SAG-Sep: Sparse Augmented Graphs and Onion-Guided Search for Rounded Capacity Cut Separation

Haoran Liu
Department of Industrial Systems Engineering
University of Florida
Florida, United States
haoran.liu@ufl.edu

Guanyi Wang
Department of Industrial Systems Engineering and Management
National University of Singapore
Singapore
Guanyi.W@nus.edu.sg

Yu Yang
Department of Industrial Systems Engineering
University of Florida
Florida, United States
yu.yang@ise.ufl.edu

Abstract

Rounded capacity cuts (RCCs) are among the most effective cuts for capacitated vehicle-routing relaxations, but exact separation is computationally prohibitive at scale. Existing heuristic and neural separators can be fast, but often fail to produce sufficiently many effective cuts within practical separation budgets. We introduce SAG-Sep, a separator that scores node membership in RCC-inducing subsets across vehicle-count levels from a fractional LP solution in a single encoder forward pass, avoiding the iterative predict-and-coarsen inference used by NeuralSEP-style separators. SAG-Sep encodes the LP solution as a Sparse Augmented Graph (SAG), adding flow-informed multi-hop edges that expose long-range routing structure without densifying attention. Beyond LP-aware sparse encoding, we observe that high-value exact RCCs are typically nested, organizing into onion-like subset families, and SAG-Sep's node scores track subset nesting depth. This motivates Onion-Sweep (OS), a training-free decoder that ranks customers by their scores, evaluates the RCC violation of every prefix of the ranking in one vectorized pass, and emits the most violated prefix of each replica together with a cross-replica family of nested proposals, converting independent node scores into structurally coordinated RCC candidates without a tuned threshold or probing band. On the NeuralSEP benchmark with $N = 1000$ customer instances, SAG-Sep with OS reduces the average root gap by 17.7% relative to the strongest NeuralSEP baseline, while the single-pass SAG-Sep separator achieves a roughly 30-fold per-iteration neural separation speedup. Our findings suggest that LP-aware sparse encoding and the nested structure of high-value cuts are structural patterns worth leveraging in neural separation more broadly.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

The capacitated vehicle routing problem (CVRP) is a canonical NP-hard combinatorial optimization problem and a central testbed for large-scale routing algorithms [Uchoa et al., 2017]. State-of-the-art exact solvers are built on branch-and-cut (B&C) or branch-price-and-cut (BPC) frameworks [Lysgaard et al., 2004, Pessoa et al., 2020, You and Yang, 2025], where cutting planes iteratively tighten linear relaxations to improve dual bounds. Among the many valid inequalities proposed for CVRP, *rounded capacity cuts* (RCCs) are particularly effective and are routinely used in both B&C and BPC solvers [Lysgaard et al., 2004, Fukasawa et al., 2006].

Despite their effectiveness, RCCs pose a fundamental scalability challenge. Separating violated RCCs is NP-hard [Diarrassouba, 2017], and exact separation formulations rapidly become computationally prohibitive as the number of customers grows [Augerat et al., 1995, Fukasawa et al., 2006, Pavlikov et al., 2024]. Although exact methods provide strong per-iteration tightening of the linear program (LP) relaxation, they are impractical at large scales, forcing solvers to rely on heuristics. The widely used CVRPSEP heuristic [Lysgaard, 2003] performs well on small-to-medium instances, but on larger problems it often enters a low-throughput regime, generating few effective cuts and delivering only marginal improvements in the dual bound.

Recent work has explored learning-based alternatives. NeuralSEP [Kim et al., 2024] introduced the first neural RCC separator by training a graph neural network (GNN) to predict cut-defining customer subsets from fractional LP solutions. However, operating solely on the sparse LP-support graph limits scalability: fixed-depth message passing captures only local structure, while deeper networks suffer from over-smoothing and over-squashing [Li et al., 2018, Alon and Yahav, 2020, Wu et al., 2021]. NeuralSEP addresses this with recursive graph coarsening, but this multi-stage pipeline incurs substantial training and inference overhead, restricts GPU parallelism, and requires repeated CPU–GPU synchronization. These limitations motivate a separator that captures long-range dependencies without sequential coarsening and instead enables highly parallel, hardware-efficient computation.

To address these challenges, we propose **SAG-Sep**, a single-pass RCC separator built around a **Sparse Augmented Graph** (SAG) representation of the current LP solution. SAG starts from the LP-support graph and augments it with a small number of flow-informed multi-hop edges, exposing long-range routing structure while retaining sparse attention. Given this representation, SAG-Sep scores customer membership in RCC-inducing subsets across vehicle-count levels in one encoder pass, avoiding the iterative predict-and-coarsen procedure of NeuralSEP-style separators.

The continuous vehicle-count conditioning also creates a richer test-time search problem. Aggressive probing of the frozen network showed that it contains useful candidate cuts beyond the hard-threshold subset of each replica, but such probing treats the replicas as independent predictions. Studying the exact RCC separator revealed a consistent pattern: high-value exact cuts often form nested *onion* families, providing cut diversity through ordered subset growth rather than unrelated threshold decisions. This observation leads to **Onion-Sweep (OS)**, a training-free decoder that reads subsets directly off the score ranking instead of thresholding it: within each replica, OS evaluates the RCC violation of every prefix of the score ranking in one vectorized pass and emits the most violated one; across replicas, it aggregates the scores into a global priority ordering and emits the nested prefixes at which the required vehicle count increases. No tuned threshold, band width, or sampling is involved, and the trained encoder is unchanged.

1.1 Contributions

We summarize our contributions as follows:

- We introduce **SAG-Sep**, a one-shot RCC separator based on a **Sparse Augmented Graph** encoding of the fractional LP solution. The SAG representation preserves the sparsity of the LP-support graph while adding flow-informed multi-hop edges, enabling the model to capture long-range routing dependencies without dense attention or NeuralSEP-style iterative coarsening. This design substantially improves GPU efficiency and sharply reduces neural separation time while retaining strong root-node bounds.
- We study test-time enhancement in the learned probability space: aggressive conditioned probing, which densifies the vehicle-count conditioning grid, improves root bounds and re-

veals that the trained network contains useful cuts beyond those exposed by hard-thresholded replicas.

- We further analyze the learned score space together with exact RCC separation and identify a stable nested, onion-like structure among high-value exact cuts. This motivates **Onion-Sweep (OS)**, a training-free decoder that replaces hard thresholding by a violation-guided sweep over the score ranking within each replica and by onion-guided nested proposals across replicas, removing the threshold dependence of independently thresholded outputs and substantially improving separation effectiveness.
- More broadly, the design illustrates how a neural module can take over an NP-hard separation subroutine at the root node of an exact solver, accelerating relaxation tightening without sacrificing the validity of generated cuts.

1.2 Related Work

End-to-end learning for CVRP. A substantial body of work applies machine learning to CVRP to produce high-quality *primal* solutions: neural construction heuristics formulated routing as sequence generation with pointer networks [Vinyals et al., 2015], strengthened by reinforcement learning [Nazari et al., 2018], attention-based architectures [Kool et al., 2019], and symmetry- and multi-start-based training such as POMO [Kwon et al., 2020] and Sym-NCO [Kim et al., 2022].

Recent methods scale to very large instances through decomposition or divide-and-conquer [Li et al., 2021, Ye et al., 2024, Zheng et al., 2024], or *improve* incumbents through learned local search [Lu et al., 2019, Xin et al., 2021, Ma et al., 2023]. Although often highly competitive on large benchmarks, these neural heuristics operate purely in the primal space and typically provide no guarantees on solution quality.

Learning in exact solvers. A complementary direction incorporates learning directly into *exact* solvers, leveraging solver states such as LP solutions and dual prices within B&C or BPC frameworks while preserving optimality guarantees [Bengio et al., 2021]. A major line of work focuses on *learning to branch* [Khalil et al., 2016, Alvarez et al., 2017, Gasse et al., 2019], with extensions to more general branching schemes [Yang et al., 2021, 2022] and multi-stage procedures [You et al., 2026b]. Learning has also been explored for *column generation*, where models propose or prioritize promising columns to reduce pricing effort [Zhang et al., 2022, Morabit et al., 2021], prefill the master with learned column pools [Liu and Yang, 2026], or stabilize the dual iterates with learned dual-optimal inequalities [You et al., 2026a].

In contrast, *learning for cutting planes* remains underexplored: most work addresses *cut selection* from large candidate pools [Tang et al., 2020, Huang et al., 2022], whereas learning-based *cut generation*, directly proposing new valid inequalities, is much less studied, particularly when exact separation is intractable. We target this gap by predicting RCC-defining subsets directly from fractional LP solutions, extending recent neural separation approaches [Kim et al., 2024].

2 Preliminaries

2.1 The Capacitated Vehicle Routing Problem (CVRP)

Let $G = (V, E)$ be a complete undirected graph with depot 0 and customer set $V^+ := V \setminus \{0\}$. Each customer $i \in V^+$ has integer demand $d_i \in \mathbb{Z}_{>0}$ and all vehicles have integer capacity $C \in \mathbb{Z}_{>0}$, with $d_i \leq C$ and $d_0 = 0$. Let $c_{ij} \geq 0$ denote the travel cost of edge $\{i, j\} \in E$. The CVRP seeks a set of depot-rooted routes that serve every customer exactly once, incurring the minimum possible total travel cost.

A classical two-index formulation uses edge variables x_{ij}, with $x_{ij} \in \{0, 1\}$ for customer–customer edges and $x_{0i} \in \{0, 1, 2\}$ for depot edges (allowing two traversals in undirected formulations). The degree constraints $\sum_{j \in V \setminus \{i\}} x_{ij} = 2,\ \forall i \in V^+$ ensure that each customer has exactly two incident edges, and hence is visited exactly once. The objective is $\min \sum_{\{i,j\} \in E} c_{ij}\, x_{ij}$. To tighten the relaxation and eliminate infeasible subtours, one can iteratively add capacity inequalities. For any subset $S \subseteq V^+$, define the cutset $\delta(S) := \{\{i, j\} \in E : i \in S,\ j \in V \setminus S\}$ and its load

$x(\delta(S)) := \sum_{\{i,j\}\in\delta(S)} x_{ij}$. The (strong) capacity cut is

$$x(\delta(S)) \;\geq\; 2\,r(S), \qquad \forall S \subseteq V^+,\, |S| \geq 2,$$

where $r(S)$ denotes the minimum number of vehicles required to serve the customers in S.

Since computing $r(S)$ is NP-hard, solvers typically use the rounded lower bound $r(S) \geq \lceil d(S)/C \rceil$, where $d(S) := \sum_{i\in S} d_i$, yielding the *rounded capacity cut* (RCC):

$$x(\delta(S)) \;\geq\; 2\left\lceil \frac{d(S)}{C} \right\rceil, \qquad \forall S \subseteq V^+,\, |S| \geq 2.$$

Because RCCs form an exponential family, solvers generate them on demand: they repeatedly solve the current LP relaxation and invoke separation routines to detect violated RCCs, which are added until the relaxation is sufficiently tight or further bound improvements become marginal.

2.2 Exact Separation of RCCs

Given a fractional LP solution $\bar{x} \in \mathbb{R}^E_+$, RCC separation aims to identify a customer subset $S \subseteq V^+$ such that the capacity inequality is violated, i.e., $\bar{x}(\delta(S)) < 2\,\lceil d(S)/C \rceil$. Exact separation seeks the subset with the maximum violation.

Following the classical formulation of Fukasawa et al. [2006], we enumerate candidate right-hand sides by fixing the number of required vehicles. For each $m \in \{0, \ldots, \lceil d(V^+)/C \rceil - 1\}$, we solve a mixed-integer program that searches for a subset S whose total demand exceeds mC (thus requiring at least $m{+}1$ vehicles) while minimizing the crossing value $\bar{x}(\delta(S))$. Intuitively, this formulation looks for a "light" cut in the fractional solution that separates a demand-heavy customer group.

Let $y_i \in \{0,1\}$ indicate whether customer i belongs to S ($y_i = 1$ iff $i \in S$), and let $\mu_{ij} \geq 0$ indicate whether edge $\{i,j\}$ crosses the induced cut. The separation problem is:

$$\begin{aligned}
\min_{y,\mu}\; & z(m) \;=\; \sum_{\{i,j\}\in E} \bar{x}_{ij}\,\mu_{ij} && \\
\text{s.t. }\; & \mu_{ij} \;\geq\; y_i - y_j, && \forall \{i,j\} \in E,\\
& \mu_{ij} \;\geq\; y_j - y_i, && \forall \{i,j\} \in E,\\
& mC + 1 \;\leq\; \sum_{i\in V^+} d_i\, y_i, && \\
& y_0 \;=\; 0, && \\
& y_i \in \{0,1\}, && \forall i \in V^+,\\
& \mu_{ij} \;\geq\; 0, && \forall \{i,j\} \in E.
\end{aligned}$$

Let (y^*, μ^*) be an optimal solution and define $S(m) = \{i \in V^+ : y^*_i = 1\}$. The corresponding RCC has a right-hand side of at least $2(m+1)$ and leads to a violation whenever $z(m) < 2(m+1)$. This violated inequality is added to the LP relaxation. We further discuss this exact separation formulation and its variants in Appendix B.

3 Algorithmic Design of SAG-Sep

3.1 Pipeline Overview

Figure 1 summarizes the cut-generation pipeline of **SAG-Sep** within a single cutting-plane iteration. Given the current fractional LP solution $\bar{x}$, we first construct the LP support graph G_{LP} and the corresponding sparse augmented graph $\widetilde{G}$ as defined in Section 3.2. The pipeline further integrates with **Onion-Sweep (OS)**, a training-free decoder introduced in Section 3.3.

Overall, we use graph replication to produce J augmented graphs $\left\{\widetilde{G}^{(j)}\right\}_{j=1}^{J}$ from $\widetilde{G}$, and process this batch in a single encoder forward pass. The model returns node probabilities $\left\{p_i^{(j)}\right\}_{i\in V^+}$, where

$p_i^{(j)}$ estimates whether customer i belongs to an RCC-defining subset. Candidate subsets $S \subseteq V^+$ are then constructed either by simple thresholding or by the more structured OS decoder.

Each candidate subset is translated into a valid RCC by computing the rounded bound $\lceil d(S)/C \rceil$ and evaluating its violation under $\bar{x}$, i.e., the difference $2\lceil d(S)/C \rceil - \bar{x}(\delta(S))$. Any violated RCC is added to the LP relaxation, which is then re-optimized to obtain an updated solution $\bar{x}$. This *separate–add–resolve* loop continues until either (i) no violated RCC is identified, or (ii) a prescribed time or iteration budget is exhausted.

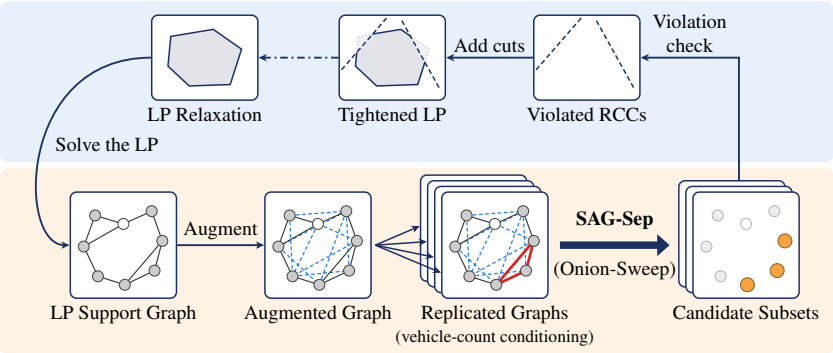


Figure 1: Pipeline overview of SAG-Sep for RCC separation. The current LP solution is encoded as a sparse augmented graph, replicated and evaluated in one batched forward pass, and converted into candidate subsets by thresholding or Onion-Sweep. Violated RCCs are added back to the LP relaxation in a separate–add–resolve loop.

3.2 Augmented LP Support Graph

We first construct a sparse *LP support graph* and then augment it with a small set of informative multi-hop edges, yielding an augmented graph $\widetilde{G}$ on which SAG-Sep performs masked self-attention.

LP support graph. Following the same notation, we define the LP support graph

$$G_{\mathrm{LP}} = (V, E_{\mathrm{LP}}), \qquad E_{\mathrm{LP}} := \{\{i,j\} \in E : \bar{x}_{ij} \geq \tau_1\},$$

where $\tau_1 > 0$ is a small threshold. This filtering step retains only edges that carry nontrivial LP mass, resulting in a sparse graph that reflects the dominant structure of the current LP relaxation.

