Synergistic Paradigms in Neural Combinatorial Optimization: A Comprehensive Synthesis of Hybrid Imitation and Reinforcement Learning for the Capacitated Vehicle Routing Problem

1. Introduction: The Convergence of Operations Research and Machine Learning

The Capacitated Vehicle Routing Problem (CVRP) remains one of the most enduring and computationally stubborn challenges in the field of Operations Research (OR) and Combinatorial Optimization (CO). Defined by the imperative to service a geographically dispersed set of customers with a fleet of capacity-constrained vehicles while minimizing total travel cost, the CVRP is an NP-hard problem that underpins vast sectors of the global economy, from last-mile logistics to ride-hailing and supply chain management.[1] For decades, the resolution of these problems has relied on two distinct methodological pillars: exact methods, such as Branch-and-Cut-and-Price, which guarantee optimality but scale poorly (often intractable beyond a few hundred nodes), and metaheuristics, such as the Lin-Kernighan-Helsgaun (LKH) algorithm or Hybrid Genetic Search (HGS), which trade optimality for speed and scalability but rely heavily on handcrafted, domain-specific rules.[3]

In recent years, a third pillar has emerged: Neural Combinatorial Optimization (NCO). This paradigm seeks to replace or augment manual heuristic design with data-driven policies learned by deep neural networks. The central premise is that a neural network, trained on a distribution of problem instances, can learn a mapping from problem specifications (node coordinates, demands) to high-quality solutions, potentially discovering heuristics that elude human intuition.[5] Within NCO, two primary learning modalities have dominated the landscape: Reinforcement Learning (RL) and Imitation Learning (IL).

RL, typically employing policy gradient methods like REINFORCE or Proximal Policy Optimization (PPO), treats the VRP as a sequential decision-making process. The agent builds a tour node-by-node, receiving a reward (negative tour length) upon completion. The allure of RL lies in its unsupervised nature; it does not require labeled optimal solutions, theoretically allowing the agent to surpass human-designed heuristics by exploring the solution space via

trial and error.[7] However, RL in combinatorial spaces faces severe challenges: sparse rewards, high variance in gradient estimation, and the "needle in a haystack" problem—as the number of nodes N increases, the state space grows factorially ($N!$), making random exploration increasingly futile.[9]

Conversely, IL leverages the existence of powerful solvers (experts) to train neural networks via supervised learning. By minimizing the divergence between the learner's policy and the expert's demonstrations, IL provides a dense, stable training signal that accelerates convergence.[10] Yet, IL is fundamentally limited by the performance of the expert (the teacher) and suffers from distribution shift (covariate shift), where the learner encounters states during inference that were not present in the training distribution, leading to compounding errors.[11]

This report posits that the state-of-the-art in NCO is no longer defined by the binary choice between RL and IL, but by **hybrid architectures** that synergize these paradigms. By integrating IL to "warm-start" or guide exploration and RL to refine policies and generalize beyond the expert, researchers are developing solvers that offer the scalability of heuristics with the adaptability of neural networks. This document provides an exhaustive analysis of these hybrid methodologies, detailing theoretical formulations, algorithmic implementations, and the emerging frontier of Large Language Model (LLM)-driven heuristic generation.

2. Theoretical Foundations and Mathematical Formulation

To construct a robust hybrid solver, one must first rigorously define the CVRP within a framework compatible with both sequential decision-making (MDPs) and supervised learning.

2.1 The CVRP Graph Formulation

We formally define a CVRP instance on a complete undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The node set $\mathcal{V} = \{0, 1, \dots, N\}$ consists of a central depot (node 0) and N customer nodes. The edge set $\mathcal{E}$ connects all pairs of nodes.

