724	*	20.8m	HGS-PyVRP	10.510	*	10.4m	16.926	*	20.8m
	CaDA <sup>‡</sup>	<u>6.671</u>	<u>2.475%</u>	2s	10.122	4.052%	8s	CaDA <sup>‡</sup>	<u>10.613</u>	<u>0.961%</u>	2s	<u>17.226</u>	<u>1.752%</u>	9s
	CaDA	6.680	2.623%	1s	<u>10.093</u>	<u>3.773%</u>	8s	CaDA	10.631	1.133%	1s	17.280	2.073%	9s
	CCL	<b>6.637</b>	<b>1.968%</b>	5s	<b>10.067</b>	<b>3.495%</b>	20s	CCL	<b>10.569</b>	<b>0.546%</b>	6s	<b>17.123</b>	<b>1.143%</b>	22s
VRPB	HGS-PyVRP	9.687	*	10.4m	14.377	*	20.8m	HGS-PyVRP	18.292	*	10.4m	29.467	*	20.8m
	CaDA <sup>‡</sup>	<u>9.960</u>	<u>2.800%</u>	2s	14.960	4.038%	8s	CaDA <sup>‡</sup>	<u>18.500</u>	<u>1.117%</u>	2s	<u>30.059</u>	<u>1.999%</u>	9s
	CaDA	9.979	3.010%	1s	14.910	3.721%	8s	CaDA	18.543	1.361%	1s	30.174	2.390%	9s
	CCL	<b>9.916</b>	<b>2.352%</b>	5s	<b>14.882</b>	<b>3.526%</b>	19s	CCL	<b>18.430</b>	<b>0.738%</b>	6s	<b>29.911</b>	<b>1.494%</b>	21s
VRPBL	HGS-PyVRP	10.186	*	10.4m	14.779	*	20.8m	HGS-PyVRP	18.361	*	10.4m	29.026	*	20.8m
	CaDA <sup>‡</sup>	<u>10.543</u>	<u>3.461%</u>	2s	15.525	5.001%	8s	CaDA <sup>‡</sup>	<u>18.848</u>	<u>1.376%</u>	2s	<u>30.520</u>	<u>2.359%</u>	9s
	CaDA	10.576	3.776%	1s	<u>15.490</u>	<u>4.771%</u>	8s	CaDA	18.894	1.623%	1s	30.620	2.700%	9s
	CCL	<b>10.484</b>	<b>2.883%</b>	5s	<b>15.407</b>	<b>4.219%</b>	19s	CCL	<b>18.773</b>	<b>0.976%</b>	6s	<b>30.366</b>	<b>1.842%</b>	21s
VRPL	HGS-PyVRP	10.587	*	10.4m	15.766	*	20.8m	HGS-PyVRP	16.356	*	10.4m	25.757	*	20.8m
	CaDA <sup>‡</sup>	<u>10.731</u>	<u>1.333%</u>	2s	16.057	1.847%	8s	CaDA <sup>‡</sup>	<u>16.669</u>	<u>1.879%</u>	2s	<u>26.540</u>	<u>2.995%</u>	9s
	CaDA	10.749	1.505%	1s	<u>16.036</u>	<u>1.725%</u>	8s	CaDA	16.709	2.130%	1s	26.631	3.358%	9s
	CCL	<b>10.710</b>	<b>1.145%</b>	5s	<b>16.009</b>	<b>1.561%</b>	19s	CCL	<b>16.579</b>	<b>1.333%</b>	6s	<b>26.366</b>	<b>2.321%</b>	20s
VRP	HGS-PyVRP	8.455	*	10.4m	12.584	*	20.8m	HGS-PyVRP	14.175	*	10.4m	22.730	*	20.8m
	CaDA <sup>‡</sup>	<u>8.646</u>	<u>2.256%</u>	2s	13.022	3.595%	8s	CaDA <sup>‡</sup>	<u>14.380</u>	<u>1.172%</u>	2s	<u>23.314</u>	<u>2.033%</u>	9s
	CaDA	8.662	2.437%	1s	<u>12.993</u>	<u>3.372%</u>	8s	CaDA	14.409	1.366%	1s	23.394	2.381%	9s
	CCL	<b>8.609</b>	<b>1.802%</b>	5s	<b>12.950</b>	<b>3.013%</b>	19s	CCL	<b>14.319</b>	<b>0.749%</b>	6s	<b>23.187</b>	<b>1.476%</b>	21s

Figure 4: Error-bar analysis of CCL under  $N=100$ .

## C ADDITIONAL ANALYSES AND DISCUSSIONS

### C.1 COMPARISON WITH NEURAL SOTA METHODS

#### C.1.1 COMPARISON BETWEEN RE-IMPLEMENTED SOTA AND REPORTED SOTA

**Main Results.** We compare CCL against the reported SOTA method (CaDA<sup>‡</sup> (Li et al., 2025)) and our re-implemented one (CaDA (Li et al., 2025)), with the strong heuristic baseline HGS-PyVRP (Wouda et al., 2024) included for reference. Table 5 presents the results for all 16 in-distribution tasks, and the overall average scores across these tasks both with and without the TW constraint. Under the  $N=100$

918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
Table 6: Improvement of  $CCL^\dagger$  over  $CaDA^\dagger$ .

Tasks	$\Delta$	P-value	Tasks	$\Delta$	P-value	Tasks	$\Delta$	P-value	Tasks	$\Delta$	P-value
CVRP	8.16%	2.7e-05	OVRP	29.21%	1.2e-76	VRPBBLTW	41.61%	2.2e-68	OVRPBBLTW	51.19%	6.2e-68
VRPB	20.14%	5.0e-42	OVRPB	29.86%	1.4e-62	VRPBTLW	46.84%	4.0e-81	OVRPBTLW	50.63%	2.8e-68
VRPBL	20.03%	2.0e-39	OVRPBL	31.05%	4.7e-68	VRPLTLW	41.39%	2.2e-97	OVRPLTLW	51.50%	2.9e-79
VRPL	7.66%	2.4e-04	OVRPL	28.69%	3.5e-74	VRPTWL	45.72%	2.9e-99	OVRPTWL	50.85%	3.2e-75

Table 7: Per-task results on 32 unseen out-of-distribution tasks ( $N=50$ ).

