# TOWARDS GENERIC AND EFFICIENT ARCHITECTURES FOR GRAPH NEURAL COMBINATORIAL OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Constructive neural solvers have become a leading paradigm for combinatorial optimization (CO). However, building an architectural framework that is both generic enough across diverse problems without per-problem component redesign and efficient enough for step-wise inference remains an open challenge. To reduce this gap, we propose GECO, a generic and efficient architecture for edge-aware graph neural CO. To reduce computational cost, GECO introduces ScoreMixer-in-Attention (SMA), which follows a head-coupled attention-logit modeling principle: it treats the pre-softmax multi-head logits as a compact node–edge compatibility representation and refines it through a light-weight sub-network, avoiding expensive high-dimensional pairwise feature transformations. To enable generic instantiation, GECO further introduces a learnable warm-up attention layer that initializes node embeddings by aggregating edge information in a task-agnostic manner, together with an instance-preserving normalization framework that provides a principled feature-scaling normalization recipe for CO problems. Experiments on five representative routing and scheduling tasks show that GECO achieves strong solution quality while substantially reducing computational cost compared with state-of-the-art edge-aware architectures. Overall, GECO offers a concrete step toward systematically instantiable architectures for efficient edge-aware graph-based CO.

## 1 INTRODUCTION

Combinatorial Optimization (CO) powers many applications such as logistics, routing, and scheduling. Advances in Neural Combinatorial Optimization (NCO) have shown that neural networks can learn effective approximation solvers directly from data (Bengio et al., 2021), with numerous attention-based architectures proposed for various CO tasks (Kool et al., 2019; Kwon et al., 2020; Kim et al., 2022; Drakulic et al., 2023; Luo et al., 2023). Among them, *constructive neural solvers* have become one of the most widely adopted paradigms, because they provide a simple and extensible interface for new CO problems. A constructive solver builds a solution sequentially by selecting one feasible element at each step, conditioned on the current partial solution. This casts a wide class of CO problems as a sequence of action prediction problems, where the neural model scores feasible actions and the solution is obtained by repeatedly applying this scoring rule (Kool et al., 2019; Kwon et al., 2020; Drakulic et al., 2023; Luo et al., 2023).

Recently, NCO has been moving from specialized solvers toward unified frameworks that apply across diverse CO problems (Zheng et al., 2024; Liu et al., 2024; Zhou et al., 2024; Berto et al., 2024; Drakulic et al., 2025; Zhou et al., 2025b). For constructive solvers, a practical general-purpose architecture should meet two requirements. First, it should efficiently handle explicit pairwise edge features, since many problems, such as the Asymmetric Traveling Salesman Problem (ATSP) or scheduling with operation–machine dependencies, cannot be described by node features alone. Second, it should be *generic*. Once a problem is specified by its node and edge features, feasible actions, and state transitions, the same backbone and components should be instantiated by a fixed procedure, with a separate model still trained for each problem. This motivates our central question: *Can we design a constructive NCO architecture that efficiently handles edge-aware graph representations while remaining generic in its architectural instantiation?*

For the *efficiency* requirement, the major bottleneck lies in the processing of edge features. Existing edge-aware attention modules build a wide hidden representation for every query–key pair.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

GOAL (Drakulic et al., 2025) constructs edge-conditioned query and key vectors for each pair, and MatNet (Kwon et al., 2021) keeps a separate hidden score-mixing activation for each pair and each head. Although expressive, these pairwise hidden tensors substantially increase intermediate storage and computation, and this cost recurs at every construction step. Thus, a practical architecture requires an edge-aware attention mechanism that uses edge information effectively while keeping the per-pair representation small. *Genericity* is typically limited by two components that are usually designed per problem. For node-agnostic problems such as ATSP, existing methods initialize node embeddings with input-agnostic random identifiers (Kwon et al., 2021; Drakulic et al., 2025) or hand-crafted edge-pooling rules (Zhou et al., 2025a; Pan et al., 2025). For input normalization, rules such as scaling demands by vehicle capacity (Kool et al., 2019; Kwon et al., 2020; Drakulic et al., 2025) do not indicate how to normalize features in other formulations. These choices are often treated as implementation details, yet they substantially affect performance and transferability. A generic architecture therefore needs an initialization derived from the available edge information and a normalization recipe derived from the problem formulation.