K-hop reachability. For an integer $K \geq 2$, let $\mathrm{dist}_{\mathrm{LP}}(i,j)$ denote the shortest-path distance (i.e., the number of edges) from i to j in G_{LP}, with $+\infty$ if i and j are not connected. Define the K-hop neighborhood as

$$\mathcal{N}_K(i) := \{j \in V \setminus \{i\} : \ \mathrm{dist}_{\mathrm{LP}}(i,j) \leq K\}.$$

Multi-hop LP connectivity score. For any $j \in \mathcal{N}_K(i)$, define the connectivity score as

$$e_{ij}^{\mathrm{flow}} := \max_{p: i \rightsquigarrow j,\ |p| = \mathrm{dist}_{\mathrm{LP}}(i,j) \leq K} \ \min_{(u,v) \in p} \bar{x}_{uv},$$

where the maximum is taken over all shortest paths p connecting i with j of length at most K. This is the largest bottleneck value among the shortest LP-support paths between the two nodes, a hop-limited variant of the classical *widest-path* value. By construction, $e_{ij}^{\mathrm{flow}} = \bar{x}_{ij}$ when $\{i,j\} \in E_{\mathrm{LP}}$ (via a 1-hop path), while larger values of e_{ij}^{flow} indicate that i and j are linked through a short path whose edges all take large LP values in $\bar{x}$.

Augmented graph construction. We define the augmented edge set as

$$\widetilde{E} \ := \ E_{\mathrm{LP}} \ \cup \ E_{\mathrm{syn}},$$

where E_{syn} contains *synthetic* edges that connect multi-hop neighbors:

$$E_{\mathrm{syn}} := \{\{i,j\} : i,j \in V,\ j \in \mathcal{N}_K(i),\ \{i,j\} \notin E_{\mathrm{LP}}\}.$$

G_{LP} is typically sparse; with small K, the edge growth under the K-hop closure remains controlled.

Proposition 3.1. *Let $N = |V|$. Assume the undirected LP-support graph $G_{\mathrm{LP}} = (V, E_{\mathrm{LP}})$ is sampled from the Erdős-Rényi model $\mathcal{G}(N, \rho)$. Let $\widetilde{G} = (V, \widetilde{E})$ denote the K-hop closure of G_{LP}. For fixed $K \geq 1$, let $\lambda_N = \rho(N-1)$ and assume $\lambda_N^K = o(N)$. Then*

$$\frac{\mathbb{E}[|\widetilde{E}|]}{\mathbb{E}[|E_{\mathrm{LP}}|]} = (1 + o(1)) \sum_{k=0}^{K-1} \lambda_N^k.$$

In particular, if $\lambda_N \to \lambda$, the ratio converges to $\sum_{k=0}^{K-1} \lambda^k$.

The proposition (proof in Appendix A.5) quantifies why the hop radius must stay small: under this model the expected edge count grows geometrically in K through the mean LP-support degree λ, so a radius of two keeps the expected $|\widetilde{E}|$, which dominates memory and attention cost, within a factor of about $1 + \lambda$ of $|E_{\mathrm{LP}}|$.

Each edge $\{i, j\} \in \widetilde{E}$ is assigned an input feature vector

$$e_{ij}^{\mathrm{input}} := \left[\, e_{ij}^{\mathrm{flow}},\ \mathrm{dist}_{\mathrm{LP}}(i, j) \,\right].$$

The depot is also joined to every customer; missing depot edges get the feature vector $(0, 0)$. Each node $i \in V$ is assigned a feature vector following NeuralSEP [Kim et al., 2024]:

$$h_i^{\mathrm{input}} = \left[\frac{d_i}{C}, \frac{m}{\max(M-1, 1)} \right],$$

where $M = \lceil d(V^+)/C \rceil$ is the capacity-based lower bound on the number of vehicles required to serve all customers. To encode vehicle-index information, we create $J = M$ replicas of $\widetilde{G}$ indexed by $m \in \{0, 1, \ldots, M-1\}$. Within replica m, all nodes share the same graph-level conditioning value $m/\max(M-1, 1)$, so the conditioning values span $[0, 1]$. In contrast, the demand feature d_i/C and all edge features remain unchanged across the M replicas originating from the same $\widetilde{G}$.

Shared edge encoding. The replicated graphs have identical edge structure and edge features and differ only in their conditioning value. At large scales $\widetilde{E}$ is often much larger than V, so duplicating it across all M replicas creates substantial edge-encoding and attention-memory pressure. We therefore encode edge features once, precompute the edge attention bias once per layer, and stack only the node features across replicas, which reduces the edge-side memory from $O(M|\widetilde{E}|D)$ to $O(|\widetilde{E}|D)$ for hidden dimension D compared with materializing M graph copies; the replica-dependent attention weights still take $O(M|\widetilde{E}|H)$ memory, with $H \ll D$ heads. The two effects are complementary: Proposition 3.1 controls $|\widetilde{E}|$ itself, while the shared encoding removes the factor M (details in Appendix C.3).

3.3 Onion-Sweep Decoding

From thresholds to violation-guided decoding. The basic readout turns the probabilities of replica j into one subset by hard thresholding, $\hat{S}^{(j)} = \{i \in V^+ : p_i^{(j)} \geq \tau\}$, which discards most of the information in the scores: whether a subset induces a violated RCC is decided by the LP solution, not by a fixed cutoff, and a few nodes near τ can flip the induced cut. **Onion-Sweep (OS)** instead treats each replica's score ranking as an ordered family of nested subsets and selects among them by their actual violation, through a per-replica *sweep* and a cross-replica *onion* step; both reuse the encoder probabilities and require no additional forward pass.

Onion-Sweep candidate generation. Input: probabilities $\{p^{(j)}\}_{j=1}^{J}$, demands d, capacity C, current LP solution $\bar{x}$, and the threshold τ (used only as a fallback).

1. Initialize candidate pool $\mathcal{C} \leftarrow \emptyset$.
2. **Per-replica sweep.** For each replica j, sort customers by decreasing $p_i^{(j)}$ into $v_1, \ldots, v_{|V^+|}$ and consider the nested prefixes $P_r = \{v_1, \ldots, v_r\}$. Compute the violation $\nu(P_r) = 2\lceil d(P_r)/C \rceil - \bar{x}(\delta(P_r))$ of every prefix in one pass, using $\bar{x}(\delta(P_r)) = \bar{x}(\delta(P_{r-1})) + \deg(v_r) - 2\sum_{t<r} \bar{x}_{v_t v_r}$ with $\deg(i) = \sum_{k \in V} \bar{x}_{ik}$, and accumulating $d(P_r)$ likewise. Let $r^\star$ be the first maximizer of $\nu(P_r)$ over $r \geq 2$. If $\nu(P_{r^\star}) > \varepsilon$, add $P_{r^\star}$ to $\mathcal{C}$; otherwise add the hard-threshold subset $\hat{S}^{(j)}$.
3. **Cross-replica onion.** Compute the soft frequency score $a_i = \frac{1}{J}\sum_{j=1}^{J} p_i^{(j)}$, sort customers as $v_1, \ldots, v_{|V^+|}$ by decreasing a_i, form nested prefixes $P_k = \{v_1, \ldots, v_k\}$, and add P_k to $\mathcal{C}$ whenever $r_k = \lceil d(P_k)/C \rceil$ reaches a new vehicle-count level.
4. **Finalize subsets.** Remove duplicates and subsets whose RCC is already in the LP, and keep a proposed subset S only if $2\lceil d(S)/C \rceil - \bar{x}(\delta(S)) > 0$.

Output: violated RCC-defining subsets in $\mathcal{C}$.

Violation-guided sweep. The sweep replaces the single cutoff τ by an exhaustive, deterministic evaluation of the $|V^+|$ nested prefixes induced by the ranking. Because $\bar{x}(\delta(P_r))$ and $d(P_r)$ are updated incrementally, all prefix violations of a replica are obtained in one vectorized pass over the LP-support adjacency, at a cost that is negligible compared with the subsequent LP re-optimization. The threshold subset $\hat{S}^{(j)}$ is itself a prefix of the same ranking, so the sweep never returns a less violated subset than the threshold readout, and the fallback fires only when no prefix of size at least two is violated; the fallback subset is then itself a non-violated prefix and is removed by the final filter, so up to the tolerance ε the output of OS does not depend on τ. OS has no band width, probing strategy, or random component; the tolerance $\varepsilon = 10^{-6}$ and the minimum subset size of two are fixed throughout.

Onion-guided nested proposals. The sweep still treats each replica independently. Exact RCC separation suggests a more structured view: when exact cuts are ordered by their required vehicle count, high-value subsets often form nested *onion* families (the banded exact separator of Appendix B reflects this one-cut-per-demand-band structure, and Appendix B.4 quantifies the nesting of exact labels and its alignment with the encoder's scores). The onion step uses the cross-replica soft frequency score as a global priority ordering, so high-priority customers enter many nested candidates while lower-priority customers enter only in larger prefixes, converting independent node scores into a coordinated family of nested RCC proposals, one per vehicle-count level; onion prefixes with violation below 10^{-4} are discarded before the final filter.

Relation to margin-band probing. Our earlier decoder, Onion-MBP (Appendix C.4), kept the hard threshold and explored a margin band of width Δ around it with four fixed probes, combined with the same onion step. OS removes the dependence on τ and Δ: instead of perturbing nodes near τ, it evaluates the exact violation of every prefix of the ranking; since two of the probes are not prefixes, OS does not contain Onion-MBP as a special case. Onion-MBP is kept as an ablation in Section 4.4.

4 Experiments

In this section, we organize the empirical evaluation around three main claims:

(i) **Single-pass competitiveness.** SAG-Sep matches, and can even outperform, recursive coarsening pipelines while using a deterministic single-pass separator.

(ii) **Sparse augmented graph design.** Structured graph augmentation components drive the separation improvement.

(iii) **Structured test-time decoding.** Onion-Sweep further strengthens SAG-Sep by exploiting the violation profile of the score ranking and cross-replica nested structure.

Table 1: Root-node performance on CVRP test instances. Each cell reports mean optimality gap (%, top) and mean runtime in seconds (bottom, in parentheses). Lower is better for both metrics.

Method	50	75	100	200	300	400	500	750	1000
Exact	1.98	2.76	4.52	6.35	11.98	18.89	24.87	41.00	50.61
	(55.45)	(203.97)	(671.00)	(3593.72)	(3651.60)	(3698.08)	(3718.09)	(4252.43)	(4624.86)
CVRPSEP	1.97	2.77	4.54	6.28	7.78	11.89	15.68	24.82	28.72
	(0.22)	(1.89)	(7.12)	(212.37)	(1800.42)	(3604.61)	(3608.27)	(3602.54)	(3609.18)
NeuralSEP$_2$	5.25	5.16	6.41	8.31	10.09	13.52	14.48	21.83	26.09
	(4.11)	(13.64)	(35.86)	(181.47)	(893.64)	(1948.70)	(3276.36)	(3672.58)	(3740.21)
π-NeuralSEP$_2$+GC	3.68	4.40	5.97	7.69	8.67	10.81	13.11	20.65	25.82
	(6.12)	(16.33)	(37.28)	(282.44)	(1833.29)	(3075.62)	(3639.13)	(3723.46)	(3842.77)
SAG-Sep	4.37	5.91	7.11	9.10	9.42	**9.98**	**11.35**	**17.41**	**22.03**
	(0.32)	(1.22)	(2.82)	(23.01)	(343.30)	**(1897.26)**	**(3045.45)**	(3650.99)	(3655.57)

4.1 Experimental Setup

We evaluate SAG-Sep against representative exact, heuristic, and learning-based RCC separators. The exact baseline is the classical exact RCC separator of Fukasawa et al. [2006]; CVRPSEP [Lysgaard et al., 2004, You and Yang, 2025] is run with only its RCC routines enabled; NeuralSEP$_1$ [Kim et al., 2024] is the original recursive-coarsening neural separator, while NeuralSEP$_2$ [Sim et al., 2025] is its upgraded pipeline. For the NeuralSEP family, our main comparison focuses on NeuralSEP$_2$ and its stochastic coarsening policy π and GraphChip (GC) variants (details in Appendix D; exact separation in Appendix B). All methods are evaluated at the root node of the cutting-plane procedure on the NeuralSEP benchmark testing instances, using the CVRP formulation described in Section 2.1.

All main-table results use a single AMD EPYC 9655P CPU thread for LP solving, a single NVIDIA L4 GPU for neural inference, and a 3600-second root-relaxation time limit, checked between separation rounds (so times can exceed 3600 s); no main result uses multi-GPU inference or multi-threaded LP solving. Code, the trained checkpoint, and the benchmark instances are available at `https://github.com/henryleo180/SAG-Sep`. Training details are provided in Appendix C; additional ablations over hop radius, threshold choices, numerical precision, and test-time settings are in Appendix D.

4.2 Single-Pass Separation vs. Recursive Coarsening

Table 1 reports the main root-node comparison: the mean optimality gap $100\,(U - L)/U$ of the final root bound L to the best-known solution value U (BKS), and the total root-node runtime in seconds under the single-thread LP and single-GPU setting.

On small and medium instances, CVRPSEP remains hard to beat: it encodes decades of CVRP-specific separation heuristics in a tightly engineered package. Recursive NeuralSEP pipelines also obtain good gaps, but the improvement over SAG-Sep is modest relative to their runtime cost. At $N = 200$, for example, NeuralSEP$_2$ improves the gap by 0.79 percentage points (pp) over SAG-Sep but is about $7.9\times$ slower; the GC variant improves the gap by 1.41 pp but is about $12.3\times$ slower.

As the instance size grows, this tradeoff turns in favor of SAG-Sep. At $N = 300$ it is faster than both recursive neural pipelines and tighter than NeuralSEP$_2$. From $N = 400$ onward, SAG-Sep gives the best gaps in Table 1; at $N = 400$–500 it also has the lowest runtime, while at $N \geq 750$ every method hits the time limit. At $N = 1000$, it reduces the gap from 25.82% for π-NeuralSEP$_2$+GC and 28.72% for CVRPSEP to 22.03%, with a per-round neural separation time of only 0.65s—versus 16–19s for the NeuralSEP pipelines (Appendix D.2). Appendix D.1 further shows that SAG-Sep is much stronger than the one-shot NeuralSEP base model, indicating that the large-scale gain comes from the sparse augmented encoding and the cleaner single-pass structure, not only from a speed advantage.

SAG-Sep is not meant to replace CVRPSEP's carefully engineered small-instance heuristics; its value is a stronger learned encoding and a high-throughput separation path, which on large instances translate into better bounds under the same root limit and provide a cleaner platform for Onion-Sweep.

Table 2: Ablation on augmented LP graph variants: mean optimality gap (%), lower is better. Separately trained models from an earlier development stage (augmented variants with hop radius 3); rows compare with each other, not with Table 1.

Variant	50	75	100	200	300	400	500	750	1000
Raw	7.79	9.50	9.78	11.95	12.31	12.81	14.51	22.00	27.80
+Conn	7.88	9.97	11.22	14.26	16.19	19.09	21.33	24.71	27.64
+Hop	6.57	7.63	8.28	9.93	10.70	11.40	12.87	19.65	24.63
+Val	**6.37**	**6.41**	**7.38**	**9.00**	**10.04**	**10.88**	**12.16**	**19.23**	**24.04**

4.3 Augmented LP Graphs for RCC Separation

We isolate the effect of graph construction by fixing the separator architecture and varying the LP-graph augmentation. Table 2 reports the main ablation.

Simply increasing connectivity is not enough: adding zero-feature synthetic edges (`+Conn`) degrades performance relative to the raw LP-support graph at eight of nine sizes, since uninformative long-range edges introduce spurious connectivity that the model cannot disentangle from genuine LP support; *how* we add connectivity matters more than *how much*.

Performance improves only when the augmented edges carry interpretable structure: hop-type information (`+Hop`) distinguishes native LP edges from synthetic multi-hop connections and consistently improves over the raw graph, and flow-based edge values (`+Val`) give the best results at all sizes, reflecting that RCC violations are tied to bottleneck and flow structure. We use hop radius 2 (Appendix D.5) and a six-layer transformer (Appendix C) by default.

4.4 Test-Time Refinement

We next compare SAG-Sep with training-free test-time refinements that reuse the same trained encoder. SAG-Sep+Aggr corresponds to the dense conditioned probing strategy from Appendix A.3, combined with cut management. SAG-Sep+OS uses the Onion-Sweep decoder of Section 3.3. SAG-Sep+Onion-MBP, our earlier margin-band decoder, is kept as an ablation (configuration in the caption of Table 3 and Appendix C.4). Paired per-instance statistics for these comparisons are reported in Appendix D.6.

Table 3: Effect of test-time refinements: mean optimality gap (%, top) and mean runtime in seconds (bottom); lower is better. SAG-Sep+Onion-MBP is our earlier decoder, kept as an ablation.

Method	50	75	100	200	300	400	500	750	1000
SAG-Sep	4.37	5.91	7.11	9.10	9.42	9.98	11.35	17.41	22.03
	(0.32)	(1.22)	(2.82)	(23.01)	(343.30)	(1897.26)	(3045.45)	(3650.99)	(3655.57)
SAG-Sep+Aggr	3.27	4.54	5.88	8.07	8.91	9.60	10.77	**16.99**	21.70
	(0.82)	(3.73)	(9.14)	(52.85)	(411.19)	(1863.35)	(3045.63)	(3619.31)	(3635.41)
SAG-Sep+Onion-MBP	3.47	4.18	5.90	8.01	8.85	9.41	**10.66**	17.24	21.26
	(0.24)	**(0.88)**	(2.46)	(27.50)	**(299.47)**	**(1523.99)**	**(2897.77)**	(3665.06)	(3705.87)
SAG-Sep+OS	**2.60**	**3.80**	**5.43**	**7.62**	**8.56**	**9.19**	10.78	17.02	**21.25**
	(0.29)	(0.95)	**(2.37)**	**(22.67)**	(400.33)	(2031.73)	(3105.12)	(3633.69)	(3673.29)

Aggressive sampling is useful but time-consuming. It improves the bounds at all sizes, confirming that denser conditioning grids expose additional violated cuts, but encoding the additional replicas is costly on small instances; on large ones the LP loop dominates and this cost becomes marginal.