Tasks	RF-TE			CaDA			CaDA $^\dagger$			CCL			CCL $^\dagger$		
	Obj. $\downarrow$	Gap $\downarrow$	Time $\downarrow$	Obj. $\downarrow$	Gap $\downarrow$	Time $\downarrow$	Obj. $\downarrow$	Gap $\downarrow$	Time $\downarrow$	Obj. $\downarrow$	Gap $\downarrow$	Time $\downarrow$	Obj. $\downarrow$	Gap $\downarrow$	Time $\downarrow$
VRPMB	9.879	8.861%	1s	9.781	7.749%	2s	<b>9.722</b>	<b>7.097%</b>	<b>3s</b>	9.943	9.538%	3s	9.940	9.486%	4s
MDCVRP	12.559	56.957%	2s	12.335	54.083%	2s	13.593	70.078%	4s	10.829	34.846%	4s	<b>10.554</b>	<b>31.361%</b>	6s
MDOVRP	6.876	29.051%	1s	6.825	28.069%	2s	<b>6.260</b>	<b>17.329%</b>	<b>3s</b>	6.413	20.155%	4s	6.286	17.719%	5s
MDVRPB	12.725	60.956%	2s	12.654	60.054%	2s	12.190	54.066%	3s	11.679	47.450%	4s	<b>11.001</b>	<b>38.750%</b>	5s
MDVRPL	12.618	57.370%	2s	12.426	54.988%	2s	13.433	67.697%	4s	11.278	40.248%	4s	<b>10.912</b>	<b>35.667%</b>	6s
OVRPMB	6.949	13.690%	1s	6.872	12.440%	1s	<b>6.819</b>	<b>11.570%</b>	<b>3s</b>	6.917	13.149%	3s	6.854	12.124%	4s
VRPMBL	10.239	8.050%	1s	10.163	7.215%	2s	<b>10.061</b>	<b>6.130%</b>	<b>3s</b>	10.376	9.456%	3s	10.229	7.898%	5s
MDOVRPB	7.556	31.861%	1s	7.622	33.009%	2s	6.999	22.032%	3s	6.876	19.787%	4s	<b>6.738</b>	<b>17.332%</b>	5s
MDOVRPL	6.871	28.946%	1s	6.807	27.746%	2s	<b>6.270</b>	<b>17.504%</b>	<b>3s</b>	6.442	20.716%	4s	6.303	18.070%	5s
MDVRPBL	12.831	61.175%	2s	12.585	58.011%	2s	12.017	50.770%	3s	11.483	43.846%	4s	<b>11.208</b>	<b>40.413%</b>	6s
MDVRPMB	12.856	76.544%	2s	12.493	71.550%	2s	12.046	65.185%	3s	11.540	58.118%	4s	<b>11.032</b>	<b>51.055%</b>	6s
MDVRPTW	17.818	48.941%	2s	16.971	41.739%	2s	17.581	46.865%	4s	16.107	34.403%	4s	<b>15.475</b>	<b>29.004%</b>	6s
OVRPMBL	6.949	13.686%	1s	6.871	12.423%	1s	6.818	11.563%	3s	6.932	13.404%	3s	<b>6.819</b>	<b>11.555%</b>	4s
VRPMBTW	17.298	8.074%	1s	17.198	7.434%	2s	17.282	7.954%	3s	<b>16.988</b>	<b>6.099%</b>	3s	17.158	7.142%	5s
MDOVRPBL	7.550	31.772%	1s	7.606	32.713%	2s	6.996	21.966%	3s	6.858	19.463%	4s	<b>6.827</b>	<b>18.939%</b>	6s
MDOVRPMB	7.617	47.411%	1s	7.541	45.953%	2s	6.845	32.344%	3s	6.720	29.826%	4s	<b>6.519</b>	<b>25.920%</b>	5s
MDOVRPTW	10.618	34.976%	1s	10.204	29.610%	2s	10.407	32.287%	4s	9.783	24.150%	4s	<b>9.632</b>	<b>22.146%</b>	5s
MDVRBTLW	18.591	37.364%	2s	19.025	40.629%	2s	19.977	47.721%	4s	18.425	36.029%	4s	<b>17.645</b>	<b>30.121%</b>	5s
MDVRPLTW	18.127	51.276%	2s	17.125	42.765%	2s	17.965	49.851%	4s	16.769	39.728%	5s	<b>16.126</b>	<b>34.232%</b>	6s
MDVRPMBL	12.744	74.112%	2s	12.434	69.825%	2s	11.647	58.969%	3s	11.474	56.409%	4s	<b>10.993</b>	<b>49.737%</b>	6s
OVRPMBTW	11.132	6.265%	1s	11.087	5.849%	2s	11.121	6.160%	3s	10.966	4.658%	3s	<b>10.855</b>	<b>3.617%</b>	5s
VRPMBLTW	17.597	7.982%	1s	17.495	7.337%	2s	17.559	7.728%	3s	17.514	7.433%	3s	<b>17.402</b>	<b>6.710%</b>	5s
MDOVRPBTLW	11.399	32.332%	2s	11.190	29.807%	2s	11.423	32.600%	4s	10.774	24.830%	4s	<b>10.313</b>	<b>19.359%</b>	5s
MDOVRPLTW	10.599	34.731%	2s	10.196	29.501%	2s	10.408	32.286%	4s	9.721	23.331%	4s	<b>9.704</b>	<b>23.040%</b>	5s
MDOVRPMBL	7.602	47.108%	1s	7.540	45.940%	2s	6.848	32.401%	3s	6.733	30.088%	4s	<b>6.548</b>	<b>26.474%</b>	5s
MDVRBTLW	19.048	40.583%	2s	19.243	42.015%	2s	20.076	48.201%	4s	18.544	36.702%	5s	<b>17.612</b>	<b>29.641%</b>	7s
MDVRPMBTW	17.830	48.485%	2s	18.394	53.345%	2s	19.327	61.083%	4s	17.103	42.314%	5s	<b>16.729</b>	<b>39.084%</b>	5s
OVRPMBLTW	11.138	6.317%	1s	11.090	5.875%	2s	11.116	6.109%	3s	10.993	4.922%	3s	<b>10.835</b>	<b>3.427%</b>	5s
MDOVRPBLTW	11.389	32.207%	2s	11.185	29.743%	2s	11.391	32.216%	4s	10.767	24.749%	4s	<b>10.350</b>	<b>19.783%</b>	5s
MDOVRPMBTW	11.055	40.153%	2s	10.833	37.232%	2s	11.150	41.336%	4s	10.171	38.688%	4s	<b>10.110</b>	<b>27.872%</b>	5s
MDVRPMBLTW	18.222	51.554%	2s	18.511	54.026%	2s	19.362	61.119%	4s	17.940	49.174%	5s	<b>16.825</b>	<b>39.635%</b>	6s
MDOVRPMBLTW	11.054	40.136%	2s	10.818	37.044%	2s	11.135	41.125%	4s	10.203	29.114%	4s	<b>9.804</b>	<b>23.943%</b>	6s
Avg. Gap		36.529%			34.866%			34.417%			27.588%			<b>24.102%</b>	
# Best (Best/Total)		0/32			0/32			5/32			1/32			26/32	

without TW, CaDA outperforms the reported  $CaDA^\ddagger$ , while in all other settings it is slightly inferior to  $CaDA^\ddagger$ . In contrast, CCL consistently surpasses both CaDA and  $CaDA^\ddagger$  across all 16 tasks.

**Error Bar Analysis.** Since the testing update probability  $P_{ts}$  is set to 0.5 for  $N=100$ , we further analyze the error bars of CCL under different random seeds. Fig. 4 plots the mean gap and its standard deviation over three independent test runs of CCL. Across all 16 tasks, the standard deviation of CCL’s gap remains tightly bound between 0.005% and 0.060%. Visually, the error bars in Figure 5 are negligible compared to the performance difference between CCL and CaDA/ $CaDA^\ddagger$ , indicating that the choice of seed has minimal impact on test-time results.