To address these challenges, we propose **GECO** (*Generic and Efficient CO*), a construction-based architecture for graph-based CO problems with edge features. For efficiency, GECO introduces *ScoreMixer-in-Attention* (SMA), an edge-aware attention mechanism that fuses projected edge features directly into the multi-head attention logits and refines them jointly across heads with a narrow nonlinear residual mixer. For genericity, GECO introduces a simple learnable *warm-up attention layer* that initializes node embeddings by aggregating edge information in a task-agnostic manner, avoiding random identifiers and handcrafted initialization schemes. GECO further formalizes an *instance-preserving normalization* principle and develops a formulation-guided feature-scaling normalizer for Mixed-Integer Linear Programming (MILP)-admitted CO problems, preserving feasibility and relative objective ordering. Empirically, GECO achieves strong performance across five representative graph-based CO problems while substantially reducing computational cost compared with state-of-the-art edge-aware architectures.

## 2 Related Work

**Constructive Neural Solvers**. Wu et al. (2024) classify NCO solvers into four main categories, among which constructive solvers currently constitute the most widely adopted approach. Constructive solvers (Kool et al., 2019; Kwon et al., 2020; 2021; Luo et al., 2023; Xin et al., 2021) incrementally build a solution by selecting one element at each decision step, conditioned on the current partial solution. They can be further categorized into light and heavy decoder types, depending on different allocations of modeling capacity between encoder and decoder components (Huang et al., 2025). Light decoder–based methods (Kool et al., 2019; Kwon et al., 2020) can yield competitive results on small-scale instances, but often exhibit limited generalization when applied to larger problem sizes or different data distributions. More recently, heavy-decoder approaches have received growing attention by placing greater modeling capacity at the decoding stage (Luo et al., 2023; Drakulic et al., 2023; 2025). This design is more closely aligned with the constructive formulation (Drakulic et al., 2023; Huang et al., 2025) and can improve both in-distribution performance and generalization, despite the increased computational cost.

**NCO with Edge Information**. Despite the availability of explicit edge information in many neural combinatorial optimization problems, most existing approaches (Kool et al., 2019; Kwon et al., 2020; Luo et al., 2023; Gao et al., 2024) do not effectively exploit such information. To address this limitation, several studies have attempted to incorporate edge information through modifications of the attention mechanism. A notable example is MatNet (Kwon et al., 2021), which employs a small MLP to combine edge information and attention logits. Similarly, GOAL (Drakulic et al., 2025) injects pairwise edge features into both the query and key projections of the attention mechanism. In a related line of work, Graph Attention Networks have been employed to incorporate edge information in scheduling domain (Wang et al., 2023; Corsini et al., 2024; Song et al., 2022).

## 3 Methodology

The overall framework of GECO is shown in Figure 1. We organize the method around three components. First, Sec. 3.1 introduces ScoreMixer-in-Attention, an edge-aware attention module

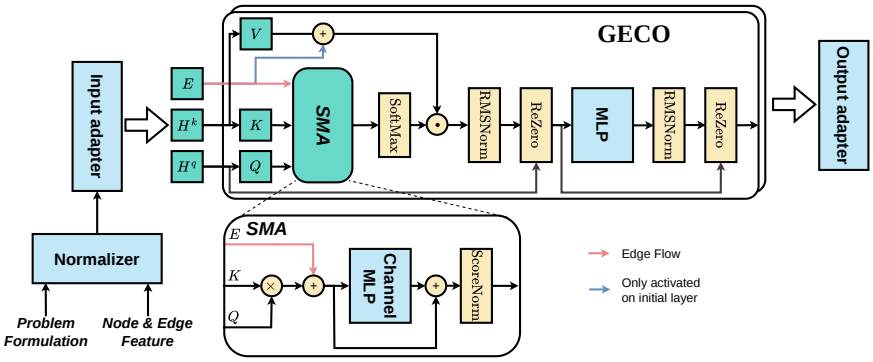


Figure 1: Overview of the GECO architecture. The warm-up attention layer is the first backbone layer and is applied at every construction step.