Onion-Sweep gives the best bound at most sizes and needs no extra forward pass. Replacing the threshold readout by OS improves the mean root bound at every size and on 85 of the 90 benchmark instances (Appendix D.6): by 1.5–2.1 pp at $N \leq 200$ at equal or lower runtime, and by 0.4–0.9 pp at $N \geq 300$, where at the time-limit-bound sizes $N \geq 500$ repeated runs of one configuration differ by up to 0.4 pp (Appendix C.4). Compared with Onion-MBP, OS removes the threshold, band-width, and cut-management dependence while improving the bound by 0.2–0.9 pp at $N \leq 400$ (71 of 90 paired wins overall); at $N \geq 500$ the two are within 0.25 pp of each other in either direction. Because OS proposes more cuts per round without cut management, its root runtime at $N = 300$–500 exceeds the threshold readout and the cut-managed Onion-MBP. This also helps explain why CVRPSEP remains competitive on small instances: rapidly producing many diverse cuts is itself a powerful tightening mechanism (Appendix B).

5 Limitations

What is established, and what is not. This paper focuses on the root node of the two-index formulation, where we set out to show how much more efficient learned separation can be: SAG-Sep with OS attains lower root gaps than learned and classical separators within the same time limit in large-scale, LP-bound regimes, where LP re-optimization dominates the root loop. At $N \leq 300$ CVRPSEP remains stronger, and the effect on complete branch-and-cut or branch-price-and-cut solves is only preliminarily tested (Appendix E). Beyond the root the LP states change character: in exact-separation branch-and-cut runs on our $N = 50$–100 instances, 86% of the cuts found in the first tenth of the root rounds are capacity-dominated, none in the second half of the root loop or at tree nodes, and the mean violation falls from 7.9 to 0.13 (Appendix E.3); in a route-based master with capacity-feasible closed depot routes, $\bar{x}(\delta(S)) \geq 2d(S)/C$ holds, so only the rounding part can ever be violated. Our labels and decoder target the batch-oriented root regime; the small, rounding-driven violations beyond it need an additional search layer, which we release but do not claim here.

What transfers. The separator consumes only the LP support graph, the demands, and the capacity, so the encoder and decoder plug, without architectural change, into any relaxation that exposes edge values, the host formulation supplying the valid inequality and its violation check: the nodes of a branch-and-cut tree, the root of a branch-price-and-cut solver (where the released integration fine-tunes the checkpoint on column-generation states), and the VRP variants that keep a single capacity (e.g., time windows and duration limits). We regard this component as the reusable part of the work, and the released branch-and-cut examples exercise it zero-shot against CVRPSEP (Appendix E.4): on 30 capacitated, asymmetric and time-windowed instances the engine reaches exactly CVRPSEP's root bounds and both solve all 30; on the open VRP the bounds agree within 0.004 points of gap; on the capacitated minimum spanning tree problem, with the capacity cut as the only family and no CMST training, its root bound is higher on 13 of 20 instances (by up to 5.8 points at tight capacity) and lower on 6 (by at most 0.44). Linehaul–backhaul and pickup-and-delivery loads need variant-specific inequalities, and heterogeneous fleets remain open, since without a single C our vehicle-count conditioning does not apply directly.

Labels, benchmark, confounders. Training relies on exact-separation labels, which are expensive at the training size; smaller-size labels recover much of the performance (Table 8), but label dependence limits fast transfer to other cut families, and we release the checkpoint and the label set. The main evaluation uses the NeuralSEP benchmark distribution, with held-out XML sizes and CVRPLIB X instances as partial mitigation (Appendix D.7). Implementation quality matters as well: the main comparisons run our Python research implementation against the baselines' own code, so a same-language comparison with CVRPSEP is the fairest measure of speed; Appendix E.1 runs the released C engine against the native C CVRPSEP in the same root loop, and Appendix D.2 gives a fixed-round comparison that removes implementation speed altogether. OS removes the threshold and band dependence of Onion-MBP but remains a hand-designed decoder; a learned decoder is future work.

6 Conclusion

We introduced SAG-Sep, a learning-based RCC separator built on a Sparse Augmented Graph representation of LP solutions that adds flow-informed long-range structure to the LP-support graph and enables single-pass neural separation. Exact separation revealed that high-value RCCs are often nested, onion-like, and the encoder reflects this in its scores; Onion-Sweep exploits it as a training-free decoder that tightens root bounds without extra forward passes or threshold tuning.

Beyond RCC separation, our results suggest that cut generation can be reframed as structured prediction over solver states, with attention architectures taking over expensive separation subroutines at the root node. Two directions follow: a learned decoder that reasons over candidate diversity and relaxation improvement could replace the hand-designed test-time logic, and reinforcement learning or online feedback could optimize entire cut sets rather than individual supervised labels.

Acknowledgments and Disclosure of Funding

Haoran Liu and Yu Yang were supported by the National Science Foundation, Division of Civil, Mechanical and Manufacturing Innovation, through grant CMMI-2309667. The authors declare no competing interests.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

Philippe Augerat, D Naddef, JM Belenguer, E Benavent, A Corberan, and Giovanni Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. 1995.

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

Ibrahima Diarrassouba. On the complexity of the separation problem for rounded capacity inequalities. *Discrete Optimization*, 25:86–104, 2017.

Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Zeren Huang, Kerong Wang, Furui Liu, Hui-Ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123:108353, 2022.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Hyeonah Kim, Jinkyoo Park, and Changhyun Kwon. A neural separation algorithm for the rounded capacity inequalities. *INFORMS Journal on Computing*, 36(4):987–1005, 2024.

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

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

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

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

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

Haoran Liu and Yu Yang. Imperfect is perfect: Learned column prefilling for branch-price-and-cut. *arXiv preprint arXiv:2610.21001*, 2026.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*, 2019.

Jens Lysgaard. Cvrpsep: A package of separation routines for the capacitated vehicle routing problem. 2003.

Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical programming*, 100(2):423–445, 2004.

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

Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning–based column selection for column generation. *Transportation Science*, 55(4):815–831, 2021.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

Konstantin Pavlikov, Niels Christian Petersen, and Jon Lilholt Sørensen. Exact separation of the rounded capacity inequalities for the capacitated vehicle routing problem. *Networks*, 83(1): 197–209, 2024.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523, 2020.

Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa, and Thibaut Vidal. 10,000 optimal CVRP solutions for testing machine learning based heuristics. In *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)*, 2022.

Yoonju Sim, Hyeonah Kim, and Changhyun Kwon. Test-time search in neural graph coarsening procedures for the capacitated vehicle routing problem. *arXiv preprint arXiv:2510.00958*, 2025.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR, 2020.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in neural information processing systems*, 34:13266–13279, 2021.

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

Yu Yang, Natashia Boland, and Martin Savelsbergh. Multivariable branching: A 0-1 knapsack problem case study. *INFORMS Journal on Computing*, 33(4):1354–1367, 2021.

Yu Yang, Natashia Boland, Bistra Dilkina, and Martin Savelsbergh. Learning generalized strong branching for set covering, set packing, and 0–1 knapsack problems. *European Journal of Operational Research*, 301(3):828–840, 2022.

Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 20284–20292, 2024.

Zhengzhong You and Yu Yang. Routeopt: An open-source modular exact solver for vehicle routing problems. *Available at SSRN 5314242*, 2025.

Zhengzhong You, Bo Tang, Haoran Liu, and Baichuan Mo. Learned pairwise deep dual-optimal inequalities for stabilizing column generation. *arXiv preprint arXiv:2607.13373*, 2026a.

Zhengzhong You, Yu Yang, Xinshang Wang, and Wotao Yin. Two-stage learning to branch in branch-price-and-cut algorithms for solving vehicle routing problems exactly. *Operations Research*, 2026b.

Xiangyi Zhang, Lu Chen, Michel Gendreau, and André Langevin. Learning-based branch-and-price algorithms for the vehicle routing problem with time windows and two-dimensional loading constraints. *INFORMS Journal on Computing*, 34(3):1419–1436, 2022.

Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 37:6081–6125, 2024.

Appendices

Contents

A Additional Methodological and Theoretical Details of SAG-Sep **14**

A.1 Transformer Layer 14

A.2 Lipschitz Continuity of the Attention Module 16

A.3 Aggressive Sampling via Conditioned Probing 17

A.4 Proof of the Resolution Error Bound 19

A.5 Proof of Sparse Augmentation Complexity 19

B Exact RCC Separation Study and Onion Analysis **20**

B.1 Three Exact RCC Separation Formulations 20

B.2 Diversity and Repetition Properties 21

B.3 Empirical Comparison of Exact Formulations 22

B.4 Onion Structure in Exact Labels and Learned Scores 23

C Implementation and Training Details **24**

C.1 Training Procedure 24

C.2 Loss Functions 25

C.3 Shared Edge Encoding and Inference Memory 25

C.4 Hyperparameter Selection 26

D Additional Discussions and Results **28**

D.1 NeuralSEP$_1$ Reproduction and One-Shot Base Model 28

D.2 Fixed-Round Comparisons at N=1000 28

D.3 fp32 vs. TF32 Precision Sensitivity 29

D.4 Serial SAG-Sep + CVRPSEP Hybrid 29

D.5 Detailed K-hop Ablation 30

D.6 Paired Per-Instance Comparisons 30

D.7 Generalization Beyond the Benchmark 30

E Code Release and Scope Beyond the Root Node **31**

E.1 The Released Engine Run to Closure Against CVRPSEP 31

E.2 Preliminary Complete Branch-and-Cut 32

E.3 Capacity and Rounding Parts of Exact RCC Violations 32

E.4 Transfer Across Problem Classes: The Released Examples 33

A Additional Methodological and Theoretical Details of SAG-Sep

A.1 Transformer Layer

The transformer layer in SAG-Sep follows the standard encoder block [Vaswani et al., 2017] and consists of a multi-head attention (MHA) module followed by a feed-forward network (FFN). The implementation uses residual connections and layer normalization around these two components. This subsection focuses on the attention module, where edge features provide an additive bias to the scaled dot-product logits; see Figure 2.

Attention layer. Let $X \in \mathbb{R}^{n\times d}$ denote the input node embeddings of the current layer, where $n = |V|$ and each row, denoted $x_i^\top$, corresponds to node i. The query, key, and value projections are

$$\mathbf{Q} = XW_Q, \quad \mathbf{K} = XW_K, \quad \mathbf{U} = XW_U, \tag{1}$$

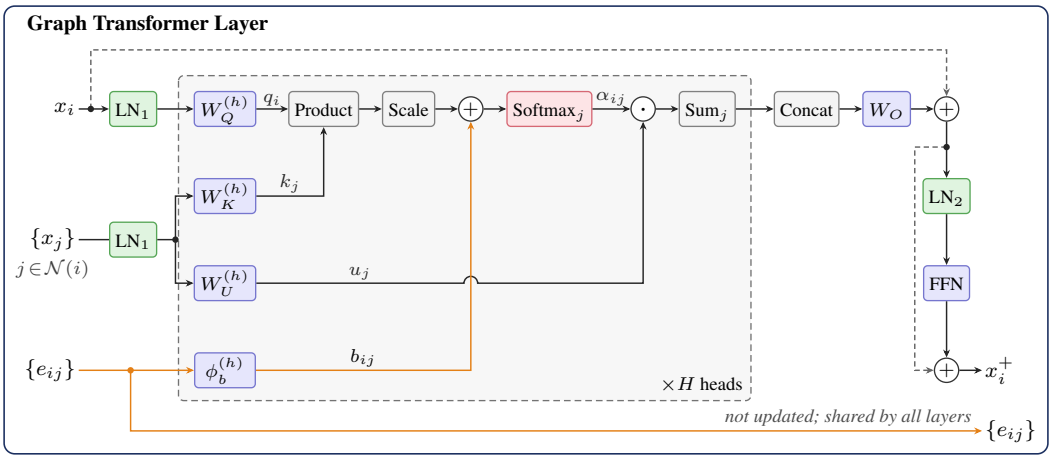


Figure 2: Graph Transformer layer with edge-biased multi-head attention (pre-normalization). Edge embeddings are encoded once, shared by all layers, and enter each head only through the scalar attention bias b_{ij}.

where $W_Q, W_K \in \mathbb{R}^{d\times d_k}$ and $W_U \in \mathbb{R}^{d\times d_v}$ are learnable matrices. We write $q_i^\top, k_i^\top, u_i^\top$ for the rows of $\mathbf{Q}, \mathbf{K}, \mathbf{U}$.

Edge-biased sparse attention. Let $\widetilde{G} = (V, \widetilde{E})$ be the LP-induced augmented graph and $\mathcal{N}(i) = \{j : \{i,j\} \in \widetilde{E}\}$ be the neighbor set of node i. Attention is asymmetric (the weights $i \to j$ and $j \to i$ are computed independently), and we do not add self-loop attention edges; the residual connection at the layer level carries each node's previous representation. Every vertex has positive LP degree and the depot is joined to every customer, so no neighborhood $\mathcal{N}(i)$ is empty. Each undirected edge $\{i,j\} \in \widetilde{E}$ has an encoded edge embedding $e_{ij} = e_{ji}$ obtained from the input feature e_{ij}^{input} in Section 3.2 by a learnable edge encoder. From this embedding we compute a scalar attention bias

$$b_{ij} = \phi_b(e_{ij}), \qquad \{i,j\} \in \widetilde{E}, \tag{2}$$

where $\phi_b(\cdot)$ is a learnable layer-specific linear map. We restrict attention to $\widetilde{E}$ by normalizing over $\mathcal{N}(i)$ (equivalently, using a mask with $-\infty$ on non-edges). In SAG-Sep, edge embeddings are used to generate layer-specific attention biases and are not updated by the node-attention block. The attention weights are then

$$\alpha_{ij} = \frac{\exp\left(\frac{q_i^\top k_j}{\sqrt{d_k}} + b_{ij}\right)}{\sum_{t\in\mathcal{N}(i)} \exp\left(\frac{q_i^\top k_t}{\sqrt{d_k}} + b_{it}\right)}, \qquad j \in \mathcal{N}(i), \tag{3}$$

and the output is

$$z_i = \sum_{j\in\mathcal{N}(i)} \alpha_{ij}\, u_j, \qquad Z = \text{Attention}(X) \in \mathbb{R}^{n\times d_v}. \tag{4}$$

In matrix form, letting $B \in \mathbb{R}^{n\times n}$ collect the edge biases with $B_{ij} = \phi_b(e_{ij})$ for $j \in \mathcal{N}(i)$ and $B_{ij} = -\infty$ otherwise, we write

$$\text{Attention}(X) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + B\right)\mathbf{U}, \tag{5}$$

where $\text{Softmax}(\cdot)$ is applied row-wise.

Multi-head attention. For H heads, we use head-specific projections $W_Q^{(h)}, W_K^{(h)}, W_U^{(h)}$ and head-specific edge-bias maps $\phi_b^{(h)} : e_{ij} \mapsto b_{ij}^{(h)}$. Each head produces $Z^{(h)} = \text{Attention}^{(h)}(X)$ via (1)–(5), and the MHA output is

$$\text{MHA}(X) = \text{Concat}\left(Z^{(1)}, \ldots, Z^{(H)}\right) W_O, \tag{6}$$

where $W_O \in \mathbb{R}^{(Hd_v)\times d}$ is a learnable projection.

Residual, normalization, and FFN. The full layer uses a pre-normalization transformer block:

$$\bar{X} = \mathrm{LN}_1(X), \tag{7}$$

$$Y = X + \mathrm{Dropout}\big(\mathrm{MHA}(\bar{X})\big), \tag{8}$$

$$X^+ = Y + \mathrm{Dropout}(\mathrm{FFN}(\mathrm{LN}_2(Y))), \tag{9}$$

where FFN is a two-layer feed-forward network applied row-wise. The layer output is X^+, while the edge embeddings remain shared auxiliary features for computing attention biases.

A.2 Lipschitz Continuity of the Attention Module

We derive a Lipschitz bound for the edge-biased sparse attention mechanism with respect to the node features X. This module-level stability result supports the local stability assumption used in the conditioned-probing analysis of Appendix A.3, which guided the sampling budget of the aggressive-sampling baseline; the Onion-Sweep decoder does not rely on it, so we only sketch the argument. We use the spectral norm $\|\cdot\|_2$ for matrices and the Euclidean norm for vectors.

Assumption A.1 (Bounded Input)**.** We assume the input node features lie in a bounded set, i.e., $\|X\|_2 \le C_X$ for some constant $C_X > 0$. This local boundedness assumption is standard in self-attention analyses and is stabilized in practice by layer normalization and finite inference activations.