### C.1.2 STATISTICAL SIGNIFICANCE

We continue to include t-tests to assess statistical significance. We first collected the gap values of 1,000 test instances from both CCL $^\dagger$  and the strongest SOTA  $CaDA^\dagger$  from Table 1, then we report the improvement percentages ( $\Delta$ ) along with the corresponding p-values (shown in Table 6). Here, the improvements are computed as the average gap reductions of CCL $^\dagger$  over  $CaDA^\dagger$ , *i.e.*,  $-(\text{Gap}(\text{CCL}^\dagger) - \text{Gap}(\text{CaDA}^\dagger)) / \text{Gap}(\text{CaDA}^\dagger) \times 100\%$ . Across all 16 tasks, CCL achieves 7-51% improvement. In particular, OVRPBBLTW, OVRPBTLW, OVRPLTLW, and OVRPTWL exceed 50%. All p-values are below 0.001, indicating that these gains are statistically significant.

### C.1.3 DETAILED RESULTS ON UNSEEN OUT-OF-DISTRIBUTION TASKS

We provide per-task results on 32 unseen out-of-distribution tasks. Each method is evaluated in a zero-shot setting (*i.e.*, directly tested without fine-tuning). For the  $N=50$  setting, the test-time update probability is set to  $P_{ts}=0.15$ , and for  $N=100$ , it is set to  $P_{ts}=0.02$ . Table 7 and Table 8 present the full results under the  $N=50$  and  $N=100$  settings, respectively. For each task, we report the objective (Obj.), performance gap (Gap), and inference time (Time). Additionally, the bottom rows summarize

972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
Table 8: Per-task results on 32 unseen out-of-distribution tasks ( $N=100$ ).  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

Tasks	RF-TE			CaDA			CaDA <sup>†</sup>			CCL			CCL <sup>†</sup>		
	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓
VRPMB	14.888	10.189%	8s	14.710	8.822%	8s	<b>14.652</b>	<b>8.399%</b>	13s	15.322	13.438%	14s	14.936	10.530%	18s
MDCVRP	19.684	67.107%	10s	20.628	75.392%	11s	20.964	78.117%	18s	<b>16.769</b>	<b>41.675%</b>	17s	16.834	42.305%	24s
MDOVRP	10.683	34.368%	9s	10.605	33.387%	10s	9.849	23.695%	15s	10.095	26.715%	15s	<b>9.753</b>	<b>22.409%</b>	20s
MDVRPB	18.721	61.761%	9s	19.494	68.604%	11s	18.780	62.221%	17s	<b>18.185</b>	<b>56.977%</b>	19s	18.625	60.931%	25s
MDVRPL	20.100	70.498%	10s	20.867	77.336%	12s	21.001	78.277%	18s	<b>17.494</b>	<b>47.699%</b>	24s	18.407	55.828%	29s
OVRPMB	10.711	18.899%	7s	10.490	16.449%	8s	<b>10.468</b>	<b>16.205%</b>	13s	10.780	19.642%	14s	10.744	19.235%	19s
VRPMBL	15.198	10.375%	7s	15.016	9.011%	8s	<b>14.949</b>	<b>8.536%</b>	13s	15.761	14.464%	14s	15.538	12.835%	18s
MDOVRPB	11.752	35.659%	9s	11.790	36.137%	10s	11.104	28.078%	16s	10.959	26.274%	16s	<b>10.533</b>	<b>21.340%</b>	21s
MDOVRPL	10.703	34.620%	9s	10.574	32.985%	10s	9.854	23.749%	15s	10.272	29.033%	16s	<b>9.718</b>	<b>21.967%</b>	20s
MDVRPBL	19.606	68.898%	11s	19.827	70.875%	13s	18.693	60.860%	18s	18.446	58.520%	21s	<b>17.588</b>	<b>51.069%</b>	28s
MDVRPMB	18.698	76.325%	9s	19.458	83.817%	11s	18.374	73.239%	16s	18.997	79.149%	18s	<b>17.544</b>	<b>65.282%</b>	23s
MDVRPTW	29.468	53.283%	11s	<b>25.955</b>	<b>34.783%</b>	10s	28.751	49.517%	17s	26.901	39.557%	18s	29.016	50.829%	28s
OVRPMBL	10.709	18.877%	7s	10.486	16.399%	8s	<b>10.470</b>	<b>16.225%</b>	13s	10.883	20.748%	14s	10.749	19.284%	19s
VRPMBTW	28.256	10.840%	8s	28.317	11.074%	9s	28.310	11.038%	14s	28.100	10.239%	15s	<b>27.971</b>	<b>9.722%</b>	20s
MDOVRPBL	11.761	35.771%	9s	11.788	36.106%	10s	11.105	28.077%	16s	10.983	26.545%	16s	<b>10.511</b>	<b>21.090%</b>	21s
MDOVRPMB	11.748	53.650%	9s	11.692	53.010%	10s	10.945	43.077%	16s	10.876	42.044%	17s	<b>10.431</b>	<b>36.189%</b>	21s
MDOVRPTW	18.299	41.356%	10s	17.205	32.804%	10s	17.636	36.201%	17s	17.009	31.200%	17s	<b>16.733</b>	<b>29.076%</b>	23s
MDOVRPTBW	30.681	39.926%	10s	<b>30.176</b>	<b>37.559%</b>	10s	31.716	44.719%	19s	30.677	39.759%	19s	31.269	42.473%	30s
MDVRPLTW	29.640	53.976%	11s	<b>26.200</b>	<b>35.862%</b>	10s	29.327	52.284%	17s	27.977	45.055%	23s	29.593	53.413%	31s
MDVRPMBL	19.216	80.892%	11s	19.460	83.444%	13s	18.183	71.054%	18s	18.479	73.647%	19s	<b>18.112</b>	<b>70.333%</b>	32s
OVRPMBTW	18.449	8.724%	8s	18.478	8.901%	9s	18.430	8.607%	15s	18.427	8.590%	16s	<b>18.211</b>	<b>7.321%</b>	21s
VRPMBLTW	28.604	10.805%	8s	28.658	11.002%	9s	28.641	10.925%	14s	<b>28.374</b>	<b>9.907%</b>	15s	28.582	10.698%	20s
MDOVRPBTW	19.590	36.897%	10s	19.305	34.918%	10s	19.341	35.157%	18s	18.764	30.960%	17s	<b>18.265</b>	<b>27.473%</b>	26s
MDOVRPLTW	18.232	40.817%	10s	17.221	32.923%	10s	17.665	36.428%	17s	16.838	29.857%	18s	<b>16.707</b>	<b>28.872%</b>	23s
MDOVRPMBL	11.751	53.691%	9s	11.670	52.726%	10s	10.931	42.890%	16s	10.830	41.416%	16s	<b>10.415</b>	<b>35.973%</b>	21s
MDVRPLTW	31.044	41.408%	10s	<b>30.537</b>	<b>39.033%</b>	10s	32.239	46.923%	18s	30.667	39.535%	24s	34.176	55.800%	32s
MDVRPMBTW	29.650	54.395%	10s	29.383	53.001%	10s	30.722	60.046%	18s	<b>28.388</b>	<b>47.661%</b>	18s	30.844	60.585%	26s
OVRPMBLTW	18.452	8.739%	8s	18.476	8.887%	9s	18.415	8.516%	15s	<b>18.404</b>	<b>8.443%</b>	16s	18.476	8.877%	20s
MDOVRPBLTW	19.553	36.632%	10s	19.341	35.154%	10s	19.349	35.212%	18s	18.853	31.599%	17s	<b>18.194</b>	<b>26.963%</b>	25s
MDOVRPMBTW	18.888	46.216%	10s	18.735	45.019%	10s	18.761	45.232%	17s	17.823	37.814%	18s	<b>17.638</b>	<b>36.412%</b>	23s
MDVRPMBLTW	29.825	55.132%	10s	29.611	53.989%	11s	31.157	62.147%	18s	<b>29.000</b>	<b>50.703%</b>	23s	31.057	61.451%	35s
MDOVRPMBLTW	18.846	45.873%	10s	18.770	45.286%	10s	18.787	45.427%	17s	17.803	37.665%	17s	<b>17.667</b>	<b>36.646%</b>	24s
Avg. Gap	41.144%		39.834%			39.096%			34.892%			<b>34.788%</b>			
# Best (Best/Total)	0/32		4/32			4/32			7/32			17/32			