- **Features:** Each node $i \in \mathcal{V}$ is associated with a feature vector x_i, typically comprising 2D Euclidean coordinates.
- **Demands:** Each customer $i \in \{1, \dots, N\}$ has a discrete demand $\delta_i > 0$. The depot has demand $\delta_0 = 0$.
- **Constraints:** A homogenous fleet of vehicles with capacity Q is available.
- **Objective:** Find a set of routes $\Pi = \{\pi_1, \dots, \pi_K\}$ such that each customer is visited exactly once, every route starts and ends at the depot, the sum of demands in any route π_k does not exceed Q, and the total Euclidean distance is minimized.[1]

$$\min \sum_{k=1}^K \sum_{(i,j) \in \pi_k} \| x_i - x_j \|_2$$

2.2 The Markov Decision Process (MDP) for Constructive Heuristics

NCO typically solves the CVRP constructively, adding one node at a time to a partial tour. This is modeled as an MDP tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$.[2]

1. State Space ($\mathcal{S}$):
The state s_t at time step t must capture the static graph structure and the dynamic context of the tour.

$$s_t = \{ \mathcal{G}, \pi_{1:t-1}, q_t, l_t \}$$

Where:

- $\mathcal{G}$ represents the static node embeddings (derived from a Graph Neural Network or Transformer encoder).
- $\pi_{1:t-1}$ is the sequence of nodes visited so far.
- q_t is the remaining capacity of the current vehicle.
- l_t is the current location (the last visited node π_{t-1}).

2. Action Space ($\mathcal{A}_t$):
The action a_t corresponds to selecting the next node to visit. Crucially, the action space is dynamic and constrained. We define a valid action mask $\mathcal{M}_t \in \{0, 1\}^{N+1}$:

$$\mathcal{M}_{t, j} = \begin{cases} 1 & \text{if } j \notin \pi_{1:t-1} \text{ and } \delta_j \le q_t \quad \text{(Visit Unserved Customer)} \\ 1 & \text{if } j=0 \text{ and } l_t \neq 0 \quad \text{(Return to Depot)} \\ 0 & \text{otherwise} \end{cases}$$

The depot (node 0) is available for visitation to replenish capacity, but typically only if the vehicle is not already at the depot. Masking prevents invalid solutions during training and inference.14
3. Transitions (T):
The transition is deterministic. Upon selecting a_t:

- If $a_t \neq 0$: $q_{t+1} = q_t - \delta_{a_t}$. The node is marked as visited.
- If $a_t = 0$: $q_{t+1} = Q$. A new route begins.

4. Reward (R):
The standard objective is to minimize total tour length $L(\pi)$. In RL, the reward is often sparse, given only at the terminal state s_T:

$$R(s_T) = -L(\pi)$$

Intermediate rewards are zero. Some formulations use "shaped" rewards, providing the negative edge weight $-\|x_{t-1} - x_t\|_2$ at each step, but this can be myopic. The sparse terminal reward aligns the agent with the global objective but complicates credit assignment.7

2.3 Neural Architecture: The Attention Model (AM)

The dominant architecture for parameterizing the policy $\pi_\theta(a_t | s_t)$ is the Attention Model (AM), introduced by Kool et al. and refined in subsequent works like POMO and Sym-NCO.[8] This is an Encoder-Decoder architecture based on the Transformer.

Encoder:
The encoder maps the input node features X to high-dimensional embeddings $H = \{h_0, \dots, h_N\}$. It utilizes Multi-Head Self-Attention (MHA) to capture the geometric relationships between nodes.

$$h_i^{(l+1)} = \text{BN}(h_i^{(l)} + \text{MHA}(h_i^{(l)}, H^{(l)}, H^{(l)}))$$

This step allows the model to understand the global topology of the CVRP instance—identifying clusters, outliers, and the depot's centrality.
Decoder:
The decoder generates the probability distribution over the next node using a context vector c_t. The context typically aggregates the embedding of the graph ($\bar{h}$), the embedding of the current node (h_{l_t}), and the remaining capacity (q_t).

$$u_{t, j} = C \cdot \tanh \left(\frac{q^T k_j}{\sqrt{d_k}} \right)$$

where query q comes from the context and key k_j comes from node embedding h_j. Finally, the probability is computed via a masked softmax:

$$p_\theta(a_t = j | s_t) = \frac{e^{u_{t,j}} \cdot \mathcal{M}_{t,j}}{\sum_{k} e^{u_{t,k}} \cdot \mathcal{M}_{t,k}}$$
This architecture serves as the backbone for both the RL and IL components of the hybrid strategies discussed below.

3. Methodological Convergence: Hybrid Learning Strategies

The core thesis of this report is that neither pure RL nor pure IL is sufficient for state-of-the-art performance on large-scale CVRP. Pure RL struggles with the "cold start" problem—random initialization yields such poor policies that the probability of stumbling upon a high-quality solution (and thus a meaningful gradient) is negligible. Pure IL suffers from the

lack of generalization; it learns to copy the expert's local moves but fails to internalize the global optimization logic, leading to failure when the test distribution shifts (e.g., from 50 nodes to 1000 nodes).