Lemma A.2 (Lipschitz Bound for a Single Attention Head)**.** *Under Assumption A.1, consider the single-head attention function $f(X) : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d_v}$ defined in Eq.* (5)*. Let L_σ be a Lipschitz constant of the row-wise Softmax map on the bounded logit set induced by $\|X\|_2 \le C_X$, measured in spectral norm. Let*

$$C_S := \sup_X \|S(X)\|_2, \tag{10}$$

where $S(X) = \mathrm{Softmax}(A(X))$ is the masked attention matrix and the supremum is taken over $\|X\|_2 \le C_X$. For masked row-softmax attention on $\widetilde{G}$, C_S is finite; for example, $C_S \le \sqrt{\Delta_{\widetilde{G}}}$, where

$$\Delta_{\widetilde{G}} := \max_{j\in V} \big|\{i \in V : j \in \mathcal{N}(i)\}\big| \tag{11}$$

is the maximum column sparsity of the attention mask. When the mask is symmetric, $\Delta_{\widetilde{G}}$ is the maximum degree of $\widetilde{G}$, and in all cases $C_S \le \sqrt{n}$. The Lipschitz constant of f with respect to X, denoted as $\mathcal{L}_{\mathrm{Attn}}$, is bounded by:

$$\mathcal{L}_{\mathrm{Attn}} \le \|W_U\|_2 \left(C_S + \frac{2L_\sigma C_X^2}{\sqrt{d_k}} \|W_Q\|_2 \|W_K\|_2 \right). \tag{12}$$

Proof sketch. Write $A(X) = \frac{1}{\sqrt{d_k}}(XW_Q)(XW_K)^\top + B$ for the pre-softmax logits, $S(X) = \mathrm{Softmax}(A(X))$, and $Z = f(X) = S(X)XW_U$. For a perturbation $X \to X + \Delta$, the product rule gives $\delta Z = (\delta S)XW_U + S(\delta X)W_U$. The second term is bounded by $C_S\|\delta X\|_2\|W_U\|_2$. For the first term, the bias B depends only on edge features, so $\delta B = 0$ and $\|\delta A\|_2 \le \frac{2\|X\|_2}{\sqrt{d_k}}\|W_Q\|_2\|W_K\|_2\|\delta X\|_2$; the row-wise Softmax Jacobian bound then yields $\|\delta S\|_2 \le L_\sigma\|\delta A\|_2$. Combining the two terms bounds the operator norm of the Jacobian uniformly over $\{X : \|X\|_2 \le C_X\}$, and integrating this bound along the line segment between any two points of this set gives the stated Lipschitz constant. □

Extension to multi-head attention. For the MHA module defined in Eq. (6), the output is a concatenation projected by W_O. Let $\mathcal{L}^{(h)}$ be the Lipschitz constant of the h-th head derived in Lemma A.2. Using $\|[A_1, \ldots, A_H]\|_2 \le \sqrt{\sum_{h=1}^H \|A_h\|_2^2}$ for block concatenation, the total Lipschitz constant $\mathcal{L}_{\mathrm{MHA}}$ is:

$$\mathcal{L}_{\mathrm{MHA}} \le \|W_O\|_2 \cdot \sqrt{\sum_{h=1}^{H} \big(\mathcal{L}^{(h)}\big)^2}. \tag{13}$$

This derivation highlights that the stability of the attention module depends quadratically on the magnitude of node features C_X, multiplicatively on the spectral norms of the weight matrices, and on the sparse attention pattern through C_S.

A.3 Aggressive Sampling via Conditioned Probing

Aggressive sampling increases the number of conditioned replicas evaluated at test time. Instead of restricting SAG-Sep to the base grid $\{m/\max(M-1,1)\}_{m=0}^{M-1}$, it samples a denser set $\mathcal{W} = \{w_j\}_{j=1}^{J^\star} \subset [0,1]$ and constructs one conditioned graph for each w_j. All replicas are evaluated in a batch, and each probability vector is thresholded to obtain

$$\hat{S}(w_j) = \{\, i \in V^+ : p_i(w_j) \geq \tau \,\}. \tag{14}$$

The induced RCCs are then evaluated under the current LP solution, deduplicated, and ranked by violation. This procedure can discover useful cuts missed by the base grid, but it increases both inference workload and the cost of subsequent LP re-optimization. The concrete size-adaptive grid rule used in our ablation is given in Appendix C.4.

Intuitively, the base grid observes only a small number of hard-thresholded subsets from a continuous conditioning space. Between two adjacent conditioning values, many replicas remain identical if all node probabilities stay safely away from the threshold; new subsets appear only when one or more boundary nodes cross it. Aggressive sampling is therefore most useful when these unstable regions occupy a non-negligible fraction of the conditioning interval, and it naturally saturates once the grid is finer than the model's effective margin scale. This perspective also motivates the Onion-Sweep decoder in Section 3.3, which replaces the cutoff by a violation-guided selection along the score ranking instead of evaluating a much denser conditioning grid.

The benefit of dense conditioning can be understood through local stability. Appendix A.2 provides a module-level Lipschitz bound for the attention block; here we use the corresponding local stability assumption for the scalar conditioning channel. Let R be the grid resolution, with grid points $w_r = r/R$ and intervals $I_r = [w_r, w_{r+1})$ for $r = 0, \ldots, R-1$. Assume the predictor is Lipschitz continuous in the conditioning value: for all nodes i and all $w, w' \in [0,1)$,

$$|p_i(w) - p_i(w')| \leq L|w - w'|. \tag{15}$$

Define the safety margin at w_r as

$$\gamma(w_r) = \min_{i \in V^+} |p_i(w_r) - \tau|. \tag{16}$$

If $\gamma(w_r) > L/R$, the thresholded subset is stable throughout I_r; otherwise, the interval may contain a threshold crossing. Let

$$\mathcal{U} = \bigcup_{r:\gamma(w_r) \leq L/R} I_r \tag{17}$$

denote the unstable region.

Proposition A.3 (Resolution Error Bound under Marginal Anti-concentration)**.** *Assume there exists a function $q(\cdot)$ such that, for every node $i \in V^+$, every conditioning value $w \in [0,1)$, and every $\varepsilon > 0$,*

$$\Pr(|p_i(w) - \tau| \leq \varepsilon) \leq q(\varepsilon), \tag{18}$$

where the probability is over the distribution of LP states/instances and any inference randomness. The expected Lebesgue measure of the unstable region satisfies

$$\mathbb{E}[\mathrm{vol}(\mathcal{U})] \leq |V^+|\, q(L/R). \tag{19}$$

This bound controls the measure of the conditioning interval in which thresholded subsets can change; it does not bound missed cuts, but it explains why dense conditioning saturates rather than improving indefinitely. Increasing the grid resolution reduces the perturbation scale L/R, so once R exceeds the model's effective margin scale, the only intervals that can still produce new thresholded subsets are those containing nodes extremely close to the decision boundary. When the model has little probability mass near the threshold, dense probing quickly reaches diminishing returns rather than producing an unbounded number of useful new cuts. This is also why Onion-Sweep decodes the violation profile of the score ranking directly instead of relying on ever-denser conditioning grids.

Empirical check of marginal anti-concentration. The proposition is informative only if $q(\varepsilon) \to 0$ as $\varepsilon \to 0$. We check this empirically on the trained SAG-Sep predictor, pooling predictions over nodes and conditioning levels (the proposition assumes the bound for each of them). Using 442

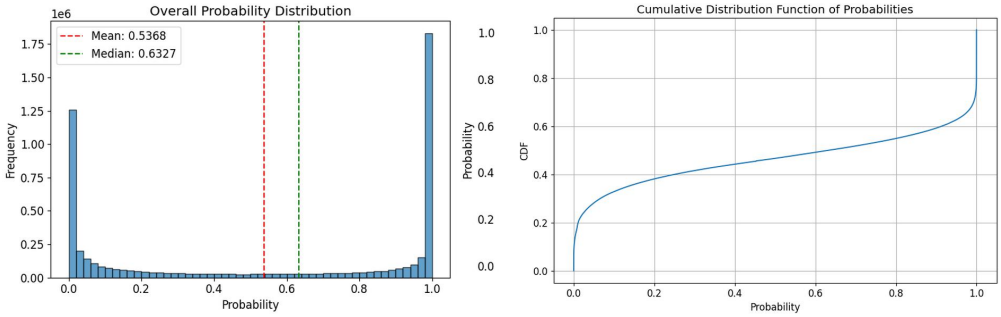


Figure 3: Empirical marginal of $p_i(w)$ on $N = 300$ root-relaxation LP states (approximately 6×10^6 post-sigmoid predictions). Left: histogram is sharply bimodal at 0 and 1 with low mass near the threshold $\tau = 0.5$. Right: the CDF is shallow on the central region, giving $\Pr(|p_i(w) - \tau| \le \varepsilon) \lesssim 0.6\,\varepsilon$ for $\varepsilon \le 0.1$ and supporting $q(\varepsilon) = O(\varepsilon)$.

random LP states drawn from $N = 300$ root relaxations, we collect roughly 6×10^6 post-sigmoid predictions $p_i(w)$ across nodes, conditioning levels, and instances; Figure 3 reports the resulting marginal distribution and CDF. Two observations follow. First, the predictor concentrates its mass near 0 and 1 with sharply lower density in the central region around τ, and this bimodal pattern is consistent across instances rather than driven by a few outliers (Figure 4). Reading the empirical CDF, $\Pr(|p_i(w) - \tau| \le \varepsilon) \lesssim 0.6\,\varepsilon$ for $\varepsilon \le 0.1$, so $q(\varepsilon)$ scales linearly with a small constant in the relevant regime. Second, the same bimodal structure explains why the hard-threshold readout is already a reasonable decoder and why its remaining failures concentrate on the few nodes near τ; Onion-Sweep removes this dependence by selecting prefixes of the score ranking by violation rather than by a cutoff.

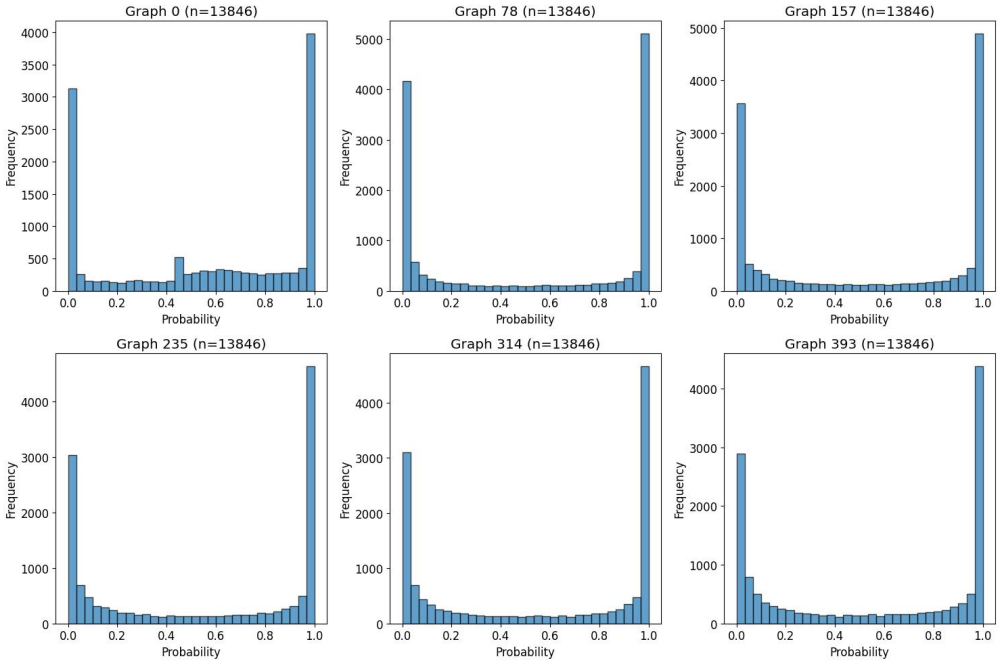


Figure 4: Per-instance marginals of $p_i(w)$ on six randomly sampled $N = 300$ root-relaxation LP states. The bimodal pattern, and the resulting low density near $\tau = 0.5$, is structural rather than instance-specific; isolated secondary modes (e.g. Graph 0 near $p = 0.45$) do not violate the linear anti-concentration rate read from the aggregate CDF.

A.4 Proof of the Resolution Error Bound

Proof of Proposition A.3. Let $\delta = 1/R$ and let U_r be the indicator that interval I_r is unstable:

$$U_r = \mathbb{I}\left(\gamma(w_r) \le L\delta\right) = \begin{cases} 1 & \text{if } \min_{i \in V^+} |p_i(w_r) - \tau| \le L\delta, \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Since the intervals are disjoint and each has measure δ,

$$\mathbb{E}[\mathrm{vol}(\mathcal{U})] = \delta \sum_{r=0}^{R-1} \Pr(U_r = 1). \tag{21}$$

For each grid point w_r, the unstable event is a union over nodes. By the union bound,

$$\Pr(U_r = 1) = \Pr\left(\bigcup_{i \in V^+} \{|p_i(w_r) - \tau| \le L\delta\} \right) \tag{22}$$

$$\le \sum_{i \in V^+} \Pr\left(|p_i(w_r) - \tau| \le L\delta\right) \le |V^+|\, q(L/R), \tag{23}$$

where the last inequality uses the marginal anti-concentration assumption. Substituting into (21) gives

$$\mathbb{E}[\mathrm{vol}(\mathcal{U})] \le \delta \sum_{r=0}^{R-1} |V^+|\, q(L/R) = |V^+|\, q(L/R). \tag{24}$$

□

A.5 Proof of Sparse Augmentation Complexity

Proof of Proposition 3.1. Let $\lambda_N = \rho(N-1)$. For two distinct fixed nodes $u, v \in V$, let $d_G(u,v)$ denote their shortest-path distance in G_{LP}. By symmetry,

$$\mathbb{E}[|E_{\mathrm{LP}}|] = \binom{N}{2}\rho, \tag{25}$$

$$\mathbb{E}[|\widetilde{E}|] = \binom{N}{2}\mathbb{P}(1 \le d_G(u,v) \le K). \tag{26}$$

Therefore

$$\mathcal{R}_N := \frac{\mathbb{E}[|\widetilde{E}|]}{\mathbb{E}[|E_{\mathrm{LP}}|]} = \frac{\mathbb{P}(1 \le d_G(u,v) \le K)}{\rho}. \tag{27}$$

Let X_t be the number of simple paths of length t between u and v, and let $X_{\le K} := \sum_{t=1}^{K} X_t$. Then $\{1 \le d_G(u,v) \le K\}$ is exactly the event $\{X_{\le K} \ge 1\}$. For a fixed t, a simple length-t path has $t-1$ ordered intermediate nodes, so

$$M_t = \frac{(N-2)!}{(N-t-1)!} = (N-2)(N-3)\cdots(N-t) \tag{28}$$

such paths are possible, with the product interpreted as 1 when $t = 1$. By independence of edge events under $\mathcal{G}(N,\rho)$, each such path appears with probability ρ^t. Hence

$$\mu_N := \mathbb{E}[X_{\le K}] = \sum_{t=1}^{K} M_t \rho^t. \tag{29}$$

Since K is fixed and $\lambda_N^K = o(N)$, we have $\mu_N = o(1)$. A Bonferroni bound gives

$$\mu_N - \sum_{P<Q} \mathbb{P}(P \text{ and } Q \text{ both occur}) \le \mathbb{P}(X_{\le K} \ge 1) \le \mu_N, \tag{30}$$

where $P < Q$ ranges over unordered pairs of simple u–v paths of length at most K. The pair intersection term is $o(\mu_N)$ in the same sparse local regime: edge-disjoint path pairs contribute

$O(\mu_N^2)$, while distinct path pairs sharing edges form a connected subgraph with a cycle. For each fixed overlap type, a union subgraph with e edges has at most $e-2$ free internal vertices, so its contribution is $O(\lambda_N^e/N^2)$. Summing over the finitely many overlap types with $e \le 2K$ gives a total shared-edge contribution of order $O\Big(N^{-2}(\sum_{t=1}^{K}\lambda_N^t)^2\Big) = O(\mu_N^2)$. Since $\mu_N = o(1)$, the full pair intersection term is $o(\mu_N)$. Thus

$$\mathbb{P}(1 \le d_G(u,v) \le K) = \mathbb{P}(X_{\le K} \ge 1) = (1+o(1))\mu_N. \tag{31}$$

Substituting this estimate into (27) yields

$$\begin{aligned}\mathcal{R}_N &= (1+o(1))\sum_{t=1}^{K} M_t \rho^{t-1} \\ &= (1+o(1))\sum_{t=1}^{K} \lambda_N^{t-1} \prod_{r=2}^{t} \frac{N-r}{N-1}.\end{aligned} \tag{32}$$

The empty product for $t=1$ equals 1, and for every fixed t the product converges to 1. Therefore

$$\mathcal{R}_N = (1+o(1))\sum_{t=1}^{K}\lambda_N^{t-1} = (1+o(1))\sum_{k=0}^{K-1}\lambda_N^k. \tag{33}$$

If $\lambda_N \to \lambda$, this gives $\lim_{N\to\infty}\mathcal{R}_N = \sum_{k=0}^{K-1}\lambda^k$. □

B Exact RCC Separation Study and Onion Analysis

The EXACT baseline in Table 1 uses exact separation for rounded capacity constraints (RCCs). This appendix spells out three closely related exact formulations and explains why the upper-bounded, demand-banded variant is more useful for generating diverse cuts and for studying the nested structure that motivates Onion-Sweep.