each method's average gap and the number of tasks where it achieves the best performance, reported in the format "# Best (best/total)". Across both the  $N=50$  and  $N=100$  settings, CCL<sup>†</sup> consistently achieves the lowest average gap, demonstrating strong generalization to unseen out-of-distribution tasks. In terms of per-task performance, CCL and CCL<sup>†</sup> together outperform all baselines on 27 out of 32 tasks for  $N=50$ , and on 24 out of 32 tasks for  $N=100$ , further highlighting the robustness and effectiveness of our method across different problem scales.

## C.2 COMPARISON WITH TRADITIONAL SOLVER

We follow RouteFinder (Berto et al., 2024a;b) and CaDA (Li et al., 2025) in using HGS-PyVRP (Wouda et al., 2024) as a strong traditional solver. Moreover, we compare our method against additional traditional solvers, including Gurobi (Gurobi Optimization, 2024) and LKH (Lin and Kernighan, 1973) on CVRP instances with 50 customers. The total times, denoted as "Time", are accumulated over 1000 instances, which are exactly the same as the ones used in (Berto et al., 2024a;b; Li et al., 2025). We also provide the average per-instance time for reference, denoted as "Avg. Time". As shown in Table 9, the results of HGS-PyVRP are taken from (Li et al., 2025), while the results of Gurobi and LKH are obtained using a 32-core CPU. Based on this, Gurobi further uses 4 threads per CPU core, enabling 4×32 instances to be solved in parallel. LKH is executed for 10 runs, with a 10-second time limit per instance. We set a 15-minute limit on Gurobi and report its generated (approximate) solutions. These results show that CCL achieves performance comparable to traditional solvers, while its total inference time is approximately 10x, 1,200x, and 100x faster than LKH, Gurobi, and HGS-PyVRP, respectively (corresponding per-instance speedups of 5x, 75x, and 50x). This demonstrates that learning-based models are practical for real-time applications, especially when solving multiple VRP instances simultaneously. Moreover, in multi-task scenarios, CCL can learn generalizable patterns across different VRP variants, without requiring experts to manually design heuristics. Once trained, the model can solve 48 VRP variants without re-training, which will broaden its practical deployment. These findings show that both traditional algorithms and our learning-based method have their own merits and demerits. Research in either direction provides important insights for the VRP community.

996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
Table 9: Comparison with traditional solvers.

Methods	Obj. ↓	Time ↓	Avg. Time ↓
HGS-PyVRP	10.372	10m	10.0s
Gurobi-15m	10.568	120m	15.0m
LKH	10.392	63s	1.1s
CCL <sup>†</sup>	10.463	6s	0.2s

1026  
1027

### C.3 ABLATION RESULTS

1028  
1029

We conducted ablation studies to examine whether the effectiveness of our method depends on ReLD (Huang et al., 2025). Starting from  $CCL^\dagger$ , we remove ReLD, RGCR, TSNR, and both RGCR and TSNR. The corresponding average results across 16 in-distribution tasks are presented in Table 10. The results show that both RGCR and TSNR remain effective even without ReLD. Moreover,  $CCL$  reduces the average gap by 0.6% compared with  $CCL$ -TSNR-RGCR, whereas  $CCL^\dagger$  provides only a 0.18% improvement over  $CCL$ . This indicates that the main performance improvement comes from  $CCL$  rather than ReLD.

1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040

### C.3.1 ABLATION RESULTS WITHOUT ReLD

1041  
1042

To validate the effectiveness of RGCR, we test 16 in-domain VRP variants, each with 1,000 instances, and compare RGCR with four alternatives: (1) "Concat Attributes", which directly concatenates the constraint attributes; (2) "Concat Embeddings", which embeds each constraint into a high-dimensional space and concatenates them; (3) "+ Random Scores" employ random importance weights as the correlation scores; and (4) "+ Cosine Similarity" uses cosine similarity to measure the correlation scores. We present both the model complexity and performance results on the left of Fig. 2. Specifically, "# Params" denotes the total number of parameters in the encoder and decoder, and "Time" is the accumulated inference time over 1,000 instances. "Avg. Gap" denotes the average gap across all 16 tasks, while "w/ TW" and "w/o TW" refer to the subsets of 8 tasks with and without time-window constraints, respectively. Compared with concatenating attributes, RGCR achieves strong performance while increasing the model size by only 0.1M parameters and adding 1.3s to inference time. Compared to random weights, RGCR shows modest performance in average gap across the 16 tasks, but demonstrates clear superiority on tasks with time windows. These results demonstrate that RGCR benefits more on complex tasks than on simpler ones. This may be attributed to the fact that tasks without time windows often include a lot of padding information, which may introduce some noise during model training. Moreover, RGCR introduces no additional parameters compared to the "Concat Embeddings" setting, yet still reduces the average gap by 0.041%. This indicates that the gains arise from improved constraint prioritization rather than model capacity.

1055  
1056  
1057  
1058  
1059  
1060  
1061

### C.3.3 DESIGN CHOICE OF ATTENTION STRATEGY IN TSNR

1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073

In TSNR, we adopt a cross-attention mechanism, where the node embedding serves as the query and the unified node-constraint embedding as the key and value. The following theoretical analysis and empirical results show that this approach reduces computational complexity compared to the vanilla Transformer, which applies self-attention on the unified embedding. Specifically, the unified embedding has dimension  $(N + (N + 1)) \times D$ , while the context and node embeddings have dimensions  $N \times D$  and  $(N + 1) \times D$ , respectively. Consequently, self-attention computes  $(2N + 1) \times (2N + 1)$  attention weights, whereas cross-attention computes only  $(N + 1) \times (2N + 1)$ . As shown in Table 11, cross-attention achieves comparable performance while reducing inference time. For example, across tasks with TW (i.e., the right half of Table 11), it narrows the gap from 0.727% to 0.689% and reduces inference time by 1s.