Hybridization addresses these failures through three distinct mechanisms: **Warm-Starting**, **Self-Improvement (Self-Imitation)**, and **Hierarchical Delegation**.

3.1 Warm-Starting: The "Pretrain-Finetune" Paradigm

This strategy treats IL as a robust initialization method for RL. The process unfolds in two distinct phases.

Phase I: Imitation Learning (Pretraining)
We generate a dataset of random CVRP instances $\mathcal{D}_{train} = \{G_i\}_{i=1}^M$. For each instance, we run a high-performance heuristic (the "Teacher"), typically LKH-3 or HGS, to generate a ground-truth tour $\pi^*_i = (a^*_1, \dots, a^*_T)$.
The network is trained to minimize the Cross-Entropy loss between its predicted probabilities and the expert's actions:

$$\mathcal{L}_{IL}(\theta) = - \sum_{i=1}^M \sum_{t=1}^{T_i} \log p_\theta(a_{i,t}^* | s_{i,t})$$

This phase imparts the "grammar" of routing to the model. The network learns basic feasibility rules (don't exceed capacity, return to depot when full) and local geometric heuristics (visit nearest neighbors). It effectively pushes the model parameters θ into a region of the landscape where the policy produces valid, reasonable tours.17
Phase II: Reinforcement Learning (Fine-tuning)
Once the model has learned the basics, we switch the objective to reward maximization using PPO or REINFORCE.

$$\mathcal{L}_{RL}(\theta) = \mathbb{E}_{\pi \sim p_\theta} [(L(\pi) - b(s)) \nabla \log p_\theta(\pi)]$$

Here, the model is no longer bound by the expert. It can explore variations of the expert's strategy. Because the policy is warm-started, the initial generated tours are high-quality, providing meaningful baselines and gradients. This approach prevents the RL agent from wasting epochs learning basic constraints and allows it to focus on global optimization.17
Insight: This two-stage approach mirrors the "pretraining" paradigm in Natural Language Processing (LLMs). Just as LLMs are pretrained on text prediction before being fine-tuned on instructions, NCO models are pretrained on expert traces before being fine-tuned on the true objective function.

3.2 Imitation Improvement Learning (IIL)

While warm-starting is sequential, Imitation Improvement Learning (IIL) interleaves IL and RL during the training process. This method acknowledges that static datasets of expert trajectories are expensive to generate and may not cover the state space the RL agent explores.

In IIL, the "expert" is dynamic. The framework, often used in iterative improvement models, functions as follows:

1. **Generate:** The current policy π_θ generates a candidate solution π_{curr}.
2. **Improve:** A fast local search heuristic (e.g., 2-opt, node swap) improves π_{curr} to obtain π_{imp}.
3. Imitate: The model is updated to maximize the likelihood of generating π_{imp} (treating the improved solution as the target).

   $$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{CE}(\pi_{imp}, \pi_\theta)$$

This creates a **Self-Reinforcing Curriculum**. As the policy improves, the starting point for the local search improves, allowing the local search to find even better solutions, which in turn become better targets for the policy. This symbiotic loop allows the neural network to eventually internalize the logic of the local search operator, learning to generate solutions that are already local-optima.[3]

3.3 Self-Improvement: Sample-Based "Bootstrap" Learning

Recent state-of-the-art methods like **Gumbeldore** and **MACSIM** remove the external expert entirely, relying on the statistical power of sampling to drive improvement.[19]

The Hypothesis: Given a stochastic policy π_θ, if we sample K solutions $\{\pi_1, \dots, \pi_K\}$ for a single instance, the best solution $\pi_{best} = \arg\min L(\pi_k)$ is likely significantly better than the average solution.

The Algorithm (Self-Critical Training):

1. For instance G, sample K trajectories via beam search or ancestral sampling.
2. Identify π_{best}.
3. Treat π_{best} as the pseudo-label (expert).
4. Update θ using Behavior Cloning on π_{best}.