Notation. Let $G=(V,E)$ be the complete graph with depot 0 and customers $V^+ = V\setminus\{0\}$. Let $\bar{x} \in \mathbb{R}_+^E$ (customer edges in $[0,1]$, depot edges in $[0,2]$) denote the current fractional LP solution, $d_i \in \mathbb{Z}_{>0}$ the demand of customer i, and C the vehicle capacity. For $S \subseteq V^+$, define $D(S) = \sum_{i\in S} d_i$ and $\bar{x}(\delta(S)) = \sum_{\{i,j\}\in E:\,|\{i,j\}\cap S|=1} \bar{x}_{ij}$. The RCC associated with S is

$$\bar{x}(\delta(S)) \ge 2\left\lceil \frac{D(S)}{C} \right\rceil, \tag{34}$$

with violation $\nu(S) = 2\lceil D(S)/C\rceil - \bar{x}(\delta(S))$. All formulations below use binary variables y_i to encode $S = \{i \in V^+ : y_i = 1\}$ and nonnegative variables μ_{ij} to linearize the cut indicator.

B.1 Three Exact RCC Separation Formulations

Fukasawa lower-bound formulation. The classical exact separator of Fukasawa et al. [2006] enumerates $m \in \{0,\dots,\lceil D(V^+)/C\rceil - 1\}$ and solves

$$\min_{y,\mu}\ z_F(m) = \sum_{\{i,j\}\in E} \bar{x}_{ij}\,\mu_{ij} \tag{35}$$

$$\text{s.t. } \mu_{ij} \ge y_i - y_j, \qquad \forall \{i,j\}\in E, \tag{36}$$

$$\mu_{ij} \ge y_j - y_i, \qquad \forall \{i,j\}\in E, \tag{37}$$

$$mC + 1 \le \sum_{i\in V^+} d_i y_i, \tag{38}$$

$$y_0 = 0, \tag{39}$$

$$y_i \in \{0,1\}, \qquad \forall i \in V^+, \tag{40}$$

$$\mu_{ij} \ge 0, \qquad \forall \{i,j\}\in E. \tag{41}$$

The returned subset is accepted as violated whenever $z_F(m) < 2(m+1)$. This formulation is efficient and standard, but the demand constraint only lower-bounds the required vehicle count.

Direct violation-maximization variant. A tempting alternative is to keep the same lower-bound feasible region but optimize the true RCC violation directly:

$$\max_{y,\mu} \; z_V(m) = 2\left\lceil \frac{\sum_{i\in V^+} d_i y_i}{C} \right\rceil - \sum_{\{i,j\}\in E} \bar{x}_{ij}\,\mu_{ij} \tag{42}$$

$$\text{s.t.} \;\; \mu_{ij} \geq y_i - y_j, \qquad \forall \{i,j\} \in E, \tag{43}$$

$$\mu_{ij} \geq y_j - y_i, \qquad \forall \{i,j\} \in E, \tag{44}$$

$$mC + 1 \leq \sum_{i\in V^+} d_i y_i, \tag{45}$$

$$y_0 = 0, \tag{46}$$

$$y_i \in \{0,1\}, \qquad \forall i \in V^+, \tag{47}$$

$$\mu_{ij} \geq 0, \qquad \forall \{i,j\} \in E. \tag{48}$$

The ceiling term can be linearized with an auxiliary integer vehicle-count variable, but the compact form makes the behavior clearer: this model optimizes the actual RCC violation while retaining the same lower-bound demand constraint. In practice, this version has a severe diversity problem, because a high-demand high-violation subset remains feasible for many smaller m values and can dominate the entire enumeration loop.

Demand-banded formulation. Our exact-label generation use a demand-banded variant. For each m, we solve

$$\min_{y,\mu} \; z_B(m) = \sum_{\{i,j\}\in E} \bar{x}_{ij}\,\mu_{ij} \tag{49}$$

$$\text{s.t.} \;\; \mu_{ij} \geq y_i - y_j, \qquad \forall \{i,j\} \in E, \tag{50}$$

$$\mu_{ij} \geq y_j - y_i, \qquad \forall \{i,j\} \in E, \tag{51}$$

$$mC + 1 \leq \sum_{i\in V^+} d_i y_i \leq (m+1)C, \tag{52}$$

$$y_0 = 0, \tag{53}$$

$$y_i \in \{0,1\}, \qquad \forall i \in V^+, \tag{54}$$

$$\mu_{ij} \geq 0, \qquad \forall \{i,j\} \in E. \tag{55}$$

Within this band, minimizing $z_B(m)$ is equivalent to maximizing $2(m+1) - \bar{x}(\delta(S))$. The only change from (35)–(41) is the additional upper bound, but it changes the structure of the returned cut set substantially.

B.2 Diversity and Repetition Properties

Lemma B.1 (Cut-value equivalence)**.** *For any fixed binary vector y, an optimal choice of μ in* (36)–(41) *satisfies $\mu_{ij} = |y_i - y_j|$ for all $\{i,j\} \in E$, and thus $\sum_{\{i,j\}\in E} \bar{x}_{ij}\mu_{ij} = \bar{x}(\delta(S))$ for $S = \{i \in V^+ : y_i = 1\}$.*

Proof. Given binary y, constraints (36)–(37) and $\mu_{ij} \geq 0$ imply $\mu_{ij} \geq |y_i - y_j|$. Choosing $\mu_{ij} = |y_i - y_j|$ is feasible and, since $\bar{x}_{ij} \geq 0$, optimal (and forced on every edge with $\bar{x}_{ij} > 0$). Summing over edges gives $\bar{x}(\delta(S))$. □

Proposition B.2 (Repeated feasible sets under lower-bound formulations)**.** *Fix any subset $S \subseteq V^+$ with demand $D(S)$. Under any formulation that uses only the lower-bound demand constraint* (38)*, the same S is feasible for every integer m satisfying*

$$m \leq \left\lfloor \frac{D(S) - 1}{C} \right\rfloor. \tag{56}$$

Consequently, both the Fukasawa formulation and the direct violation-maximizing variant can return duplicated subsets across multiple m values.

Table 4: Comparison of two usable exact RCC separation formulations: classical lower-bound Fukasawa vs. demand-banded diverse separation.

Method	Size	Avg. Gap (↓)	Avg. LB (↑)	Avg. Runtime	Avg. Iter.	Avg. Cuts	Count
Exact (banded, with UB)	50	**1.97%**	9363.38	54.91	59.20	227.70	10
	75	2.76%	13357.85	198.81	79.70	511.10	10
	100	4.52%	15940.77	654.56	125.20	981.00	10
	200	6.36%	21354.73	3575.54	169.40	2231.10	10
	300	**11.65%**	30204.47	3654.00	49.60	1210.40	10
	400	**18.69%**	40430.81	3715.66	23.20	821.80	10
	500	**24.41%**	44997.84	3767.76	14.20	573.10	10
	750	**40.59%**	54228.71	4059.01	4.20	278.50	10
	1000	**50.08%**	46757.73	4520.46	2.30	169.00	10
Exact (classic Fukasawa)	50	1.98%	9362.89	55.45	58.00	222.10	10
	75	2.76%	13357.85	203.97	78.60	504.50	10
	100	4.52%	15940.56	671.00	122.30	979.90	10
	200	**6.35%**	21358.92	3593.72	169.40	2229.90	10
	300	11.98%	30047.31	3651.60	49.10	1187.20	10
	400	18.89%	40364.69	3698.08	24.30	852.60	10
	500	24.87%	44489.87	3718.09	13.80	556.00	10
	750	41.00%	53653.12	4252.43	4.30	283.50	10
	1000	50.61%	46304.41	4624.86	2.30	169.00	10

Proof. Feasibility with respect to m is determined by $mC + 1 \leq D(S)$, which is equivalent to $m \leq (D(S) - 1)/C$. Taking the floor gives the claim. □

By Proposition B.2, this repetition affects both lower-bound formulations. It is mild but still visible in the Fukasawa formulation: the objective minimizes only the crossing value, so a low-crossing high-demand set can remain attractive across many lower-bound subproblems. It is much more severe for the direct violation-maximization variant, whose objective explicitly rewards large rounded demand; empirically this can collapse the loop onto the same high-demand subset and makes the formulation nearly unusable as a diverse cut generator.

Proposition B.3 (One demand band per subset)**.** *Under the band constraint* (52)*, any subset S with demand $D(S)$ is feasible for at most one integer m, namely*

$$m = \left\lceil \frac{D(S)}{C} \right\rceil - 1. \tag{57}$$

Thus a single enumeration over m cannot repeatedly admit the same subset across multiple subproblems.

Proof. Since demands are integral, (52) is equivalent to $mC < D(S) \leq (m+1)C$. Dividing by C gives $m < D(S)/C \leq m + 1$, which is equivalent to $m = \lceil D(S)/C \rceil - 1$. □

By Proposition B.3, the demand-banded formulation provides diversity by construction. It does not merely change the acceptance test for violated RCCs; it changes which subsets are available to each exact subproblem, yielding one representative cut per demand band and preventing repeated subsets within a separation round.

B.3 Empirical Comparison of Exact Formulations

Table 4 compares the classical lower-bound Fukasawa formulation with the demand-banded variant. We omit the direct violation-maximization variant from the table because its repeated-subset behavior makes it ineffective as a multi-cut separator despite optimizing the true RCC violation objective.

The banded formulation is consistently competitive and becomes stronger on the larger instances where diversity matters most. At $N = 750$ and $N = 1000$, it improves the final average gap from 41.00% to 40.59% and from 50.61% to 50.08%, respectively, while also reducing runtime. This supports the interpretation above: restricting each exact solve to a demand band avoids wasting the cut budget on duplicated subsets and yields a more useful sequence of exact cuts.

Table 5: Exact onion nestedness and alignment between exact membership frequency and frozen SAG-Sep probabilities on random evaluation LP states (each MILP solved to a relative optimality gap of 10^{-4}, on the first 20 LP states of the exact-separation root loop of each of the ten instances per size).

Size	LP states	Avg. K	Contain.	M-nest	$\rho(\bar{p}, f)$	$\rho_{\text{best}}(p^{(j)}, f)$
$N = 50$	200	5.2	0.894	0.846	0.838	0.870
$N = 75$	200	8.4	0.881	0.811	0.864	0.883
$N = 100$	200	11.2	0.917	0.831	0.893	0.910
$N = 200$	200	15.8	0.929	0.806	0.844	0.862
$N = 300$	200	27.3	0.931	0.790	0.847	0.860
$N = 400$	200	39.0	0.947	0.792	0.853	0.863
$N = 500$	200	48.4	0.945	0.777	0.838	0.849
$N = 750$	200	72.7	0.951	0.765	0.837	0.850
$N = 1000$	200	79.1	0.948	0.737	0.808	0.828

B.4 Onion Structure in Exact Labels and Learned Scores

The demand-banded formulation gives one non-duplicated subset per vehicle-count band, making it a clean object for studying the structure of exact RCC labels. For an LP state, let

$$\mathcal{S}_{\text{ex}} = \{S_{m_1}, \ldots, S_{m_K}\}, \qquad m_1 < \cdots < m_K, \tag{58}$$

be the exact subsets returned across demand bands. We report two complementary nesting measures. *Containment* averages the overlap ratio over all unordered band pairs,

$$\text{Containment}(\mathcal{S}_{\text{ex}}) = \frac{2}{K(K-1)} \sum_{1 \le r < s \le K} \frac{|S_{m_r} \cap S_{m_s}|}{\min\{|S_{m_r}|, |S_{m_s}|\}}, \tag{59}$$

and captures whether smaller exact subsets are largely contained in larger ones across the entire band sequence. ***M-nesting*** is the strict version of the same quantity, the fraction of band pairs in which the smaller-band subset is contained in the larger-band one without exception,

$$\text{M-nesting}(\mathcal{S}_{\text{ex}}) = \frac{2}{K(K-1)} \sum_{1 \le r < s \le K} \mathbb{I}\{S_{m_r} \subseteq S_{m_s}\}. \tag{60}$$

A value of one in either metric means perfect nesting. Containment tolerates a few displaced nodes, whereas M-nesting counts a pair only if it is exactly nested, so it is the stricter measure; in Table 5 it decreases as K grows while containment stays high. Across the exact-label datasets, M-nesting is consistently high: on XML100 exact labels, grouped by route-length bucket, it stays between 0.836 and 0.868; on the random evaluation labels it stays between 0.74 and 0.85 from $N = 50$ to $N = 1000$ (Table 5), decreasing only mildly as the number of bands grows. The all-pair containment is even higher, 0.88–0.95 across the same sizes. This indicates that the high-value exact RCCs are not arbitrary unrelated subsets, but ordered "onion" layers, and that this structure persists at the headline sizes.

We next ask whether the trained SAG-Sep encoder has learned this priority structure. For each exact-labeled LP state, define the exact membership frequency

$$f_i = \frac{1}{K} \sum_{r=1}^{K} \mathbb{I}\{i \in S_{m_r}\}, \qquad i \in V^+. \tag{61}$$

In a perfectly nested family, high-frequency nodes are core onion nodes and low-frequency nodes enter only in outer layers; for the approximately nested families observed here, f serves as a priority surrogate. Running the frozen SAG-Sep checkpoint on the same LP states gives probability vectors $\{p^{(j)}\}_{j=1}^{J}$ over the vehicle-count replicas. We compare the exact frequency f with both the cross-replica mean score $\bar{p}_i = J^{-1} \sum_j p_i^{(j)}$ and the best-aligned single replica, using Spearman correlation. The correlations remain strong at every size: $\rho(\bar{p}, f)$ is between 0.84 and 0.89 through $N = 750$ and 0.81 at $N = 1000$, while the best replica reaches 0.83–0.91.

These results support the design of Onion-Sweep. The exact separator produces diverse cuts by moving through demand bands, and those cuts are largely nested. The neural encoder already assigns

higher probability to nodes that appear in more exact onion layers, but hard thresholding each replica independently discards much of this ranking information. Onion-Sweep recovers it at test time: within each replica it selects the most violated prefix of the score ranking, and across replicas it sorts nodes by the soft frequency score and forms demand-triggered prefixes, thereby converting independently predicted probabilities into a coordinated nested family of RCC candidates without additional training.

C Implementation and Training Details

This section documents the training pipeline used in Section 4.1, including data generation, feature construction, model configuration, and optimization.

C.1 Training Procedure

XML datasets. The main supervised checkpoint is trained on XML100 [Queiroga et al., 2022]. The XML100 repository provides 10,000 CVRP instances whose names follow the format `XML100_ABCD_EF`, where `ABCD` encodes depot positioning, customer positioning, demand distribution, and average route length. For each instance, we generate exact-label supervision offline at the root relaxation: solve the LP, apply exact RCC separation across vehicle-count levels, add the found cuts, and repeat until no further violated RCC is found. Each intermediate LP state yields supervised examples $(G_{\bar{x}}, m, y^\star(m))$, where $y^\star(m)$ is the exact subset returned for vehicle-count index m.

Exact-label generation at size 100 is expensive because the exact separator must be solved repeatedly until the root cutting-plane loop converges, and difficult instances can run near the one-hour cap. To study cheaper supervision, we also generate XML20, XML30, XML40, and XML50 datasets using the same XML generation taxonomy but smaller customer counts. These smaller instances are much cheaper to label: XML20 is a few seconds per instance in our labeling runs, XML30 is on the order of tens of seconds, and XML40/XML50 remain around the minute scale. They therefore provide a practical bootstrap path when full XML100 exact labeling is too costly.

Train/test split. All XML datasets use the same distribution-stratified split. For each XML distribution family, the 25 instances with suffixes 1–25 are used for training, while the remaining instances are reserved for evaluation. For model selection, we sample a compact held-out set of 2,048 graphs from the instances with suffix `26`. The split is instance-disjoint and stratified by XML family (families are shared across splits), and the small validation set keeps validation inexpensive. Unless otherwise noted, the main SAG-Sep variants share the same trained checkpoint; the training-size ablation below trains one checkpoint per XML size.

Features. For each customer node $i \in V^+$, the input feature is $[d_i/C,\ m/\max(M-1,1)]$, where $M = \lceil d(V^+)/C \rceil$ and $m \in \{0, \dots, M-1\}$ is the replica index. The depot is masked out in the loss. Each edge in the augmented LP-support graph carries the two features used in Section 3.2: the LP-flow feature e_{ij}^{flow} and the LP-support distance $\text{dist}_{\text{LP}}(i,j)$.

Training settings. Training is run on a single NVIDIA B200 GPU paired with an Intel Xeon Platinum 8570 CPU. The XML100 checkpoint takes about one day on one B200. Smaller XML datasets train substantially faster: XML50 is roughly a half-day run, while for XML20 the validation loss already shows clear overfitting within about one hour. The model and optimizer settings used for the main checkpoint are listed in Tables 6 and 7.

The architecture is described in Appendix A.1 and in Figure 2. We summarize the training-time hyperparameters in Table 6.