1074  
1075  
1076  
1077  
1078  
1079

### C.4 PERFORMANCE COMPARISON UNDER MATCHED INFERENCE TIME

To further validate that the effectiveness of  $CCL$  is not merely due to increased inference time, we introduce a heavy decoder variant of CaDA, denoted as CaDA-HD. This version deepens the original 1-layer Transformer decoder to 4 layers, resulting in an inference time that is comparable to  $CCL$ . Table 12 presents the performance comparison between CaDA-HD and  $CCL$  under similar inference budgets. These results indicate that  $CCL$  and CaDA-HD perform similarly on tasks without time windows (TW), with CaDA-HD sometimes showing slightly better results. In contrast, on tasks with TW,  $CCL$  consistently outperforms CaDA-HD by a significant margin. This suggests that  $CCL$ 's

Table 10: Ablation on  $CCL$  (variant w/o ReLD).

Methods	Obj. $\downarrow$	Gap $\downarrow$	Time $\downarrow$
$CCL^\dagger$	11.447	1.10%	6.5s
- ReLD ( $CCL$ )	11.464	1.28%	5.4s
- RGCR	11.477	1.40%	4.6s
- TSNR	11.518	1.76%	2.8s
- TSNR - RGCR	11.529	1.88%	2.0s

Table 11: Performance comparison under different attention mechanisms.

Tasks	Cross-Attention			Self-Attention			Tasks	Cross-Attention			Self-Attention			
	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓		Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	
CVRP	10.463	0.881%	6s	<b>10.461</b>	<b>0.867%</b>	7s	VRPTW	<b>16.177</b>	<b>0.907%</b>	7s	16.186	0.955%	8s	
OVRP	<b>6.610</b>	<b>1.566%</b>	6s	6.611	1.582%	7s	OVRPTW	<b>10.564</b>	<b>0.506%</b>	7s	10.568	0.540%	8s	
VRPB	9.875	1.921%	6s	<b>9.873</b>	<b>1.896%</b>	7s	VRPBTW	<b>18.419</b>	<b>0.678%</b>	7s	18.427	0.719%	8s	
VRPL	10.698	1.027%	6s	<b>10.694</b>	<b>0.993%</b>	7s	VRPLTW	<b>16.556</b>	<b>1.192%</b>	7s	16.564	1.246%	8s	
OVRPB	6.992	1.344%	6s	<b>6.992</b>	<b>1.337%</b>	7s	OVRPBWTW	<b>11.718</b>	<b>0.416%</b>	7s	11.721	0.444%	8s	
OVRPL	<b>6.610</b>	<b>1.569%</b>	6s	6.611	1.582%	7s	OVRPLTW	<b>10.564</b>	<b>0.501%</b>	7s	10.565	0.517%	8s	
VRPBL	<b>10.440</b>	<b>2.450%</b>	6s	10.445	2.489%	7s	VRPBBLTW	<b>18.758</b>	<b>0.899%</b>	7s	18.769	0.952%	8s	
OVRPBL	6.992	1.335%	6s	<b>6.992</b>	<b>1.330%</b>	7s	OVRPBBLTW	<b>11.718</b>	<b>0.414%</b>	7s	11.721	0.446%	8s	
Avg.		8.585	1.512%	6s	<b>8.585</b>	<b>1.510%</b>	7s	Avg.	<b>14.309</b>	<b>0.689%</b>	7s	14.315	0.727%	8s

Table 12: Performance comparison under matched inference time.

Tasks	CaDA-HD			CCL			Tasks	CaDA-HD			CCL				
	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓		Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓		
CVRP	<b>10.468</b>	<b>0.926%</b>	6s	10.473	0.977%	5s	VRPTW	16.293	1.631%	5s	<b>16.190</b>	<b>0.979%</b>	5s		
OVRP	<b>6.635</b>	<b>1.937%</b>	5s	6.636	1.957%	5s	OVRPTW	10.615	0.986%	5s	<b>10.569</b>	<b>0.543%</b>	6s		
VRPB	<b>9.908</b>	<b>2.267%</b>	5s	9.916	2.352%	5s	VRPBTW	18.529	1.280%	5s	<b>18.430</b>	<b>0.738%</b>	6s		
VRPL	<b>10.704</b>	<b>1.091%</b>	5s	10.710	1.145%	5s	VRPLTW	16.676	1.930%	5s	<b>16.579</b>	<b>1.333%</b>	6s		
OVRPB	7.019	1.725%	5s	<b>7.008</b>	<b>1.568%</b>	5s	OVRPBWTW	11.772	0.874%	6s	<b>11.721</b>	<b>0.436%</b>	6s		
OVRPL	<b>6.633</b>	<b>1.908%</b>	5s	6.637	1.968%	5s	OVRPLTW	10.615	0.986%	5s	<b>10.569</b>	<b>0.546%</b>	6s		
VRPBL	<b>10.480</b>	<b>2.842%</b>	5s	10.484	2.883%	5s	VRPBBLTW	18.872	1.498%	5s	<b>18.773</b>	<b>0.976%</b>	6s		
OVRPBL	7.020	1.721%	5s	<b>7.009</b>	<b>1.569%</b>	5s	OVRPBBLTW	11.771	0.864%	6s	<b>11.721</b>	<b>0.442%</b>	6s		
Avg.		<b>8.608</b>	<b>1.802%</b>	5s	8.609	1.802%	5s	Avg.		14.393	1.256%	5s	<b>14.319</b>	<b>0.749%</b>	6s

design is particularly effective in handling temporally constrained problems, and its advantage is not merely a result of longer inference time.

### C.5 CONVERGENCE ANALYSIS

Fig. 5 shows the training loss of CCL and CaDA. CCL achieves faster convergence in the early epochs and reaches a lower final loss compared to CaDA. For instance, at epoch 50, the loss of CCL is 0.0129, while CaDA is 0.0198. By the end of training, CCL attains 0.0090 versus 0.0129 for CaDA. These results indicate that CCL contributes to more efficient and effective training, yielding both faster convergence and improved final performance.

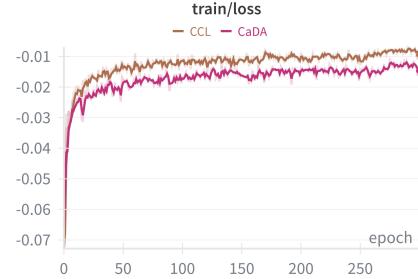


Figure 5: Training loss convergence.

### C.6 GENERALIZATION OF CCL ON ADDITIONAL SCENARIOS

Same as RouteFinder (Berto et al., 2024a;b), CaDA (Li et al., 2025), MTPOMO (Liu et al., 2024), and MvMoE (Zhou et al., 2024a), our work also focuses on solving routing problems solely. Moreover, the design of CCL can be useful for other decision-making problems. We conduct a preliminary experiment on the Flexible Flow Shop Problem (FFSP). When assigning an operation to a machine, we incorporated TSNR to allow operation embeddings to integrate information from the current machine, thereby updating the operation's state. Results in Fig. 6 (a) show that the method converges, but training can become slightly unstable from the middle to late stages. This suggests that certain modifications to TSNR may be needed for the best performance, for example, to filter out irrelevant information in the machine embeddings that does not contribute to subsequent decision-making.