This method is mathematically equivalent to a Policy Gradient method where the reward is shaped relative to the batch beam. However, framing it as "Self-Imitation" allows for the use of supervised learning losses (Cross Entropy), which typically have lower variance and faster convergence than RL gradients.
Crucially, Gumbeldore integrates Stochastic Beam Search (SBS) to ensure diversity in the samples. Without diversity, the samples $\pi_1 \dots \pi_K$ would be identical, π_{best}

would not improve, and the model would collapse (mode collapse).19

3.4 Hierarchical Learning: The "Learning to Delegate" (L2D) Framework

Standard constructive models (like POMO) degrade in performance as N exceeds 100-200 nodes due to the "long horizon" problem in RL (error accumulation) and the quadratic complexity of Attention. L2D [3] proposes a hybrid hierarchical approach that scales to real-world sizes ($N=2000+$).

Concept:
Instead of solving the full problem end-to-end, L2D uses a neural network as a "Manager" and a heuristic as a "Worker".

1. **Subproblem Selection (Manager):** An RL/IL-trained network takes the full problem state and selects a subset of nodes (a subproblem) that appears promising for improvement (e.g., a region with crossing edges).
2. **Delegation (Worker):** A high-quality sub-solver (like LKH or a small-scale neural solver) re-optimizes just this subproblem.
3. **Update:** The new sub-routes are reintegrated into the global solution.

Why Hybrid?
The Manager needs intuition (spatial pattern recognition) to identify where the solution is weak—a task neural networks excel at. The Worker needs precision to rearrange nodes optimally—a task combinatorial heuristics excel at for small N. This division of labor allows L2D to scale massively while retaining the learning benefits of NCO.21

4. Deep Dive: Key Algorithms and Architectures

In this section, we analyze the specific mechanisms of the most influential hybrid algorithms: POMO, Sym-NCO, and the emerging class of LLM-based optimizers.

4.1 POMO: Exploiting Symmetry for Self-Imitation

Policy Optimization with Multiple Optima (POMO) 8 is technically an RL method, but its mechanism relies on a form of internal imitation.
The CVRP solution space has inherent symmetries. The optimal tour is invariant to the choice of the first customer visited. Standard RL forces the model to learn one specific permutation. POMO forces the model to generate N trajectories in parallel, each starting at a different node $i \in \{1, \dots, N\}$.

- **The Baseline:** The baseline for the REINFORCE gradient is the average reward of these N trajectories: $b(s) = \frac{1}{N} \sum_{i=1}^N L(\pi_i)$.
- The Gradient:

  $$\nabla J = \frac{1}{N} \sum_{i=1}^N (L(\pi_i) - \bar{L}) \nabla \log p_\theta(\pi_i)$$

Trajectories better than the average are reinforced (positive gradient); those worse are discouraged. This acts as a competition where the model "imitates" its own best rollouts.
- **Sym-NCO** extends this by adding rotational and reflectional symmetries (data augmentation) to the batch, further stabilizing the "self-imitation" signal.[22]

4.2 RFTHGS: LLMs as Algorithm Generators

A radical shift in 2024-2025 is the move from "learning to route" to "learning to write routing algorithms".23
RFTHGS (Reinforcement Learning Framework for Fine-Tuning HGS) uses a small LLM (e.g., 7B parameter code model) to generate Python/C++ code for the crossover operator in a Genetic Algorithm.
Hybrid Loop:

1. **Action:** LLM generates code snippet C.
2. **Environment:** Compile C and run it inside the HGS solver on CVRP instances.
3. **Reward:**
   - $R_{compile}$: +1 if code compiles.
   - R_{run}: +1 if code runs without error.
   - $R_{quality}$: Proportional to the CVRP cost reduction compared to the baseline operator.
4. **Update:** PPO update on the LLM weights.

This represents the ultimate hybrid: the Neural Network (LLM) handles the creative search in the space of *algorithms*, while the Heuristic Solver handles the execution in the space of *routes*. The results show that these learned operators generalize better than human-designed ones.[23]

5. Implementation Guide: Code and Algorithms

This section provides concrete implementation details for a Hybrid RL-IL solver using PyTorch. We focus on a simplified version of the "Warm-Start" + "POMO" architecture.

5.1 The Neural Architecture (Encoder-Decoder)

We use a standard Attention Model. The code below illustrates the AttentionModel class structure required for CVRP.