We use batch size 16. The optimizer and scheduler are reported in Table 7.

Training data size ablation. Table 8 evaluates how the size of exact-labeled training instances affects downstream root separation. Smaller exact-labeled datasets are weaker than XML100, as expected, but the degradation is gradual: XML40/XML50 already recover much of the XML100 performance while requiring far cheaper label generation. This supports using small exact-labeled instances as a practical supervised bootstrap.

Table 6: Model configuration for SAG-Sep training.

Block	Item	Value
Backbone	type	transformer
	#layers (L)	6
	latent dimension (d)	256
Transformer	#heads	8
	dropout	0.1
	edge features	enabled
Policy head	MLP hidden sizes	128, 64
	output activation	Identity
	output	per-node logit

Table 7: Optimizer and scheduler configuration.

Component	Item	Value
Adam	learning rate	1×10^{-3}
	weight decay	0.0
	betas	(0.9, 0.999)
	eps	1×10^{-8}
CosineAnnealingWarmRestarts	T_0 (optimizer steps)	200
	T_{mult}	1
	$\eta_{\min}$	1×10^{-5}

C.2 Loss Functions

Let a training sample be indexed by (g, m), where g denotes the LP-support graph and m is the rounding index. Let $V^+(g)$ be the customer set and let $y_i \in \{0, 1\}$ be the teacher label indicating whether customer i belongs to the optimal separated subset for this (g, m). The model outputs a logit $s_i \in \mathbb{R}$ per customer; denote $p_i = \sigma(s_i)$.

Standard binary cross-entropy (BCE). We use the depot-masked BCE averaged over customers:

$$\mathcal{L}_{\text{BCE}} \;=\; \frac{1}{|V^+(g)|} \sum_{i \in V^+(g)} \Big(- y_i \log p_i \;-\; (1 - y_i) \log(1 - p_i) \Big). \tag{62}$$

In our final configuration, we do *not* apply additional positive-class weighting; a sweep over weighting strategies did not improve downstream separation quality (Table 9).

Optional positive-weighted BCE (ablation). For completeness, we implemented a per-m bucket weighting scheme motivated by class imbalance. For a mini-batch, define the number of positive and negative labels (with depot masked) as n_+ and n_-. The positive-weighted loss is

$$\mathcal{L}_{\text{PW}} \;=\; \frac{1}{|V^+(g)|} \sum_{i \in V^+(g)} \Big(- \alpha\, y_i \log p_i \;-\; (1 - y_i) \log(1 - p_i) \Big), \qquad \alpha \;=\; \frac{n_-}{\max\{n_+, 1\}}. \tag{63}$$

When multiple m values appear in a batch, we apply (63) within each m bucket and average across buckets. We also support a mixture between instance-equal averaging and sample-proportional averaging; empirically, these variants did not outperform (62) in our setting.

C.3 Shared Edge Encoding and Inference Memory

At inference time, SAG-Sep evaluates M vehicle-count replicas for the same LP solution. These replicas differ only in their node-level conditioning value $m/\max(M-1, 1)$; the augmented edge set $\widetilde{E}$ and all edge input features are identical. A standard batched graph implementation materializes M disjoint copies of the encoded edge tensor and the per-edge attention tensors, requiring $O(M|\widetilde{E}|D)$ edge-side activation memory for hidden dimension D.

Our shared-edge implementation stores one edge index tensor, one encoded edge tensor in $\mathbb{R}^{|\widetilde{E}| \times D}$, and a per-layer edge-bias tensor in $\mathbb{R}^{|\widetilde{E}| \times H}$, where H is the number of attention heads. These edge

Table 8: Root-gap ablation by exact-labeled training instance size. Values are average root gaps (%, lower is better) on the random benchmark. Run at an earlier development stage; absolute gaps differ from Table 1, and comparisons are within the table.

Training data	$N = 50$–100	$N = 200$–400	$N = 500$–1000
XML20	8.79	14.03	21.91
XML30	7.25	12.75	21.26
XML40	6.83	10.65	18.76
XML50	6.05	10.06	18.10
XML100	**5.88**	**9.43**	**17.35**

Table 9: Ablation on positive-weight loss: mean optimality gap (%) by instance size (lower is better). Run at an earlier development stage; absolute gaps differ from Table 1, and comparisons are within the table.

Setting	$N = 50$	$N = 100$	$N = 200$	$N = 300$	$N = 400$
Sample loss	9.47	10.86	13.22	13.39	14.85
Mix	8.55	10.74	12.38	12.72	12.98
Instance loss	9.88	11.39	13.74	14.22	14.57
No positive weight	**7.98**	**9.78**	**11.91**	**12.33**	**12.79**

quantities are reused across the replica dimension, while only the node tensor is stacked over the M conditioning values. This reduces the edge-side memory to $O(|\widetilde{E}|D)$, up to the smaller $O(|\widetilde{E}|H)$ bias term, and avoids recomputing edge encodings across replicas.

The benefit is most visible on large instances. On an L4 GPU, the standard full-batch implementation without shared edges runs out of memory on many large-vehicle instances, while shared-edge inference completes all cases in the same evaluation set (Table 10). On medium instances where both paths run, shared-edge inference is essentially cost-neutral and is slightly faster at $N = 200$ and $N = 300$, while $N = 400$ is dominated by time-limit trajectory noise. On the smallest instances ($N \leq 75$) the plain batched forward pass is faster, and the SAG-Sep and SAG-Sep+OS root times at these sizes in Tables 1 and 3 use it; the bounds are identical either way.

Direct memory profiling confirms the source of the improvement. Even on the minimum-vehicle instance at each size, shared-edge inference reduces peak allocated memory by $7.4\times$–$13.1\times$ relative to the standard full-batch path (Table 11). The reduction grows with instance size because $|\widetilde{E}|$ grows faster than $|V|$ under graph augmentation.

C.4 Hyperparameter Selection

Graph and subset thresholds. SAG-Sep uses two simple thresholds at inference time. The LP-support threshold τ_1 filters edges with LP value below τ_1 before constructing the support graph, while the policy threshold τ_2 converts node probabilities into a subset, $\hat{S} = \{i \in V^+ : p_i \geq \tau_2\}$; this τ_2 corresponds to the threshold τ used in Section 3.3 and Appendix A.3. Unless stated otherwise, all experiments use $\tau_1 = 10^{-6}$ and $\tau_2 = 0.50$. Table 12 reports a one-at-a-time sensitivity sweep.

The sweep shows that threshold changes can move the final bound, but no alternative is uniformly better. The subset threshold is especially tied to near-boundary probabilities: shifting τ_2 changes which marginal nodes are included, and the violation-guided sweep of Onion-Sweep removes this dependence by selecting prefixes of the score ranking by their violation (Section 3.3). Because these thresholds are not very interpretable and can be over-tuned to a benchmark, we keep the fixed defaults $\tau_1 = 10^{-6}$ and $\tau_2 = 0.50$ in all non-ablation experiments.

Aggressive-sampling grid. For aggressive sampling, we densify the conditioning grid without retraining the model. Let $v = \lceil d(V^+)/C \rceil$ denote the vehicle lower bound used by the current LP state. We use

$$J_{\text{aggr}} = \left\lceil \max\{10\sqrt{v},\, 1.25v\} \right\rceil \tag{64}$$

conditioning values, evenly spaced over the normalized interval. The max rule creates a natural crossover at $v = (10/1.25)^2$: below this point, the $\sqrt{v}$ term acts as a heuristic boundary-scale proxy. In geometric CVRP instances, a compact customer region with capacity volume v has perimeter

Table 10: Inference feasibility with and without shared-edge encoding on an L4 GPU. Entries report non-OOM instances out of 10 random benchmark instances.

Size	Full batch, no shared edge	Microbatch 48	Shared edge
$N = 500$	8/10	9/10	**10/10**
$N = 750$	4/10	5/10	**10/10**
$N = 1000$	3/10	6/10	**10/10**

Table 11: Peak allocated GPU memory for full-batch inference without shared edges versus shared-edge inference.

Size	Profiled instance	No shared edge	Shared edge	Reduction
$N = 500$	random-19-X-n500	2.50 GiB	0.34 GiB	7.4×
$N = 750$	random-8-X-n750	6.46 GiB	0.66 GiB	9.8×
$N = 1000$	random-19-X-n1000	15.68 GiB	1.20 GiB	13.1×

on the order of $\sqrt{v}$, and RCC usefulness is governed by the boundary $\delta(S)$ rather than by volume alone. Above the crossover, the $1.25v$ term keeps the grid tied to the natural vehicle-count scale. This replaces a fixed large grid with a size-adaptive one: it avoids injecting too many noisy marginal subsets on small instances, while leaving enough capacity for larger states where additional valid cuts can be absorbed by the relaxation. Other grids are possible, but Onion-Sweep matches or improves on aggressive sampling at every size except $N = 500$ and $N = 750$ (where it is within 0.03 pp) without any extra forward pass; for this reason, a broader grid ablation is less informative than the structured decoding studied in Section 3.3.

Onion-MBP procedure. Our earlier decoder, Onion-MBP (margin-band probing), kept the hard threshold τ and probed a band of width Δ around it in every replica, combined with the same cross-replica onion step as Onion-Sweep (Section 3.3). Its candidate generation is as follows.

Onion-MBP candidate generation. Input: probabilities $\{p^{(j)}\}_{j=1}^{J}$, threshold τ, band width Δ, demands d, capacity C, and current LP solution $\bar{x}$.

1. Initialize candidate pool $\mathcal{C} \leftarrow \emptyset$.
2. **Per-replica MBP.** For each replica j, define the core and margin band
$$C_j = \{i \in V^+ : p_i^{(j)} > \tau + \Delta\}, \qquad B_j = \{i \in V^+ : |p_i^{(j)} - \tau| \le \Delta\}.$$
Add to $\mathcal{C}$ the hard-threshold subset $H_j = C_j \cup \{i \in B_j : p_i^{(j)} \ge \tau\}$ and four boundary probes: $C_j \cup B_j$, C_j plus a random half of B_j, C_j plus the high-probability half of B_j, and C_j plus the low-probability half of B_j.
3. **Cross-replica onion.** Compute a soft frequency score $a_i = \frac{1}{J}\sum_{j=1}^{J} p_i^{(j)}$, sort customers as $v_1, \ldots, v_{|V^+|}$ by decreasing a_i, form nested prefixes $P_k = \{v_1, \ldots, v_k\}$, and add P_k to $\mathcal{C}$ whenever $r_k = \lceil d(P_k)/C \rceil$ reaches a new vehicle-count level.
4. **Finalize subsets.** Filter $\mathcal{C}$ and keep a proposed subset S only if $2\lceil d(S)/C \rceil - \bar{x}(\delta(S)) > 0$.

Output: violated RCC-defining subsets in $\mathcal{C}$.

MBP treats the nodes far above the threshold as a stable core and explores only the uncertain band around τ, where small probability shifts can change the induced RCC. It targets the same source of missed cuts as the aggressive conditioned probing of Appendix A.3, namely subsets that appear between the base conditioning values, without evaluating a denser conditioning grid.

Onion-MBP ablation. The ablation row in Table 3 uses $\Delta = 0.15$ with cut management (at most 50 cuts added per round, an active-cut limit of 1000, and removal after 15 inactive rounds) for $N \le 500$, and the threshold readout plus the onion step, without band probes, for $N \ge 750$, where the active-cut limit hurt the bound. In preliminary runs the band width was not a sensitive parameter ($\Delta = 0.10$ and $\Delta = 0.15$ were within about 0.1 pp of each other at all sizes up to $N = 400$), and beyond $N = 75$ the band probes added little over the onion step alone, which motivated replacing them by the violation-guided sweep.

Table 12: Sensitivity to graph and subset thresholds. Each cell reports average root gap (%, lower is better). Run at an earlier development stage; absolute gaps differ from Table 1, and comparisons are within the table.

Setting	τ_1	τ_2	$N = 50$–100	$N = 200$–400	$N = 500$–1000
default	10^{-6}	0.50	**5.88**	9.43	17.35
$\tau_1 = 0.05$	0.05	0.50	5.91	**9.42**	17.58
$\tau_1 = 0.10$	0.10	0.50	6.01	**9.42**	17.84
$\tau_2 = 0.45$	10^{-6}	0.45	6.31	9.45	**17.28**
$\tau_2 = 0.55$	10^{-6}	0.55	6.23	9.48	17.93

D Additional Discussions and Results

This appendix collects supplementary experiments that contextualize the main results: a NeuralSEP$_1$ reproduction and a one-shot baseline (§D.1), fixed-round comparisons at $N = 1000$ (§D.2), an fp32 vs. TF32 robustness check (§D.3), a serial SAG-Sep + CVRPSEP hybrid (§D.4), a K-hop ablation (§D.5), paired per-instance statistics (§D.6), and transfer beyond the benchmark (§D.7).

D.1 NeuralSEP$_1$ Reproduction and One-Shot Base Model

This subsection contextualizes SAG-Sep against the original NeuralSEP$_1$ pipeline by reproducing both its one-shot base model and its recursive coarsening variant. The reproduction runs on a legacy machine that differs from our main-benchmark hardware, but all rows in Table 13 use the same machine, software stack, and solver configuration, so within-table comparisons are controlled.

Reproduction setup. We reproduce the NeuralSEP$_1$ one-shot baseline using the authors' open-sourced code and best released one-shot checkpoint. The reproduction environment uses the original (older) software stack (DGL with CUDA 11.3) and runs on a legacy machine with an Intel Core i9-10980XE CPU at 3.00 GHz and two NVIDIA A5000 GPUs. We use a simple 2-GPU inference split and keep the remaining code and solver settings unchanged.

One-shot base model. The one-shot row of Table 13 confirms that NeuralSEP$_1$'s base model is reasonably competitive on its training-size regime ($N = 50$–100) but degrades sharply with instance size: at $N = 750$ and $N = 1000$ it produces essentially zero violated cuts, terminates the cutting loop early, and leaves large optimality gaps. The robustness of the full NeuralSEP$_1$ pipeline on large instances therefore comes from recursive coarsening rather than from a strong one-shot predictor.

Coarsening reference. The lower row of Table 13 reports the best-performing recursive coarsening variant on the same machine. Coarsening substantially improves robustness on large instances but incurs additional overhead on small ones. Crucially, even the coarsening variant remains far behind SAG-Sep at $N \geq 500$ in the main results, indicating that SAG-Sep's advantage on large instances is not only a runtime artifact.

D.2 Fixed-Round Comparisons at N=1000

To isolate separation quality from the time-limit cutoff that dominates large-N runs, we compare exact separation and SAG-Sep on the same ten $N = 1000$ instances under a fixed budget of ten iterations of the cutting-plane loop. Table 14 reports the final mean gap and end-to-end runtime. SAG-Sep essentially matches the exact separator at this resolution—only a 0.003 percentage-point gap difference—while reducing the mean end-to-end runtime by about $870\times$.

The speed difference comes primarily from cut generation. Across the same ten-iteration runs, SAG-Sep spends 0.65s per separation call on average, including 0.53s for the encoder forward pass, whereas exact separation spends most of its runtime inside the cut-generation routine. Over their first ten rounds on the same instances in the runs of Table 1, the NeuralSEP pipelines spend 15.98s (NeuralSEP$_2$), 18.95s (π-NeuralSEP$_2$) and 19.31s (π-NeuralSEP$_2$+GC) per separation round, 25–30 times more than SAG-Sep, and CVRPSEP spends 4.49s.

Table 13: NeuralSEP$_1$ reproduction on a legacy machine (Intel i9-10980XE + 2×A5000) using the released code and checkpoints. Both rows use the same hardware and solver setting; reported metrics are averages over instances.

Method	Size	Avg. Gap (↓)	Avg. LB (↑)	Avg. Runtime	Avg. Iter.	Avg. Cuts	Count
NeuralSEP$_1$ one-shot	50	8.76%	8719.81	6.65	23.40	69.10	10
	75	9.17%	12612.16	27.80	48.20	244.50	10
	100	10.13%	15067.78	66.89	68.70	407.80	10
	200	13.39%	19780.76	433.91	103.90	748.00	10
	300	17.18%	28617.47	1928.83	91.20	1534.60	10
	400	24.36%	37921.09	2778.64	85.60	624.40	10
	500	40.87%	36222.52	748.99	21.80	69.10	10
	750	58.42%	37319.85	91.98	1.00	0.00	10
	1000	59.73%	37986.40	149.50	1.00	0.00	10
NeuralSEP$_1$ coarsen (best)	50	4.10%	9150.81	79.57	39.60	154.60	10
	75	4.95%	13087.16	175.38	60.50	369.80	10
	100	6.81%	15578.66	346.04	88.50	609.90	10
	200	8.94%	20780.08	1125.99	134.60	1230.90	10
	300	10.65%	30957.90	2627.44	122.10	2149.20	10
	400	13.78%	43304.17	3539.18	81.20	2248.10	10
	500	16.27%	50434.53	3657.06	53.50	2024.80	10
	750	23.02%	70728.66	3784.96	26.60	1785.20	10
	1000	28.33%	67667.39	3856.85	18.90	1431.00	10

Table 14: Ten-iteration comparison between exact separation and SAG-Sep on ten $N = 1000$ instances (the final bound is the LP after nine rounds of cuts).