We also conduct an experiment using graph-structured inputs instead of coordinates. Specifically, each node's coordinates were replaced with a vector of distances to all other nodes, which is then concatenated with demand and other attributes to form the node inputs. We retrain CCL across 16 tasks, each with 50 customers. During the 300 training epochs, we report the training loss and the validation average objective length across 128 CVRP instances (also with 50 customers). Fig. 6 (b) shows that both the training loss and the validation scores of CCL converge quickly within the first 50 epochs, suggesting the potential of CCL for graph-structured VRPs. To further investigate the effectiveness of RGCR and TSNR, we apply this setting to retrain the corresponding baseline model (*i.e.*, the version without RGCR and TSNR). Results in Fig. 6(c) show that, except for CVRP, CCL

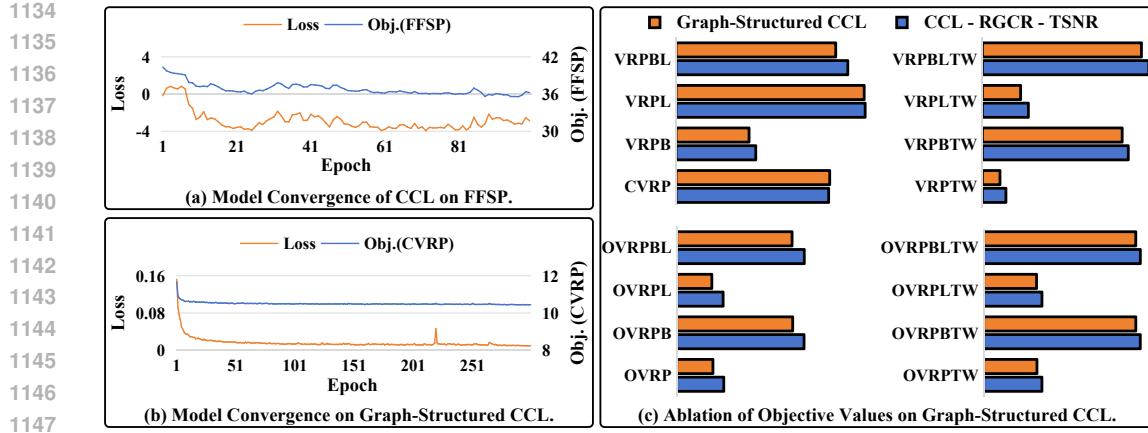


Figure 6: Generalizing CCL on the flexible flow shop problem (FFSP) and graph-structured input.

Table 13: Per-instance results on small-scale real-world instances. In CVRP, "X-n\*\*\*" denotes the customer number of each instance, while each VRPTW instance has 100 customers.

Instances	Opt.	CVRP		RF-TE		CaDA		CCL		CCL-Ens		VRPTW		RF-TE		CaDA		CCL		
		Obj. ↓	Gap ↓	Obj. ↓	Gap ↓	Obj. ↓	Gap ↓	Obj. ↓	Gap ↓	Obj. ↓	Gap ↓	Instances	Opt.	Obj. ↓	Gap ↓	Obj. ↓	Gap ↓	Obj. ↓	Gap ↓	
X-n101-k25	27591	29087	5.422%	28765	4.255%	28765	4.255%	28727	4.117%	R101	1638	X-n104-k25	27591	1604 <b>-2.058%</b>	1612	-1.569%	1619	-1.142%		
X-n106-k14	26362	27162	3.035%	27069	2.682%	26966	2.291%	26864	1.904%	R102	1467	X-n107-k14	26362	1567 <b>6.846%</b>	1572	7.187%	1553 <b>5.891%</b>			
X-n110-k13	14971	15314	2.291%	15425	3.033%	15386	2.772%	15185	1.429%	R103	1209	X-n111-k13	14971	1493 <b>23.521%</b>	1458	20.625%	1444 <b>19.467%</b>			
X-n115-k10	12747	13338	4.636%	<b>13143</b> <b>3.107%</b>	13334	4.605%	13162	3.256%	R104	972	X-n116-k10	12747	1315 <b>35.358%</b>	1325	36.387%	1308 <b>34.637%</b>				
X-n120-k6	13332	13765	3.248%	13741	3.068%	13852	3.900%	13677	2.588%	R105	1355	X-n121-k6	13332	1456 <b>7.430%</b>	1463	7.947%	1443 <b>6.471%</b>			
X-n125-k30	55539	58525	5.376%	57943	4.328%	57671	3.839%	57442	3.426%	R106	1235	X-n126-k30	55539	1455 <b>17.852%</b>	1420	15.017%	1405 <b>13.802%</b>			
X-n129-k10	28940	29598	2.274%	29517	1.994%	29599	2.277%	<b>29458</b> <b>1.790%</b>	1.790%	R107	1065	X-n130-k10	28940	1388 <b>30.378%</b>	1378	29.438%	1327 <b>24.648%</b>			
X-n134-k13	10916	11585	6.129%	11468	5.057%	<b>11464</b> <b>5.020%</b>	11464	5.020%	R108	932	X-n135-k13	10916	1310 <b>40.543%</b>	1285	37.861%	1266 <b>35.822%</b>				
X-n139-k10	13590	<b>13812</b> <b>1.634%</b>	13863	2.009%	13902	2.296%	13852	1.928%	R109	1147	X-n140-k10	13590	1601 <b>39.594%</b>	1394	21.545%	1359 <b>18.493%</b>				
X-n143-k7	15700	16257	3.548%	16233	3.395%	<b>15985</b> <b>1.815%</b>	15985	1.815%	R110	1068	X-n144-k7	15700	1527 <b>42.978%</b>	1384	29.588%	1282 <b>20.037%</b>				
X-n148-k40	43448	45036	3.655%	45395	4.481%	45324	4.318%	<b>4995</b> <b>3.464%</b>	3.464%	R111	1049	X-n149-k40	43448	1473 <b>40.460%</b>	1424	35.787%	1366 <b>30.257%</b>			
X-n153-k26	21220	23478	10.641%	<b>22815</b> <b>7.516%</b>	23245	9.543%	23172	9.199%	R112	949	X-n154-k26	21220	1357 <b>43.053%</b>	1278	34.725%	1210 <b>27.556%</b>				
X-n157-k13	16876	17339	2.744%	17225	2.068%	17184	1.825%	<b>17131</b> <b>1.511%</b>	1.511%	R101	1620	X-n158-k13	16876	1666 <b>2.852%</b>	1663	2.667%	1661 <b>2.544%</b>			
X-n162-k11	14138	14664	3.720%	<b>14584</b> <b>3.155%</b>	14702	3.989%	14671	3.777%	R102	1457	X-n163-k11	14138	1731 <b>18.773%</b>	1717	17.813%	1646 <b>12.941%</b>				
X-n167-k20	20557	21435	4.271%	21305	3.639%	20987	2.092%	<b>20934</b> <b>1.834%</b>	1.834%	R103	1258	X-n168-k20	20557	1760 <b>39.905%</b>	1656	31.638%	1624 <b>29.094%</b>			
X-n172-k51	45607	48129	5.530%	<b>47727</b> <b>4.648%</b>	48252	5.800%	47836	4.887%	R104	1132	X-n173-k51	45607	1610 <b>42.188%</b>	<b>1497</b> <b>32.209%</b>	1524	34.593%				
X-n176-k26	47812	51400	7.504%	52177	9.130%	51485	7.682%	<b>51164</b> <b>7.011%</b>	7.011%	R105	1514	X-n177-k26	47812	1867 <b>23.340%</b>	1755	15.941%	1751 <b>15.677%</b>			
X-n181-k23	25569	26097	2.065%	26228	2.577%	26180	2.390%	<b>26075</b> <b>1.979%</b>	1.979%	R106	1373	X-n182-k23	25569	1664 <b>21.221%</b>	1634	19.035%	1621 <b>18.088%</b>			
X-n186-k16	24145	25140	4.121%	<b>24909</b> <b>3.164%</b>	25046	3.732%	25002	3.549%	R107	1208	X-n187-k16	24145	1683 <b>39.344%</b>	1601 <b>32.555%</b>	1494 <b>24.027%</b>					
X-n190-k8	16980	17892	5.371%	17726	4.393%	<b>17547</b> <b>3.339%</b>	17547	3.339%	R108	1114	X-n191-k8	16980	1768 <b>58.679%</b>	1564 <b>40.370%</b>	1504 <b>34.985%</b>					
X-n195-k51	44225	47390	7.157%	46585	5.336%	46621	5.418%	<b>46121</b> <b>4.287%</b>	4.287%	R109	1262	X-n196-k51	44225	1577 <b>24.980%</b>	1606	27.278%	1533 <b>21.493%</b>			
X-n200-k36	58578	61199	4.474%	<b>61048</b> <b>4.217%</b>	61388	4.797%	61388	4.797%	R110	1092	X-n201-k36	58578	1553 <b>42.177%</b>	1480	35.494%	1433 <b>31.191%</b>				
X-n209-k10	30656	<b>31876</b> <b>3.980%</b>	32005	4.400%	32334	5.474%	32216	5.089%	R111	924	X-n210-k10	30656	1465 <b>58.601%</b>	1490	61.308%	1439 <b>55.787%</b>				
X-n228-k23	25742	28798	11.872%	28328	10.046%	<b>27641</b> <b>7.377%</b>	27641	7.377%	R112	784	X-n229-k23	25742	1372 <b>75.112%</b>	1278	63.114%	1225 <b>56.350%</b>				
X-n237-k14	27042	<b>29595</b> <b>9.441%</b>	29830	10.310%	29816	10.258%	29816	10.258%	R113	1051	X-n238-k14	27042	1573 <b>49.653%</b>	<b>1447</b> <b>37.665%</b>	1456 <b>38.522%</b>					
X-n247-k50	37274	40639	9.028%	<b>40456</b> <b>8.537%</b>	41266	10.710%	41266	10.710%	R114	963	X-n248-k50	37274	1694 <b>75.927%</b>	1503 <b>56.091%</b>	1433 <b>48.821%</b>					
X-n251-k26	38684	40399	4.433%	<b>40360</b> <b>4.333%</b>	40725	5.276%	40505	4.707%	R115	776	X-n252-k26	38684	1465 <b>88.764%</b>	1433 <b>84.641%</b>	1335 <b>72.014%</b>					
Avg. Gap		5.096%	4.625%	4.707%	4.261%	Avg. Gap		36.573%	30.828%	27.114%		# Best (Best/Total)		3/27	8/27	4/27	12/27	1/27	2/27	24/27