Python

```
import torch
```

```python
import torch.nn as nn
import torch.nn.functional as F
from torch.distributions import Categorical

class AttentionModel(nn.Module):
    def __init__(self, embedding_dim=128, n_heads=8, n_layers=3):
        super(AttentionModel, self).__init__()
        self.embedding_dim = embedding_dim

        # 1. Initial Embedding
        # Depots and Customers have different input features (coords + demand)
        self.init_embed_depot = nn.Linear(2, embedding_dim)
        self.init_embed_cust = nn.Linear(3, embedding_dim) # x, y, demand

        # 2. Encoder (Transformer)
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=embedding_dim, nhead=n_heads, dim_feedforward=512
        )
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=n_layers)

        # 3. Decoder (Pointer Network with Context)
        self.project_fixed_context = nn.Linear(embedding_dim, embedding_dim)
        self.project_step_context = nn.Linear(2 * embedding_dim + 1, embedding_dim)
        self.project_out = nn.Linear(embedding_dim, embedding_dim)

    def forward(self, input_data, decode_type="sampling"):
        """
        input_data: (batch_size, num_nodes, features)
        """
        # --- ENCODING STEP ---
        # Separate depot (node 0) and customers
        depot = input_data[:, 0, :2]
        customers = input_data[:, 1:, :]

        emb_depot = self.init_embed_depot(depot)
        emb_cust = self.init_embed_cust(customers)
        embeddings = torch.cat([emb_depot.unsqueeze(1), emb_cust], dim=1) # (B, N, D)

        # Pass through Transformer
        # (Transpose for PyTorch Transformer: Seq_Len, Batch, Dim)
        embeddings = self.encoder(embeddings.transpose(0, 1)).transpose(0, 1)

        # Graph Embedding (Mean pooling)
```

```
        graph_embedding = embeddings.mean(dim=1)

        # --- DECODING STEP (Autoregressive) ---
        outputs =
        log_probs =

        # Initial State
        batch_size, num_nodes, _ = embeddings.size()
        mask = torch.zeros(batch_size, num_nodes, dtype=torch.bool).to(input_data.device)

        # Start at depot
        prev_node = torch.zeros(batch_size, dtype=torch.long).to(input_data.device)
        current_capacity = torch.ones(batch_size).to(input_data.device) # Normalized capacity =
1.0

        # Loop for max steps (usually > N due to depot returns)
        for _ in range(num_nodes * 2):
            # Context: Graph Emb + Prev Node Emb + Remaining Capacity
            context = torch.cat([
                graph_embedding,
                embeddings[torch.arange(batch_size), prev_node],
                current_capacity.unsqueeze(1)
            ], dim=1)

            # Query-Key-Value mechanism for Pointer
            # (Simplified computation for brevity)
            query = self.project_step_context(context).unsqueeze(2) # (B, D, 1)
            keys = self.project_out(embeddings).transpose(1, 2)     # (B, D, N)
            scores = torch.bmm(keys.transpose(1, 2), query).squeeze(2) # (B, N)

            # Application of MASK (Capacity Constraints)
            # Mask nodes that are visited OR demand > capacity
            # Note: Masking logic must update dynamically.
            # Here we assume 'mask' is updated outside this snippet for clarity.
            scores = scores.masked_fill(mask, float('-inf'))

            # Probability Distribution
            probs = F.softmax(scores, dim=-1)

            # Select Action
            if decode_type == "greedy":
                selected = probs.argmax(dim=1)
            else:
```

```
            m = Categorical(probs)
            selected = m.sample()
            log_probs.append(m.log_prob(selected))

        outputs.append(selected)
        prev_node = selected

        # Update Capacity and Mask (Pseudocode logic)
        # if selected!= 0 (depot):
        #    current_capacity -= demand[selected]
        #    mask[selected] = True
        # else:
        #    current_capacity = 1.0

        # Break if all customers visited

    return torch.stack(outputs, 1), torch.stack(log_probs, 1)
```

5.2 Hybrid Loss Implementation

The critical innovation is the loss function that supports both phases.