Method	Final gap (%)	Runtime (s)	Instance wins
Exact separation	35.411	23051.54	5 / 10
SAG-Sep	35.414	26.48	5 / 10

Equal-round comparison of decoders. To separate cut quality from inference speed and hardware, Table 15 reports the mean gap after each separation round on the same ten $N = 1000$ instances, starting from the same raw LP (59.73% at round 0) and ignoring time. Measured per round, the threshold readout stays within 1 pp of the exact per-band separator, OS overtakes it from round 5 on, and π-NeuralSEP$_2$+GC is the weakest at every round. OS adds many more cuts per round (about 150 at this size, versus one per demand band for the exact separator), which suggests that its per-round advantage comes mainly from larger batches; we do not measure per-cut quality separately.

D.3 fp32 vs. TF32 Precision Sensitivity

We compare full fp32 inference against TF32-enabled inference on the same benchmark, checkpoint, and LP setting. Table 16 summarizes four matched configurations: single-B200 fp32/TF32 and single-L4 fp32/TF32. TF32 does not yield a uniform bound–time improvement. On B200 it is sometimes slightly faster on small instances, but the final gap can also worsen; on L4 it worsens the gap at $N = 50$ and $N = 75$ and is within 0.35 pp elsewhere, slightly better at $N = 200$ and $N \geq 500$. The effect is not a pure throughput issue: thresholded subset prediction changes when a few node scores move across the $\tau = 0.5$ cutoff—the same boundary instability that Onion-Sweep removes by decoding prefixes of the score ranking by violation (Section 3.3). For this reason, all main results use full fp32 inference.

D.4 Serial SAG-Sep + CVRPSEP Hybrid

We evaluate a simple serial hybrid in which SAG-Sep runs first and CVRPSEP is activated only after the learned separator stalls. This is a supplementary analysis rather than a main comparison, because it augments the learned separator with a mature classical routine instead of testing SAG-Sep on its own. Table 17 compares this hybrid against the baseline SAG-Sep root loop on $N = 50$–400. The hybrid consistently tightens the root bound, especially on small and medium instances, but the extra LP and heuristic work becomes expensive at larger sizes. Against the pure CVRPSEP results in Table 1, the hybrid is especially attractive from $N = 200$ onward: at $N = 200$ it is within 0.01 pp of CVRPSEP's gap in about a third of the runtime, and at $N = 300$ and $N = 400$ it reaches lower gaps

Table 15: Mean root gap (%) after each separation round on the ten $N = 1000$ benchmark instances, from the same raw LP and independent of time. The exact column is the exact separator of Table 14; the threshold column is the SAG-Sep run of that table, whose final bound is round 9; π-NeuralSEP$_2$+GC uses the re-run batch of Appendix D.6.

Round	Threshold	OS	π-NeuralSEP$_2$+GC	Exact
1	50.34	**49.99**	51.05	50.19
2	47.72	**47.58**	48.45	47.72
3	45.60	45.55	46.01	**45.33**
4	43.49	42.93	43.81	**42.78**
5	41.66	**40.56**	42.03	40.87
6	39.93	**38.57**	40.27	39.23
7	38.18	**36.96**	38.89	37.85
8	36.77	**35.56**	37.35	36.52
9	35.41	**34.33**	35.81	35.41

Table 16: Full fp32 versus TF32 on the same root benchmark. Each cell reports mean root gap (%) with mean runtime in seconds in parentheses. Shared-edge encoding is on at all sizes, so the L4 fp32 times at $N \leq 75$ differ from Table 1 (Appendix C.3).

Setting	$N = 50$	$N = 75$	$N = 100$	$N = 200$	$N = 300$	$N = 400$	$N = 500$	$N = 750$	$N = 1000$
B200, fp32	4.37(0.60)	5.91(1.80)	7.11(3.50)	9.10(29.50)	9.42(411.30)	9.98(2066.90)	11.20(3046.10)	17.37(3627.00)	22.19(3680.30)
B200, TF32	5.35(0.50)	6.00(1.60)	7.16(3.10)	8.94(29.90)	9.46(411.90)	9.95(2138.90)	11.62(3320.00)	17.36(3636.50)	22.18(3659.80)
L4, fp32	4.37(0.50)	5.91(1.40)	7.11(2.80)	9.10(23.00)	9.42(343.30)	9.98(1897.30)	11.35(3045.50)	17.41(3651.00)	22.03(3655.60)
L4, TF32	5.29(0.40)	6.13(1.40)	7.10(2.80)	8.96(25.30)	9.43(449.70)	9.99(2108.00)	11.17(2934.00)	17.25(3637.40)	21.68(3635.20)

in less time. The tradeoff is therefore favorable when near-exact root bounds matter. The same setup also suggests a unified bound–time evaluation protocol—run each separator until a fixed root-bound target and compare runtimes—which we leave to future work.

D.5 Detailed K-hop Ablation

Table 18 reports the ablation over the K-hop closure radius. Larger hop radii occasionally help on the smallest instances but scale poorly: from $N = 100$ onward, $K = 2$ gives the best gap at every size, and the wider radii are consistently weaker. We therefore use $K = 2$ as the default for all models.

D.6 Paired Per-Instance Comparisons

Table 19 complements the mean gaps of Tables 1 and 3 with paired per-instance statistics on the ten benchmark instances per size: win/loss/tie counts, the mean gap difference with its standard error, and the worst-case difference, for SAG-Sep+OS against the threshold readout, against Onion-MBP, and against π-NeuralSEP$_2$+GC. The π-NeuralSEP$_2$+GC runs used for the paired comparison were re-run after the OS runs on the same hardware configuration (one L4 GPU, four CPU cores), so their mean gaps (3.84 / 4.58 / 5.89 / 7.76 / 8.73 / 10.77 / 12.91 / 20.29 / 26.04 for $N = 50$ to 1000) differ slightly from the values in Table 1; the threshold runs are those of Table 3. OS improves on the threshold readout on 85 of 90 instances; on Onion-MBP on 71 of 90, losing on average only at $N = 500$ (where a single instance accounts for the -2.26 pp worst case) and tying on average at $N = 1000$ ($+0.01$ pp, 3 of 10 wins); and on π-NeuralSEP$_2$+GC on 77 of 90, winning all 30 instances at $N \geq 500$ with positive worst-case margins.

D.7 Generalization Beyond the Benchmark

To test transfer beyond the NeuralSEP benchmark distribution, we evaluate the frozen checkpoint, trained at $N = 100$, on two XML parameter settings at unseen sizes $N = 500$, 750, and 1000 (ten instances per size, 5–10$\times$ the training size) and on ten CVRPLIB X instances with N between 524 and 979, all under the same 3600-second root limit. Table 20 reports the XML regimes and Table 21 the X instances; all gaps are computed against the best known solutions; the XML instances and their best known solutions are included in the released repository (Appendix E). OS wins all 60 paired comparisons against π-NeuralSEP$_2$+GC on the XML regimes and 9 of 10 on the X instances (losing X-n524-k153 by 0.36 pp), and attains a lower gap than CVRPSEP on all 70 instances. On X-n524-k153 the demand per customer is unusually large relative to the vehicle capacity (153 routes

Table 17: Serial SAG-Sep + CVRPSEP hybrid versus the SAG-Sep baseline on $N = 50$–400. Each cell reports mean root gap (%) with mean runtime in seconds in parentheses.

Method	$N = 50$	$N = 75$	$N = 100$	$N = 200$	$N = 300$	$N = 400$
SAG-Sep baseline	4.37(0.30)	5.91(1.20)	7.11(2.80)	9.10(23.00)	9.42(343.30)	9.98(1897.30)
SAG-Sep + CVRPSEP	1.97(0.40)	2.78(1.70)	4.55(4.50)	6.29(62.30)	7.24(781.50)	8.69(2719.10)

Table 18: Ablation on the K-hop closure parameter ($N \leq 500$). We report mean lower bound (LB), mean runtime (Time, seconds), and mean optimality gap (Gap, %). Best Gap per N is in bold. Separately trained models from an earlier development stage; absolute gaps differ from Table 1, and comparisons are within the table.

N	2-hop			3-hop			4-hop			8-hop		
	LB	Time	Gap	LB	Time	Gap	LB	Time	Gap	LB	Time	Gap
50	8958.65	0.64	6.23%	8951.04	0.70	6.37%	8955.10	0.85	6.15%	8995.99	0.79	**5.61%**
75	12864.96	1.94	7.00%	12926.47	2.19	6.41%	12890.30	2.08	6.67%	12935.72	2.72	**6.35%**
100	15583.42	4.98	**6.82%**	15498.78	4.24	7.38%	15484.65	5.26	7.41%	15456.56	5.79	7.56%
200	20816.81	36.29	**8.83%**	20770.05	32.81	9.00%	20672.88	40.99	9.48%	20683.41	48.17	9.37%
300	31424.26	483.62	**9.52%**	31164.89	290.98	10.04%	31047.29	312.28	10.49%	31182.41	459.56	10.15%
400	45505.74	2066.06	**9.82%**	44908.79	1446.56	10.88%	44499.48	1322.68	11.66%	44940.15	2103.80	10.92%
500	53800.54	3072.64	**11.12%**	53048.32	2768.78	12.16%	52523.67	2459.50	13.28%	52643.90	3203.49	12.81%

for 524 customers), so the useful cuts are many small, disjoint subsets that the prefix-based decoder covers only partially; the test-time cut-refinement layer of Appendix E compensates for this (gap 17.04% on this instance, against 19.40% for OS and 19.04% for π-NeuralSEP$_2$+GC), so we did not pursue it further. The π-NeuralSEP$_2$+GC and CVRPSEP baselines are run from the NeuralSEP code base (CVRPSEP with its RCC routines only), with the same single-thread LP setting as in Table 1.

E Code Release and Scope Beyond the Root Node

The code accompanying this paper is released at `https://github.com/henryleo180/SAG-Sep`. The repository contains a C11 implementation of the separator (ONNX inference, root-only solver and a library interface for host solvers), a Python reference implementation, the trained checkpoint and its ONNX export, the training pipeline, and the benchmark instances used in this paper. The exact-label set the checkpoint was trained on is archived at `https://doi.org/10.5281/zenodo.23539579`. The `os` recipe reproduces the SAG-Sep+OS row of Table 3; the C and Python implementations decode the same candidates and differ only in floating-point details.

The repository also goes beyond the scope of the present paper. It ships branch-and-cut examples in which SAG-Sep separates RCCs for the capacitated, asymmetric, open and time-windowed VRP and for the capacitated minimum spanning tree problem, a branch-price-and-cut integration on top of RouteOpt [You and Yang, 2025] for the CVRP and the VRPTW, and a training-free test-time cut-refinement layer (TTS) for the rounding-driven states that dominate beyond the root node (Section 5); the branch-and-cut examples use the paper's checkpoint, and the branch-price-and-cut integration ships a second checkpoint fine-tuned from it on column-generation states. These components are not part of the contributions claimed here and are evaluated only preliminarily below. We report these preliminary results mainly to address questions raised during the review about the framework's potential inside branch-and-cut and branch-price-and-cut solvers; the main text still focuses only on the root node, which is also the setting in which NeuralSEP$_1$ [Kim et al., 2024] and NeuralSEP$_2$ [Sim et al., 2025] are developed and evaluated. A forthcoming extended version describes and evaluates these components in detail.

E.1 The Released Engine Run to Closure Against CVRPSEP

The main tables stop at a fixed root time limit and compare separators by the bound they reach within it. Table 22 instead reports the released C engine with its full recipe (OS followed by TTS) and the CVRPSEP separator, both run in the same root cutting-plane loop to closure, that is, until the separator proposes no admissible cut, with tail-off stopping disabled, on the ten random benchmark instances per size. Both reach the same root bound at every size to within 0.002 points of gap; the

Table 19: Paired per-instance comparison of SAG-Sep+OS on the ten benchmark instances per size. W/L/T counts instances on which OS attains a better/worse/equal root bound; Δ is the opponent's gap minus the OS gap in percentage points (positive favors OS), reported as mean $\pm$ standard error and as the worst case over the ten instances.

	vs. SAG-Sep (threshold)			vs. SAG-Sep+Onion-MBP			vs. π-NeuralSEP$_2$+GC		
N	W/L/T	mean $\Delta\pm$SE	worst Δ	W/L/T	mean $\Delta\pm$SE	worst Δ	W/L/T	mean $\Delta\pm$SE	worst Δ
50	10/0/0	$+1.77 \pm 0.29$	$+0.27$	9/1/0	$+0.87 \pm 0.19$	-0.01	10/0/0	$+1.24 \pm 0.20$	$+0.36$
75	10/0/0	$+2.11 \pm 0.51$	$+0.65$	8/2/0	$+0.37 \pm 0.11$	-0.14	9/1/0	$+0.77 \pm 0.25$	-0.71
100	10/0/0	$+1.68 \pm 0.28$	$+0.44$	10/0/0	$+0.46 \pm 0.09$	$+0.11$	8/2/0	$+0.45 \pm 0.19$	-0.56
200	10/0/0	$+1.49 \pm 0.15$	$+0.90$	10/0/0	$+0.40 \pm 0.09$	$+0.08$	5/5/0	$+0.14 \pm 0.14$	-0.41
300	10/0/0	$+0.86 \pm 0.13$	$+0.47$	9/1/0	$+0.29 \pm 0.06$	-0.05	6/4/0	$+0.17 \pm 0.21$	-0.76
400	10/0/0	$+0.79 \pm 0.09$	$+0.34$	9/1/0	$+0.21 \pm 0.09$	-0.44	9/1/0	$+1.58 \pm 0.31$	-0.23
500	8/2/0	$+0.57 \pm 0.22$	-0.66	6/4/0	-0.12 ± 0.28	-2.26	10/0/0	$+2.12 \pm 0.37$	$+0.43$
750	8/2/0	$+0.39 \pm 0.14$	-0.32	7/3/0	$+0.22 \pm 0.14$	-0.36	10/0/0	$+3.27 \pm 0.34$	$+2.14$
1000	9/1/0	$+0.78 \pm 0.19$	-0.00	3/7/0	$+0.01 \pm 0.17$	-0.85	10/0/0	$+4.79 \pm 0.22$	$+3.80$
all	85/5/0			71/19/0			77/13/0		

Table 20: Generalization to two XML parameter settings at unseen sizes, 5–10$\times$ the training size (ten instances per cell, 3600-second limit). Mean root gap (%); W/T/L counts the paired per-instance comparisons of OS against each baseline. XML-A: central depot, random customers, many small and few large demands, medium routes. XML-B: corner depot, random customers, small high-variance demands, medium routes.

Regime	N	OS	π-NeuralSEP$_2$+GC	CVRPSEP	OS vs. π-NeuralSEP$_2$+GC	OS vs. CVRPSEP
XML-A	500	**8.37**	8.85	11.54	10/0/0	10/0/0
XML-A	750	**13.47**	15.47	18.25	10/0/0	10/0/0
XML-A	1000	**17.35**	20.77	22.35	10/0/0	10/0/0
XML-B	500	**8.86**	11.78	16.53	10/0/0	10/0/0
XML-B	750	**17.19**	19.56	23.07	10/0/0	10/0/0
XML-B	1000	**21.61**	25.50	27.27	10/0/0	10/0/0

learned separator runs its forward pass on the GPU while CVRPSEP runs on the CPU, so the time columns are a systems comparison rather than a like-for-like separator speed. Under this protocol the learned engine is slower than CVRPSEP at $N = 50$ and $N = 75$, and faster from $N = 100$ on, by a factor of three at $N = 200$. On a single CPU core (no GPU) the engine needs 0.50, 6.9 and 67 seconds per instance at $N = 50$, 100 and 200, with the same bounds to within 0.002 points, so the crossover moves to $N = 200$.

E.2 Preliminary Complete Branch-and-Cut

Table 23 reports complete branch-and-cut runs, with a two-hour limit per run, on the five $N = 75$ and $N = 100$ benchmark instances that CVRPSEP solves within that limit, so the table compares solve times on these instances rather than solve rates. Four separation arms are compared in the same research harness: OS followed by CVRPSEP once OS stalls; OS with the test-time cut-refinement layer and no call to CVRPSEP; π-NeuralSEP$_2$+GC followed by CVRPSEP, run in its own code base; and CVRPSEP alone. On every instance the four root bounds agree to within 0.06 percentage points, so the differences in time and tree size come mainly from the tree search. The refinement layer carries all five complete trees without any CVRPSEP call, and on random-18 it is the fastest configuration with the smallest tree; CVRPSEP is the fastest on random-13 and random-37, and OS+CVRPSEP on random-21. The released examples reproduce the CVRPSEP and refinement-layer arms; the other two arms come from the research harness.

E.3 Capacity and Rounding Parts of Exact RCC Violations

The violation of an RCC splits into a capacity part, $2d(S)/C - \bar{x}(\delta(S))$, and a rounding part, $2(\lceil d(S)/C \rceil - d(S)/C)$. Table 24 classifies the 25,411 exact cuts found in complete branch-and-cut runs with exact separation on 30 benchmark instances ($N = 50$, 75, 100) by the phase in which they were found. Early root cuts are overwhelmingly capacity-dominated and large; by the second half of the root loop, and at every tree node, the capacity part is non-positive for essentially all cuts and the remaining violations are small and rounding-driven. The transition therefore happens within the root

Table 21: Root gap (%) on ten CVRPLIB X instances drawn uniformly from $N \in [500, 1001]$ (3600-second limit).