consistently reduces the average length compared to the baseline. This demonstrates that CCL is a plug-and-play strategy, which can be integrated into VRP solvers with various input structures.

### C.7 EVALUATION ON REAL-WORLD BENCHMARK INSTANCES

Table 13 and Table 14 present per-instance results on small and large-scale real-world benchmarks, respectively. The small-scale set consists of 27 CVRP and 27 VRPTW instances, while the large-scale set contains 60 VRPTW instances. Following (Zhou et al., 2024a), the model is trained on 16 synthetic tasks with  $N = 100$  and directly applied to all instances in a zero-shot manner. For CVRP, we set the test-time update probability  $P_{ts}$  to 0.1, denoted as CCL. We also design an ensemble variant, CCL-Ens, which applies the trained model with four update probabilities  $P_{ts} \in \{0.1, 0.15, 0.25, 0.3\}$  and selects the best solution. For small-scale VRPTW,  $P_{ts}$  is fixed at 0.25.

On the CVRP benchmark, CCL yields a slightly higher average gap compared to CaDA. However, its ensemble variant CCL-Ens achieves the best overall performance, demonstrating the benefit of test-time adaptation via varying update probabilities. In total, CCL and CCL-Ens together outperform baselines on 16 out of 27 instances. On the VRPTW benchmark, CCL attains the lowest average gap and surpasses baselines on 24 out of 27 small-scale instances and 35 out of 60 large-scale instances.

1188 These results indicate that our method can be effectively deployed in real-world settings, especially  
 1189 on complex constraints such as time windows.  
 1190