that performs node–edge interaction in compact attention-score space rather than high-dimensional pairwise feature space. Second, Sec. 3.2 presents a learnable warm-up attention layer, which initializes node embeddings by aggregating edge information through attention itself. Third, Sec. 3.3 develops an instance-preserving feature normalizer, which provides a problem formulation-guided recipe for scaling heterogeneous CO features without changing the underlying optimization instance. Together, these components define a generic and efficient architecture for constructing solutions on diverse edge-aware graph-based CO problems.

### 3.1 ScoreMixer-in-Attention

A central challenge in graph-based NCO is to incorporate edge features into attention in a way that is both expressive and efficient. When edge features condition attention through the scores, each query–key pair is ultimately summarized by its vector of multi-head compatibility logits. This suggests viewing the multi-head attention-score tensor as a compact node–node interaction representation, where each query–key location corresponds to a candidate pairwise interaction and different heads provide multiple compatibility channels for that interaction. Rather than processing each head's edge-conditioned scores independently, this view motivates a head-coupled design principle: *the edge-conditioned multi-head scores are refined jointly across heads*. The remaining question is how to organize this score-level fusion and refinement in a lightweight and stable manner. We draw inspiration from the canonical Transformer layout, which aggregates and refines representations through mixing, residual updates, and normalization (Vaswani, 2017; Yu et al., 2022). However, instead of applying this layout to node or token embeddings, SMA applies it inside attention-score space: it constructs an edge-conditioned score tensor, refines the head-wise compatibility channels through a residual channel mixer, and calibrates the resulting scores before the row-wise softmax.

**The SMA framework.** Let $H_q \in \mathbb{R}^{d\times N}$ and $H_k \in \mathbb{R}^{d\times M}$ denote the query-side and key-side node embeddings, and let $E \in \mathbb{R}^{d_e\times N\times M}$ denote the edge features, where $N$ and $M$ are the numbers of query-side and key-side entities (e.g., $N = M$ nodes in routing), $d$ is the node embedding dimension, and $d_e$ is the edge feature dimension. SMA is composed of three operators:

$$\bar{S} = \mathrm{Mixer}(H_q, H_k, E), \qquad \tilde{S} = \bar{S} + \mathrm{ChannelMixer}(\bar{S}), \qquad S = \mathrm{ScoreNorm}(\tilde{S}), \tag{1}$$

where $\bar{S}, \tilde{S}, S \in \mathbb{R}^{C\times N\times M}$ are multi-head score tensors. The operator $\mathrm{Mixer}(\cdot)$ constructs an initial node-edge-conditioned score representation, $\mathrm{ChannelMixer}(\cdot)$ refines this representation across score channels, and $\mathrm{ScoreNorm}(\cdot)$ calibrates the refined scores. This formulation defines a framework rather than a single fixed module: different choices of these operators can result in edge-aware attention mechanisms with diverse properties, while our goal is to instantiate them with a good balance of expressivity and efficiency.

**An efficient instantiation.** In this work, we instantiate the mixer using a simple additive score-space fusion. Specifically, we compute

$$Q = W_q H_q, \quad K = W_k H_k, \quad B = W_e E, \quad \bar{S} = \frac{Q^\top K}{\sqrt{d_s}} + B, \tag{2}$$

where $Q \in \mathbb{R}^{C\times d_s\times N}$, $K \in \mathbb{R}^{C\times d_s\times M}$, $B, \bar{S} \in \mathbb{R}^{C\times N\times M}$, $W_q, W_k \in \mathbb{R}^{C\times d_s\times d}$, and $W_e \in \mathbb{R}^{C\times d_e}$. Here, $Q^\top K$ denotes head-wise matrix multiplication over the feature dimension. Although such fusion is efficient and simple, it is intrinsically a shallow fusion of node-based compatibility and edge information. To introduce nonlinear interactions across compatibility channels, we instantiate $\mathrm{ChannelMixer}(\cdot)$ as a lightweight residual MLP applied along the head dimension:

$$\tilde{S} = \bar{S} + \mathrm{MLP}_C(\bar{S}) = \bar{S} + W_2\sigma(W_1\bar{S}), \tag{3}$$

where $W_1 \in \mathbb{R}^{rC\times C}$, $W_2 \in \mathbb{R}^{C\times rC}$, $r$ is the expansion ratio, and $\sigma(\cdot)$ is the SiLU activation. The same channel mixer is shared across all query-key locations. Thus, each refined score can depend nonlinearly on the compatibility scores of all heads, while the only pairwise hidden activation has width $rC$. The cost of this refinement is controlled by the channel width ratio $r$; in this work, we use a bottleneck score MLP with $r = 0.5$ for computation and memory efficiency.

**Score calibration.** Unlike token embeddings, the score tensor is fed directly into the softmax, so both the order of its entries and the gaps between them affect the attention distribution. We therefore view $\mathrm{ScoreNorm}(\cdot)$ as a form of *logit calibration*, which differs from conventional feature normalization (Ba et al., 2016; Zhang & Sennrich, 2019). We consider the form $\mathrm{ScoreNorm}(\tilde{S}) = \alpha(\tilde{S}) \cdot (\tilde{S} - \beta(\tilde{S}))$, where $\alpha(\cdot)$ is a multiplicative factor and $\beta(\cdot)$ an additive correction. A systematic comparison of score statistics (Sec. 4.2) shows that (1) the query-axis maximum is the most robust additive correction, and (2) statistic-based multiplicative factors hurt size generalization. For the multiplicative part, we instead use the support size as an external signal. Attention becomes less concentrated as the number of valid keys grows (Qin et al., 2022) (Sec. 4.2), so we adopt the $\log N_k$ scaling of SSMax (Nakanishi, 2025; Chiang & Cholak, 2022):

$$\mathrm{ScoreNorm}(\widetilde{S}) = (\alpha_2 \cdot \log N_k) \odot \left(\widetilde{S} - \alpha_1 \odot \mathrm{Max}_q(\widetilde{S})\right), \tag{4}$$

where $N_k$ is the number of valid key tokens, $\mathrm{Max}_q(\widetilde{S}) \in \mathbb{R}^{C\times 1\times M}$ denotes the maximum over the query dimension, $\alpha_1 \in (0,1)^C$ is a learnable parameter controlling the strength of the statistic-based shift, and $\alpha_2 \in \mathbb{R}^C$ is a learned head-wise gain. The attention weights are then computed as

$$A = \mathrm{Softmax}_k\left(\mathrm{ScoreNorm}(\widetilde{S})\right). \tag{5}$$

The remaining computation follows standard multi-head attention. With value tensor $V = W_v H_k \in \mathbb{R}^{C\times d_v\times M}$, where $d_v$ is the per-head value dimension, the SMA attention (SMAA) output is

$$O = VA^\top \in \mathbb{R}^{C\times d_v\times N}, \qquad \mathrm{SMAA}(H_q, H_k, E) = W_o\,\mathrm{Merge}(O) \in \mathbb{R}^{d\times N}, \tag{6}$$

where $\mathrm{Merge}(\cdot)$ concatenates the $C$ heads, and $W_o \in \mathbb{R}^{d\times Cd_v}$. The time, memory, and expressivity trade-offs of SMA relative to MatNet and GOAL are analyzed in Appendix C.

### 3.2 Warm-up Layer

Constructive neural solvers require node embeddings that reflect the underlying graph structure. However, when edge information conditions only the attention scores, it changes the weights *within* the softmax but not the values being aggregated. If node features are identical or absent (e.g., ATSP, whose input is only a cost matrix), all value vectors coincide, $v_j = v_0$, so every output $\sum_j A_{ij}v_j = v_0$ is identical regardless of the edge-dependent weights $A_{ij}$. Existing methods address this in two ways: (i) assigning random node identifiers to break symmetry (Kwon et al., 2021; Drakulic et al., 2025), or (ii) injecting edge information into node embeddings via problem-specific rules (Zhou et al., 2025a; Son et al., 2025). The former introduces uninformative artificial randomness, while the latter typically relies on hand-crafted heuristics (Zhou et al., 2025a; Pan et al., 2025). Such heuristics may not be applicable across edge attributes or problem types, and often discard the correspondence between individual edges and nodes.