Python

```
class HybridTrainer:
    def __init__(self, model, optimizer):
        self.model = model
        self.optimizer = optimizer

    def train_step_imitation(self, batch_data, expert_tours):
        """
        Phase 1: Imitation Learning (Behavior Cloning)
        expert_tours: (Batch, Seq_Len) indices of optimal tour
        """
        self.optimizer.zero_grad()

        # Forward pass forcing the expert actions (Teacher Forcing)
        # Note: In production, we need to pass expert_tours to the forward
        # method to ensure context is updated correctly based on expert moves.
        _, log_probs = self.model(batch_data, teacher_forcing=expert_tours)
```

```
        # Cross Entropy Loss: Maximize log_prob of expert actions
        loss = -log_probs.mean()

        loss.backward()
        self.optimizer.step()
        return loss.item()

    def train_step_rl_pomo(self, batch_data):
        """
        Phase 2: Reinforcement Learning (POMO Baseline)
        """
        self.optimizer.zero_grad()

        # 1. Generate N trajectories (POMO multi-start)
        # We assume batch_data is expanded to (Batch * N, Nodes, Feats)
        # to simulate starting from every node.
        tours, log_probs = self.model(batch_data, decode_type="sampling")

        # 2. Calculate Reward (Negative Tour Length)
        rewards = -self.calculate_tour_length(tours, batch_data) # (Batch, N)

        # 3. Calculate POMO Baseline
        # Mean reward across the N different start nodes for the SAME instance
        baseline = rewards.mean(dim=1, keepdim=True)

        # 4. Advantage
        advantage = rewards - baseline

        # 5. REINFORCE Loss
        # We minimize negative expected reward
        loss = -(advantage.detach() * log_probs.sum(dim=1)).mean()

        loss.backward()
        self.optimizer.step()
        return loss.item()

    def calculate_tour_length(self, tours, coords):
        # Implementation of Euclidean distance sum along sequence
        pass
```

5.3 Implementation Instructions

1. **Data Generation:** Create a generator for CVRP instances (e.g., $N=100$, capacity=50, demand $\in $).
2. **Expert Generation:** Use the PyVRP library or LKH-3 binary to solve 10,000 training instances. Save these as (instance, expert_tour) pairs.
3. **Phase 1 Training:** Run train_step_imitation for approx 50 epochs. This creates a policy that outputs valid, decent tours.
4. **Phase 2 Training:** Discard the expert labels. Run train_step_rl_pomo for 100+ epochs. Use the POMO augmentation strategy (augmenting batch size by N start nodes).
5. **Validation:** Monitor the "Optimality Gap" relative to LKH-3 on a held-out test set (e.g., CVRPLIB).

6. Critical Analysis and Challenges

6.1 Catastrophic Forgetting in Sequential Hybridization

A significant risk in the "Pretrain-Finetune" strategy (Section 3.1) is **Catastrophic Forgetting**. As the model shifts from minimizing Cross-Entropy (imitation) to maximizing Reward (RL), it may lose the structural stability learned from the expert.

- **Symptom:** The model starts generating invalid tours (loops, capacity violations) as it explores aggressively.
- **Mitigation:**
  - **Mixed Batches:** Keep a small percentage of expert data in the RL batches (Experience Replay).[26]
  - **KL-Regularization:** Add a penalty term to the RL loss: $\lambda D_{KL}(\pi_\theta |

| \pi_{pretrained})$, constraining the policy from deviating too far from the imitation prior.[28]

6.2 The Generalization Gap (Sim-to-Real)

Most NCO research benchmarks on uniform random distributions (U^2). Real-world CVRPs have clustered cities, non-Euclidean distances (road networks), and time windows.

- **Failure Mode:** A model trained on uniform data fails catastrophically on clustered data (e.g., CVRPLIB "Golden" instances).[29]
- **Solution:** Hybrid methods that use **Generative Adversarial** concepts, where a "Generator" creates hard instances (adversarial examples) and the "Solver" tries to optimize them, improve robustness. Additionally, **Multi-View Decision Fusion (MVDF)**—rotating and scaling the input during inference and averaging the logits—provides significant boosts in zero-shot generalization.[6]

6.3 Scalability

The attention mechanism in Transformers is $O(N^2)$. For $N=10,000$, this is infeasible.

- **Current Limit:** End-to-end constructive RL is effectively capped at $N \approx

500-1000$.

- **Path Forward:** Hybrid Decomposition (L2D) is the only viable path for massive scale. By learning to partition the graph, the neural component only ever sees subgraphs of size $N=100$, where it is highly effective.[21]

7. Conclusions and Future Outlook

The field of Neural Combinatorial Optimization has matured from proof-of-concept architectures to sophisticated hybrid systems that rival traditional solvers. The integration of Imitation Learning provides the necessary structural prior—the "grammar" of valid solutions—while Reinforcement Learning provides the mechanism for optimization and discovery beyond the expert's limitations.