1192 Table 14: Per-instance results on large-scale real-world instances. The customer number is 600.  
 1193

Instances	Opt.	VRPTW			RF-TE			CaDA			CCL			
		Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	Obj. ↓	Gap ↓	Time ↓	
C1-6-1	14077	17537	24.583%	1.4s	17355	23.290%	1.5s	16418	<b>16.633%</b>	2.5s				
C1-6-10	13618	35201	158.498%	1.2s	24365	78.924%	1.3s	17861	<b>31.162%</b>	1.5s				
C1-6-2	13948	20505	<b>47.007%</b>	1.1s	21571	54.650%	1.1s	20980	50.413%	1.6s				
C1-6-3	13757	22648	<b>64.635%</b>	1.1s	23142	68.226%	1.1s	24090	75.117%	1.6s				
C1-6-4	13539	22743	67.986%	1.1s	22601	<b>66.937%</b>	1.1s	23787	75.698%	1.7s				
C1-6-5	14067	18265	29.845%	1.2s	17643	25.423%	1.1s	16368	<b>16.359%</b>	1.6s				
C1-6-6	14071	22405	59.229%	3.3s	19998	42.123%	3.3s	19481	<b>38.449%</b>	4.9s				
C1-6-7	14067	26369	87.456%	1.3s	23920	70.046%	1.2s	16848	<b>19.771%</b>	1.6s				
C1-6-8	13991	26618	90.248%	1.2s	20504	46.549%	1.1s	16844	<b>20.390%</b>	1.6s				
C1-6-9	13665	60014	339.196%	1.5s	36753	168.967%	1.3s	18274	<b>33.733%</b>	1.6s				
C2-6-1	7752	15193	95.983%	3.1s	12018	<b>55.027%</b>	3.1s	12410	60.084%	4.9s				
C2-6-10	7124	29321	311.586%	1.3s	17609	147.182%	1.1s	12161	<b>70.707%</b>	1.6s				
C2-6-2	7472	14789	97.939%	3.1s	14420	93.000%	3.1s	13642	<b>82.587%</b>	4.7s				
C2-6-3	7215	15033	<b>108.358%</b>	3.1s	16259	125.350%	3.1s	18351	154.345%	4.7s				
C2-6-4	6877	15039	<b>118.685%</b>	3.1s	16236	136.091%	3.1s	18952	175.585%	4.8s				
C2-6-5	7554	22865	202.695%	1.2s	13343	76.640%	1.1s	13040	<b>72.628%</b>	1.5s				
C2-6-6	7450	22171	197.605%	1.2s	13312	78.689%	1.1s	12032	<b>61.508%</b>	1.5s				
C2-6-7	7491	25219	236.644%	3.1s	14632	95.320%	3.1s	12908	<b>72.307%</b>	4.8s				
C2-6-8	7304	22619	209.692%	1.2s	13469	84.413%	1.1s	12266	<b>67.942%</b>	1.5s				
C2-6-9	7303	23663	224.009%	3.1s	15017	105.622%	3.1s	12569	<b>72.103%</b>	4.7s				
R1-6-1	21274	29154	37.039%	1.2s	25041	<b>17.706%</b>	1.1s	26556	24.827%	1.6s				
R1-6-10	17584	30508	73.502%	1.2s	26126	48.581%	1.1s	26121	<b>48.552%</b>	1.6s				
R1-6-2	18520	26017	<b>40.482%</b>	1.1s	26262	41.805%	1.1s	27011	45.849%	1.7s				
R1-6-3	16875	26105	<b>54.697%</b>	1.1s	26601	57.636%	1.1s	28093	66.478%	1.6s				
R1-6-4	15721	24450	<b>55.526%</b>	1.1s	24521	55.978%	1.1s	26939	71.359%	1.9s				
R1-6-5	19295	31715	64.370%	1.2s	24975	<b>29.438%</b>	1.1s	25367	31.470%	1.8s				
R1-6-6	17764	25692	44.632%	1.1s	25559	<b>43.883%</b>	1.1s	26890	51.376%	1.6s				
R1-6-7	16496	25749	<b>56.090%</b>	1.1s	26401	60.043%	1.1s	26693	61.813%	1.6s				
R1-6-8	15584	23857	<b>53.084%</b>	1.1s	24533	57.421%	1.1s	24796	59.109%	1.5s				
R1-6-9	18474	30700	66.179%	1.2s	25067	<b>35.687%</b>	1.1s	25931	40.364%	1.7s				
R2-6-1	15145	31072	105.159%	1.2s	22482	48.442%	1.1s	21855	<b>44.302%</b>	1.6s				
R2-6-10	11837	35862	202.965%	1.2s	23395	97.643%	1.1s	19894	<b>68.066%</b>	1.6s				
R2-6-2	12976	22676	74.749%	1.0s	21124	<b>62.789%</b>	1.0s	24214	86.602%	1.6s				
R2-6-3	10455	20072	91.979%	1.0s	19274	<b>84.347%</b>	1.0s	24258	132.016%	1.8s				
R2-6-4	7915	16925	113.848%	1.0s	16589	<b>109.603%</b>	1.0s	22399	183.012%	1.7s				
R2-6-5	13790	33895	145.790%	1.2s	22311	61.789%	1.1s	21581	<b>56.495%</b>	1.7s				
R2-6-6	11848	22695	91.555%	1.1s	20914	<b>76.522%</b>	1.0s	22125	86.744%	1.5s				
R2-6-7	9770	19723	101.867%	1.0s	18900	<b>93.443%</b>	1.0s	23231	137.772%	1.8s				
R2-6-8	7512	16596	<b>120.918%</b>	1.0s	16716	122.515%	1.0s	23249	209.479%	2.0s				
R2-6-9	12737	35883	181.727%	1.3s	22963	80.289%	1.1s	20863	<b>63.801%</b>	1.7s				
RC1-6-1	16944	44173	160.697%	1.3s	28659	69.138%	1.2s	22506	<b>32.824%</b>	1.7s				
RC1-6-10	15651	47967	206.473%	1.3s	33429	113.586%	1.2s	23311	<b>48.940%</b>	1.6s				
RC1-6-2	15891	24480	54.053%	1.1s	25282	59.100%	1.1s	23526	<b>48.050%</b>	1.6s				
RC1-6-3	15181	23667	55.896%	1.1s	23640	<b>55.718%</b>	1.1s	23809	56.831%	1.6s				
RC1-6-4	14753	23076	56.414%	1.1s	22451	52.177%	1.1s	22178	<b>50.327%</b>	1.6s				
RC1-6-5	16536	45720	176.483%	1.3s	29036	75.589%	1.2s	22505	<b>36.095%</b>	1.7s				
RC1-6-6	16473	47520	188.467%	1.3s	30741	86.611%	1.2s	22586	<b>37.107%</b>	1.7s				
RC1-6-7	16055	45359	182.517%	1.3s	31323	95.094%	1.2s	21936	<b>36.628%</b>	1.6s				
RC1-6-8	15892	42548	167.736%	1.3s	32086	101.903%	1.2s	22792	<b>43.420%</b>	1.6s				
RC1-6-9	15804	47394	199.896%	1.3s	33968	114.940%	1.2s	24490	<b>54.966%</b>	1.7s				
RC2-6-1	11966	46194	286.041%	1.3s	26852	124.401%	1.2s	20004	<b>67.172%</b>	1.6s				
RC2-6-10	8973	44372	394.489%	1.3s	28880	221.844%	1.2s	17107	<b>90.643%</b>	1.6s				
RC2-6-2	10337	21343	106.474%	1.1s	20623	<b>99.509%</b>	1.1s	21034	103.485%	1.6s				
RC2-6-3	8895	17942	101.711%	1.0s	17270	<b>94.156%</b>	1.0s	22858	156.979%	1.6s				
RC2-6-4	6968	14864	113.333%	1.0s	14459	<b>107.521%</b>	1.0s	18521	165.820%	1.6s				
RC2-6-5	11081	45546	311.039%	1.3s	27674	149.750%	1.2s	19270	<b>73.906%</b>	1.6s				
RC2-6-6	10831	46487	329.223%	1.3s	27920	157.790%	1.2s	18374	<b>69.651%</b>	1.6s				
RC2-6-7	10289	46929	356.091%	1.3s	28173	173.806%	1.2s	18364	<b>78.475%</b>	1.6s				
RC2-6-8	9779	45416	364.424%	1.3s	29578	202.464%	1.2s	18548	<b>89.672%</b>	1.7s				
RC2-6-9	9436	44922	376.070%	1.3s	29079	208.171%	1.2s	16951	<b>79.642%</b>	1.7s				
Avg.		12694	29558	145.593%	1.4s	22917	88.188%	1.4s	20634	<b>70.961%</b>	2.0s			
# Best (Best/Total)				10/60			15/60			35/60				

1242 C.8 THE USE OF LARGE LANGUAGE MODELS (LLMs)  
12431244 We employed LLMs for polishing the paper and assisting with simple coding tasks. For writing  
1245 refinement, we first drafted the original statements in English and then used LLMs to improve fluency.  
1246 For coding, most of the implementations and modules were done based on Routefinder (Berto et al.,  
1247 2024b), while the design of our proposed CCL was completed independently. LLMs were only used  
1248 for basic Python coding tasks, such as aligning feature dimensions.

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295