Instance	OS	π-NeuralSEP$_2$+GC	CVRPSEP
X-n524-k153	19.40	**19.04**	23.49
X-n573-k30	**3.36**	5.14	5.81
X-n613-k62	**11.42**	13.57	16.36
X-n701-k44	**12.54**	16.24	19.49
X-n716-k35	**16.94**	19.65	20.75
X-n749-k98	**19.14**	20.49	26.00
X-n766-k71	**13.97**	17.28	19.81
X-n837-k142	**20.37**	23.96	26.42
X-n936-k151	**20.67**	25.85	24.42
X-n979-k58	**23.34**	28.23	27.06
Mean	**16.11**	18.94	20.96

Table 22: Released C engine (full recipe) versus CVRPSEP, both run to closure in the same root loop (ten random instances per size, means). Gap is with respect to the optimal solution. The time ratio is engine time over CVRPSEP time, computed from unrounded means.

N	CVRPSEP gap (%)	Engine gap (%)	CVRPSEP s/inst (CPU)	Engine s/inst (GPU)	Time ratio
50	1.969	1.968	0.09	0.16	1.86×
75	2.768	2.768	0.63	0.74	1.19×
100	4.539	4.538	2.93	2.20	0.75×
200	6.281	6.283	95.8	32.1	0.34×

loop through repeated separation; branching adds nothing qualitatively new, which is why separators tuned to the early, batch-oriented root regime need a different mechanism beyond it (Section 5).

E.4 Transfer Across Problem Classes: The Released Examples

The released branch-and-cut examples run the engine zero-shot, with the paper's checkpoint and its full recipe (OS followed by TTS), on four VRP variants and on the capacitated minimum spanning tree problem, against CVRPSEP in the same driver (Tables 25 and 26; one CPU thread, 1800-second limit, one run per instance). On the capacitated, asymmetric and time-windowed variants (30 instances) the two separators reach identical root bounds and both solve all 30; on the hardest VRPTW instance (rc101_100) the engine finishes in 478 seconds against 1785 for CVRPSEP. On the open VRP the root bounds agree on 3 of 7 instances and differ by at most 0.004 points of gap on the other 4; each separator solves the same 3, and on the 4 unsolved the engine ends with the higher bound on 3. The engine reaches these root bounds with 50–76% of the root cuts CVRPSEP adds, but its CPU root loop is slower, so total times split evenly: among the runs that both separators solve and in which the slower one takes more than one second, it is faster on 6, slower on 6 and level on 1 (differences within 10% counted as level). On the CMST, a different problem class with the capacity cut as its only family and no CMST training, the root bounds differ on 19 of 20 instances: the engine is higher on 13, by up to 5.8 points of gap at tight capacity ($Q \in \{200, 400\}$), and lower on 6 by at most 0.44 points; it solves 3 instances against 1 within the limit, a count that changes with the tree-commitment setting of the driver. These are runs of the released examples rather than a benchmark: one seed, small instance sets, and timings from shared cluster nodes.

Table 23: Preliminary complete branch-and-cut on the five $N{=}75/100$ benchmark instances that CVRPSEP solves within the two-hour limit (two-index formulation with lazy RCC callbacks, 2-hour limit, one LP thread, cutoff at the optimum plus one, so the runs measure the time to prove optimality, user cuts at the first 31 nodes, integer solutions checked exactly; one node type with a B200 GPU, not the L4 machine of the main tables). Each cell gives wall-clock seconds / tree nodes; TL marks the time limit, with the remaining gap to the optimum (%) in parentheses. OS+CVRPSEP: OS at the root until it stalls, then CVRPSEP; both separate in the tree. TTS: OS with a training-free test-time cut-refinement layer (a research version of the layer in the released code) at the root and in the tree, without any call to CVRPSEP. π-NeuralSEP$_2$+GC+CVRPSEP runs in its own code base and is included as a reference; the CVRPSEP column is an earlier run of the same harness.

Instance	OS+CVRPSEP	TTS (no CVRPSEP)	π-NeuralSEP$_2$+GC+CVRPSEP	CVRPSEP
random-13-X-n75	5 / 376	12 / 425	34 / 1,515	**4** / 226
random-21-X-n75	**22** / 2,832	32 / 4,134	83 / 5,134	35 / 3,804
random-37-X-n75	78 / 13,319	113 / 19,323	179 / 8,725	**53** / 8,427
random-18-X-n100	346 / 31,215	**122** / 11,046	609 / 44,518	1185 / 114,243
random-34-X-n100	2255 / 208,434	**1980** / 180,459	TL (0.51)	2100 / 186,129

Table 24: Capacity and rounding parts of the violation of exact RCCs in complete branch-and-cut runs with exact separation (30 random benchmark instances, $N = 50, 75, 100$; 25,411 cuts). A cut is capacity-dominated if its capacity part exceeds its rounding part; violations are in the units of $\bar{x}(\delta(S))$. Root round r of an instance's R rounds falls in the first tenth if $r/R \leq 0.1$ and in the first half if $r/R \leq 0.5$; none of the 6,424 late-root and 8,453 tree cuts is capacity-dominated.

Phase	Cuts	Mean violation	Capacity-dominated (%)	Capacity part > 0 (%)
Root, first 10% of rounds	2,032	7.88	86.1	96.9
Root, 10–50% of rounds	8,502	2.34	22.5	48.8
Root, last 50% of rounds	6,424	0.24	0.0	0.5
Tree nodes	8,453	0.13	0.0	0.2

Table 25: Released branch-and-cut examples on four VRP variants: CVRPSEP versus SAG-Sep, one run per instance. Both arms use the same C driver, formulation, fleet (k routes, fixed by an equality), root and tree schedule, exact rescoring and admission rule, and differ only in the source of rounded capacity cuts: the root loop runs to closure (at most 100 rounds of at most 100 cuts), user cuts are separated at the first 15 tree nodes, and every integer solution is checked exactly; one CPU thread, 1800-second limit, relative MIP gap 10^{-4}, the MIP solver's own cuts and heuristics at their defaults. SAG-Sep is the released C engine with its full recipe, including the test-time cut-refinement layer, and the paper's checkpoint, applied zero-shot through each example's directed or split-depot aggregation, with its forward pass on the CPU. Gap: root bound relative to the reference value (%); Root and Total: seconds; Nodes: tree nodes; TL: time limit, with the remaining gap of the final lower bound (%) in parentheses. References are proved optima except those marked † (best known). Counts in the text use unrounded bounds. Instances: four $N{=}50$ random benchmark instances of this paper (CVRP); the Fischetti–Toth–Vigo asymmetric instances (ACVRP); seven Christofides–Mingozzi–Toth instances as open VRPs (OVRP); Solomon series 1 and 2 with 25, 50 and 100 customers, distances truncated to one decimal (VRPTW).

				CVRPSEP				SAG-Sep (released engine, OS+TTS)			
Instance	n	k	Ref.	Gap	Root	Total	Nodes	Gap	Root	Total	Nodes
CVRP											
random-4-X-n50	50	4	7684	1.37	0.02	1.4	508	1.37	0.23	4.9	1,080
random-8-X-n50	50	3	6733	0.90	0.02	0.4	137	0.90	0.18	0.8	111
random-12-X-n50	50	4	8317	0.89	0.06	13.8	603	0.89	0.26	4.7	345
random-24-X-n50	50	3	6686	2.15	0.02	7.0	1,494	2.15	0.20	3.2	2,046
ACVRP											
A034-02f	33	2	1406	2.56	0.01	0.7	40	2.56	0.05	1.2	29
A036-03f	35	3	1644	0.74	0.03	0.3	16	0.74	0.12	0.6	20
A039-03f	38	3	1654	0.00	0.01	0.0	1	0.00	0.08	0.1	1
A045-03f	44	3	1740	0.00	0.04	0.3	1	0.00	0.15	0.4	1
A048-03f	47	3	1891	0.32	0.02	0.2	1	0.32	0.11	0.5	1
A056-03f	55	3	1739	1.24	0.02	0.7	87	1.24	0.16	1.8	20
A065-03f	64	3	1974	1.41	0.08	3.7	1	1.41	0.19	2.6	1
A071-03f	70	3	2054	1.36	0.02	0.1	1	1.36	0.18	0.4	1
OVRP											
O1	50	5	416.063	1.72	0.02	16.0	2,128	1.72	0.18	11.5	1,069
O2	75	10	567.14†	3.83	0.15	TL (2.35)	45,007	3.83	0.90	TL (2.41)	44,136
O3	100	8	639.744	1.22	0.18	313.3	29,188	1.22	0.64	123.2	11,756
O4	150	12	733.13†	1.36	1.30	TL (0.69)	34,448	1.37	3.49	TL (0.66)	28,912
O5	199	16	896.08†	6.03	9.97	TL (5.73)	2,563	6.03	14.69	TL (5.61)	13,293
O6	120	7	682.12†	5.77	3.27	TL (4.74)	15,123	5.77	4.15	TL (4.32)	21,298
O7	100	10	534.239	1.46	0.79	72.5	986	1.46	2.48	69.4	1,644
VRPTW											
c101_25	25	3	191.3	0.00	0.00	0.0	1	0.00	0.01	0.0	1
c101_50	50	5	362.4	0.00	0.00	0.0	1	0.00	0.04	0.1	1
c101_100	100	10	827.3	0.00	0.02	0.1	1	0.00	0.26	0.3	1
r101_25	25	8	617.1	0.00	0.00	0.0	1	0.00	0.01	0.0	1
r101_50	50	12	1044	0.93	0.01	0.0	1	0.93	0.02	0.0	1
r101_100	100	20	1637.7	1.31	0.01	0.1	1	1.31	0.09	0.8	1
rc101_25	25	4	461.1	11.14	0.00	0.1	171	11.14	0.02	0.2	171
rc101_50	50	8	944	13.48	0.01	5.6	3,867	13.48	0.08	9.8	8,954
rc101_100	100	15	1619.8	15.83	0.08	1784.6	399,365	15.83	0.78	478.4	109,372
c201_25	25	2	214.7	0.00	0.00	0.0	1	0.00	0.01	0.0	1
c201_50	50	3	360.2	0.00	0.00	0.0	1	0.00	0.01	0.0	1
c201_100	100	3	589.1	0.00	0.01	0.0	1	0.00	0.03	0.0	1
r201_25	25	4	463.3	2.84	0.00	0.0	1	2.84	0.01	0.0	1
r201_50	50	6	791.9	2.27	0.01	0.1	1	2.27	0.02	0.1	1
r201_100	100	8	1143.2	7.15	0.05	12.4	1,272	7.15	0.08	14.6	958
rc201_25	25	3	360.2	0.39	0.00	0.0	1	0.39	0.01	0.0	1
rc201_50	50	5	684.8	6.32	0.01	0.1	66	6.32	0.03	0.2	42
rc201_100	100	9	1261.8	9.30	0.05	13.6	1,401	9.30	0.12	15.6	1,703

Table 26: Released branch-and-cut example on the capacitated minimum spanning tree problem (OR-Library `capmst` instances; n terminals, branch capacity Q; all references are proved optima). The model is the natural undirected edge formulation with the capacity cut as its only dynamic family. Its connectivity members ($\lceil d(S)/Q \rceil = 1$) are separated exactly by maximum flow in both arms; sets with $\lceil d(S)/Q \rceil \geq 2$ come from CVRPSEP's CVRP heuristics or from SAG-Sep zero-shot (the engine, recipe and checkpoint of Table 25, no CMST training), are rescored exactly against the CMST point, and enter in inner and/or cut-set form. Driver settings, columns and limits are those of Table 25.

				CVRPSEP				SAG-Sep (released engine, OS+TTS)			
Instance	n	Q	Ref.	Gap	Root	Total	Nodes	Gap	Root	Total	Nodes
cm50r1	49	200	1098	11.14	0.16	TL (9.47)	229,865	8.65	1.88	TL (6.47)	302,447
cm50r2	49	200	974	16.31	0.15	TL (13.76)	262,483	11.84	1.72	TL (8.83)	335,159
cm50r3	49	200	1186	11.52	0.25	TL (10.20)	194,989	8.03	2.03	TL (6.07)	220,994
cm50r4	49	200	800	11.50	0.06	TL (6.50)	272,333	9.99	0.96	TL (5.25)	282,248
cm50r5	49	200	928	15.16	0.11	TL (11.31)	187,211	10.76	2.48	TL (8.19)	297,886
cm50r1	49	400	679	12.02	0.08	TL (10.31)	160,868	7.69	0.66	TL (4.86)	266,054
cm50r2	49	400	631	13.54	0.09	TL (11.09)	156,504	10.32	0.73	TL (6.66)	307,304
cm50r3	49	400	732	13.19	0.05	TL (11.07)	191,722	7.39	0.68	TL (4.78)	407,596
cm50r4	49	400	564	8.52	0.07	TL (3.72)	246,341	7.84	0.41	TL (3.01)	217,671
cm50r5	49	400	611	10.49	0.08	TL (8.35)	114,945	7.96	0.71	TL (4.91)	315,654
cm50r1	49	800	495	5.00	0.04	TL (1.62)	209,968	4.79	0.16	899.7	119,499
cm50r2	49	800	513	10.77	0.03	TL (7.02)	179,280	11.21	0.17	TL (5.65)	252,559
cm50r3	49	800	532	6.07	0.06	TL (3.38)	208,358	5.64	0.28	TL (2.26)	194,399
cm50r4	49	800	471	5.94	0.05	22.6	9,494	5.94	0.24	6.1	545
cm50r5	49	800	492	7.29	0.03	TL (3.25)	145,122	6.71	0.23	TL (0.81)	630,510
cm100r1	99	800	182	1.05	0.93	TL (0.55)	48,609	1.24	2.80	241.1	11,071
cm100r2	99	800	179	1.74	1.35	TL (0.56)	45,197	1.84	3.52	TL (1.12)	37,352
cm100r3	99	800	175	1.67	1.41	TL (1.14)	43,060	1.71	1.96	TL (1.14)	47,700
cm100r4	99	800	183	1.64	0.65	TL (1.09)	36,776	1.64	2.37	TL (1.09)	37,617
cm100r5	99	800	186	1.88	1.67	TL (1.61)	34,764	1.95	2.37	TL (1.08)	40,232

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction's claims (single-pass separation efficiency, $\sim 30\times$ neural-separation speedup at $N = 1000$, 17.7% gap reduction with Onion-Sweep) directly correspond to the experimental results in Table 1, Table 3, and Appendix D.2 (per-round separation times).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations section (Section 5) discusses root-node scope, exact-label cost and transfer limits, benchmark scope and regime, hardware confounding, hand-designed test-time logic, and problem variants.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All theoretical results (Assumption A.1; Lemmas A.2, B.1; Propositions 3.1, A.3, B.2, B.3) state assumptions explicitly and have full proofs in Appendices A and B, except Lemma A.2, whose standard argument is given as a proof sketch (it is not used by the Onion-Sweep decoder).

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Appendix C (C.1 Training Procedure) gives the data source (XML100 with stratified split), full architecture (Table 6), optimizer (Table 7), and hyperparameters (§C.4). Inference settings (single-thread LP, single GPU, 3600s root limit) are stated in §4.1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code (C and Python), the trained checkpoint, the benchmark instances, and run instructions for the reported configurations are released at `https://github.com/henryleo180/SAG-Sep`; the exact-label training set is archived at `https://doi.org/10.5281/zenodo.23539579`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and inference details are documented in Appendix C, including data splits (§C.1), model and optimizer configuration (Tables 6, 7), and threshold/grid hyperparameters (§C.4).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Each cell in the main tables averages over the 10 benchmark instances per size; Appendix D.6 reports paired per-instance win/loss counts, mean differences with standard errors, and worst-case margins for the main comparisons.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Hardware (B200 GPU for training, L4 GPU + single AMD EPYC 9655P thread for evaluation), time limits (3600s root-relaxation cap), are reported in §4.1 and Appendix C.1; per-round separation times are in Appendix D.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work concerns combinatorial optimization for vehicle routing and does not involve human subjects, personal data, or sensitive applications.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper concerns algorithmic improvements to a classical operations-research subroutine (RCC separation in CVRP) and does not have direct societal impacts beyond general logistics efficiency.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases no model that poses misuse risks (a CVRP separator); no safeguards apply.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the baselines and benchmark sources. The released repository is MIT-licensed except for the vendored RouteOpt (GPL-3.0); vendored code and data keep their own license files, e.g., CVRPSEP (CPL-1.0) and the OR-Library CMST instances (MIT).

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The released code and trained checkpoint are documented in the repository README and distributed under the license stated there; the exact-label training set is documented in its archive and released under CC BY 4.0.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used as writing assistants for proofreading, polishing English phrasing. They were not used to generate experimental results, data, or core algorithmic ideas. All technical claims, code, theorems, and analyses are the authors' own.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.