Instead, we propose a *warm-up layer* that uses attention itself as a learnable and generic edge information aggregator. The key idea is to allow edge information to flow *outside* the softmax by incorporating it into the value vectors. Concretely, the warm-up layer computes the attention tensor

$A \in \mathbb{R}^{C\times N\times M}$ in the same way as SMA and adds a scalar edge value per head to each value vector. For a single head, with the head index omitted, the output for query node $i$ is

$$o_i^{\mathrm{wu}} = \sum_j A_{ij}\big(v_j + b_{ij}^v \mathbf{1}_{d_v}\big) = \sum_j A_{ij} v_j + \mathbf{1}_{d_v} \sum_j A_{ij}\, b_{ij}^v, \tag{7}$$

where $v_j$ is the standard value vector of key $j$, $\mathbf{1}_{d_v}$ is the all-ones vector, and the scalar edge values of all heads are collected in $B^v = W_{v,e}E \in \mathbb{R}^{C\times N\times M}$ with $W_{v,e} \in \mathbb{R}^{C\times d_e}$. The second term makes the embedding of node $i$ depend on its own edge attributes even when all $v_j$ coincide. It cannot, however, separate nodes that are equivalent under a symmetry of the instance. In tensor form, $O^{\mathrm{wu}} = VA^\top + \mathrm{Broad}_{d_v}\big(\mathrm{Sum}_M(A \odot B^v)\big)$. The layer therefore operates only on $C \times N \times M$ tensors and never forms a pairwise value tensor of size $C \times d_v \times N \times M$ (Appendix B). The remaining computation is identical to SMAA (Eq. 6).

This design allows the model to learn how to aggregate edge information into node embeddings in a data-driven way, without relying on hand-crafted rules. In particular, the model can learn a sparse attention score pattern to recover nearest-neighbor-like behavior (Zhou et al., 2025a) when beneficial, while remaining flexible across different problems and edge attributes. To balance efficiency, the warm-up attention is used only as the first backbone layer. This layer is re-applied at every construction step, so all subsequent layers operate on edge-aware embeddings.

### 3.3 A Unifying Feature Normalizer

Once a CO problem is represented by heterogeneous numerical features, a generic architecture needs a unified way to normalize them before processing. Existing constructive NCO solvers often rely on problem-specific heuristics, such as normalizing CVRP demands by vehicle capacity, but such rules do not directly transfer to new objectives and constraints. Therefore, feature normalization becomes another degree of freedom that prevents the architecture from being generic. As shown in Table 1, normalization is a necessary element: z-score standardization of MatNet's distance matrix leaves the in-distribution result unchanged but drastically improves size generalization. We therefore ask how to normalize heterogeneous CO features while preserving the optimization problem represented by the instance, and address this through an instance-preserving normalization principle.

**Instance-preserving principle.** Normalization should preserve two core properties: (i) solution feasibility and (ii) relative objective ordering. Formally, given a transformation $T$ that produces normalized features $T(x)$ from instance parameters $x$, we call $T$ *instance-preserving* if the transformed representation is sufficient to recover feasibility and objective ordering: there exist functions $\widehat{\mathcal{F}}$ and $\widehat{J}$ such that, for any candidate solution $\pi$ and any $\pi_1, \pi_2 \in \mathcal{F}(x)$,

$$\begin{aligned} \pi \in \mathcal{F}(x) &\iff \pi \in \widehat{\mathcal{F}}(T(x)), \\ J(\pi_1; x) \le J(\pi_2; x) &\iff \widehat{J}(\pi_1; T(x)) \le \widehat{J}(\pi_2; T(x)). \end{aligned} \tag{8}$$

Table 1: Effects of input normalization on MatNet.