Key Takeaways:

1. **Hybrid is Necessary:** Pure RL is too unstable; Pure IL is too rigid. The combination, particularly via Warm-Starting and Self-Improvement (Gumbeldore/POMO), is the current SOTA.
2. **Symmetry is Power:** Exploiting the symmetries of the CVRP (via POMO or Sym-NCO) allows for "free" self-supervision and variance reduction.
3. **From Solving to Delegating:** For large-scale problems, the future lies not in giant monolithic Transformers, but in hierarchical systems (L2D) where neural networks act as managers that delegate sub-tasks to specialized solvers.
4. **Code Generation:** The emergence of LLMs that write heuristic code (RFTHGS) opens a new frontier: utilizing the reasoning capabilities of foundation models to discover novel algorithmic logic rather than just predicting geometric patterns.

Recommendation: For practical deployment, adopt the **POMO** architecture initialized via **Imitation Learning** on a diverse dataset (uniform + clustered), and implement **Stochastic Beam Search** for inference. For problems exceeding 500 nodes, wrap this model in a **Learning to Delegate (L2D)** framework.

This convergence of techniques marks the transition of NCO from an academic curiosity to a viable competitor in industrial optimization engines.

Works cited

1. Machine and deep learning models for vehicle routing problems: a literature review, accessed December 29, 2025, https://repository.tudelft.nl/file/File_d51bfd08-8a70-4354-8a52-672b5a7d6dc0
2. A Reinforcement Learning Framework for Scalable Partitioning and Optimization of Large-Scale Capacitated Vehicle Routing Problems - MDPI, accessed December 29, 2025, https://www.mdpi.com/2079-9292/14/19/3879
3. IMITATION IMPROVEMENT LEARNING FOR LARGE- SCALE CAPACITATED VEHICLE ROUTING PROBLEMS - OpenReview, accessed December 29, 2025,

https://openreview.net/pdf?id=K5UfKyHIBS
4. NeuroLKH: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem - Institutional Knowledge (InK) @ SMU, accessed December 29, 2025, https://ink.library.smu.edu.sg/context/sis_research/article/9163/viewcontent/NeurIPS_2021_neurolkh_combining_deep_learning_model_with_lin_kernighan_helsgaun_heuristic_for_solving_the_traveling_salesman_problem_Paper.pdf
5. Neural Combinatorial Optimization for Real-World Routing - arXiv, accessed December 29, 2025, https://arxiv.org/html/2503.16159v1
6. Improving Generalization of Neural Combinatorial Optimization for Vehicle Routing Problems via Test-Time Projection Learning - arXiv, accessed December 29, 2025, https://arxiv.org/html/2506.02392
7. Reinforcement Learning for Solving the Vehicle Routing Problem, accessed December 29, 2025, http://papers.neurips.cc/paper/8190-reinforcement-learning-for-solving-the-vehicle-routing-problem.pdf
8. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning, accessed December 29, 2025, https://papers.neurips.cc/paper_files/paper/2020/file/f231f2107df69eab0a3862d50018a9b2-Paper.pdf
9. (PDF) Efficient Active Search for Combinatorial Optimization Problems - ResearchGate, accessed December 29, 2025, https://www.researchgate.net/publication/352280422_Efficient_Active_Search_for_Combinatorial_Optimization_Problems/download
10. Self-Improvement for Neural Combinatorial Optimization: Sample Without Replacement, but Improvement - OpenReview, accessed December 29, 2025, https://openreview.net/pdf?id=agT8ojoH0X
11. Research - portal-cornell.github.io, accessed December 29, 2025, https://portal.cs.cornell.edu/research/
12. Learning to Drive by Imitating Surrounding Vehicles - arXiv, accessed December 29, 2025, https://arxiv.org/html/2503.05997v1
13. A Deep Reinforcement Learning Model to Solve the Stochastic Capacitated Vehicle Routing Problem with Service Times and Deadlines - MDPI, accessed December 29, 2025, https://www.mdpi.com/2227-7390/13/18/3050
14. Learning to Handle Complex Constraints for Vehicle Routing Problems - NIPS papers, accessed December 29, 2025, https://proceedings.neurips.cc/paper_files/paper/2024/file/a9d2a5fd12d34250c21b5e4fa8d906b0-Paper-Conference.pdf
15. Routing Problems - RL4CO, accessed December 29, 2025, http://rl4co.ai4co.org/docs/content/api/envs/routing/
16. Learning Generalizable Models for Vehicle Routing Problems via Knowledge Distillation, accessed December 29, 2025, https://proceedings.neurips.cc/paper_files/paper/2022/file/ca70528fb11dc8086c6a623da9f3fee6-Paper-Conference.pdf
17. COFormer: Towards a Foundation Model for Solving Combinatorial Optimization

Problems, accessed December 29, 2025, https://openreview.net/forum?id=5mN7TWH6OI
18. Hybrid Genetic Algorithm and Deep Reinforcement Learning Framework for IoT-Enabled Healthcare Equipment Maintenance Scheduling - MDPI, accessed December 29, 2025, https://www.mdpi.com/2079-9292/14/21/4160
19. Self-Improvement for Neural Combinatorial Optimization: Sample Without Replacement, but Improvement - arXiv, accessed December 29, 2025, https://arxiv.org/html/2403.15180v2
20. Multi-Action Self-Improvement For Neural Combinatorial Optimization - arXiv, accessed December 29, 2025, https://arxiv.org/html/2510.12273v1
21. Learning to Delegate for Large-scale Vehicle Routing, accessed December 29, 2025, https://papers.neurips.cc/paper_files/paper/2021/file/dc9fa5f217a1e57b8a6adeb065560b38-Paper.pdf
22. Symmetric Exploration in Combinatorial Optimization is Free! - OpenReview, accessed December 29, 2025, https://openreview.net/pdf?id=a5RnAsDAYP
23. Refining Hybrid Genetic Search for CVRP via Reinforcement Learning-Finetuned LLM, accessed December 29, 2025, https://openreview.net/forum?id=alTKXFeivk
24. [論文評述] Refining Hybrid Genetic Search for CVRP via Reinforcement Learning-Finetuned LLM - Moonlight, accessed December 29, 2025, https://www.themoonlight.io/tw/review/refining-hybrid-genetic-search-for-cvrp-via-reinforcement-learning-finetuned-llm
25. Refining Hybrid Genetic Search for CVRP via Reinforcement Learning-Finetuned LLM, accessed December 29, 2025, https://www.researchgate.net/publication/396459979_Refining_Hybrid_Genetic_Search_for_CVRP_via_Reinforcement_Learning-Finetuned_LLM
26. Enhancing the Cross-Size Generalization for Solving Vehicle Routing Problems via Continual Learning - arXiv, accessed December 29, 2025, https://arxiv.org/html/2510.10262v1
27. Navigating Memory Construction by Global Pseudo-Task Simulation for Continual Learning, accessed December 29, 2025, https://proceedings.neurips.cc/paper_files/paper/2022/hash/3013680bf2d072b5f3851aec70b39a59-Abstract-Conference.html
28. PolyNet: Learning Diverse Solution Strategies for Neural Combinatorial Optimization - arXiv, accessed December 29, 2025, https://arxiv.org/html/2402.14048v2
29. Improving Generalization of Neural Combinatorial Optimization for Vehicle Routing Problems via Test-Time Projection Learning | OpenReview, accessed December 29, 2025, https://openreview.net/forum?id=QPJjiNCRq1
30. On the Generalization of Neural Combinatorial Optimization Heuristics - AWS, accessed December 29, 2025, https://ecmlpkdd-storage.s3.eu-central-1.amazonaws.com/preprints/2022/lncs13717/lncs13717425.pdf
31. Improving Generalization of Neural Combinatorial Optimization for Vehicle Routing Problems via Test-Time Projection Learning - arXiv, accessed December

29, 2025, https://arxiv.org/html/2506.02392v2
32. (PDF) A Reinforcement Learning Framework for Scalable Partitioning and Optimization of Large-Scale Capacitated Vehicle Routing Problems - ResearchGate, accessed December 29, 2025, https://www.researchgate.net/publication/396031896_A_Reinforcement_Learning_Framework_for_Scalable_Partitioning_and_Optimization_of_Large-Scale_Capacitated_Vehicle_Routing_Problems