| | ATSP100 Gap | ATSP200 Gap |
|---|---|---|
| **MatNet** | 3.35% | 142.41% |
| **+ z-norm** | 3.38% | 21.76% |
| **+ tanh** | 4.28% | 142.71% |

For example, scaling all pairwise distances by a positive scalar preserves additive routing problems such as TSP and CVRP. Subtracting a global offset $b$ from every edge cost preserves TSP but not CVRP, because a CVRP solution $\pi$ with $k(\pi)$ routes over $n$ customers has the shifted cost $J(\pi) - b\,[n + k(\pi)]$, which can reorder solutions with different numbers of routes. Designing instance-preserving normalizations for arbitrary CO problems is non-trivial[1]. We therefore represent a solution $\pi$ by the action sequence of the constructive solver and assume that, for any fixed $\pi$, rescaling all parameters of one coupled group (e.g., all edge weights, or all demands together with the capacity) by $\lambda > 0$ leaves constraint satisfaction unchanged and either leaves the objective unchanged or multiplies it by $\lambda$. This holds for *MILP-admitted* problems, whose instance data enter a Mixed-Integer Linear Programming (MILP) formulation linearly as coefficients, right-hand sides, or bounds, and for positively homogeneous quantities such as the JSSP makespan and CVRPTW arrival times. It covers all five problems considered in this work.

[1]While element-wise invertible transforms (e.g., $\tanh$, sigmoid) are theoretically instance-preserving, they can perform poorly (Table 1), likely due to saturation and loss of relative scale.

We turn this principle into a normalization recipe that controls the magnitude of heterogeneous features. Under the assumption above, the instance is preserved whenever every constraint is multiplied by a positive factor and the objective by a common positive factor that does not depend on the candidate solution, as in $\sum_{i\in R} d_i \le Q \Leftrightarrow \sum_{i\in R} d_i/s \le Q/s$ for $s > 0$. This is guaranteed if *features that appear additively in the same objective term or constraint expression, including right-hand sides and bounds, share the same positive scaling factor*, which gives a two-step procedure. **Step 1: Feature Grouping.** For the input features $\{x_i\}_{i=1}^{K}$ of an instance (e.g., node vectors in $\mathbb{R}^N$ and edge matrices in $\mathbb{R}^{N\times M}$), we set $x_i \sim x_j$ if their entries appear additively in the same objective or constraint expression, and partition the features into groups $\mathcal{G}_1, \ldots, \mathcal{G}_P$ by the transitive closure of $\sim$. Features in a group share the same "physical unit" and must be scaled identically. **Step 2: Unified Scaling.** For each group $\mathcal{G}_p$, we compute a scalar statistic $u_i$ of each feature, aggregate the statistics into a shared factor $s_p > 0$, and divide each feature by it:

$$u_i = \mathrm{Stat}(x_i), \qquad s_p = \mathrm{Agg}\left(\{u_i : x_i \in \mathcal{G}_p\}\right), \qquad \widetilde{x}_i = x_i/s_p \quad \forall x_i \in \mathcal{G}_p, \tag{9}$$

where $\mathrm{Agg}(\cdot)$ is a pooling operation. Overall, this gives a deterministic normalization recipe once the problem formulation and feature interface are specified. In this work, we implement $\mathrm{Stat}$ as the Root Mean Square (RMS) and $\mathrm{Agg}$ as the mean. The choice of $\mathrm{Stat}$ and $\mathrm{Agg}$ does not affect instance preservation as long as the resulting factor is finite, positive, and shared within each group. It only determines which notion of group magnitude is normalized (Appendix F). In constructive solvers, the state changes after each decision, so the scale of state-dependent features can also change along the decoding trajectory. We therefore re-apply the normalizer at each construction step to normalize the current subproblem state.

## 4 Experiments

**Training tasks.** We evaluate GECO on five edge-based graph-based CO problems: ATSP, CVRP, CVRPTW, OP, and JSSP. Following common protocol for heavy-decoder based constructive models (Drakulic et al., 2023; Luo et al., 2023; Drakulic et al., 2025; Luo et al., 2025b), we utilize supervised learning to train GECO. GECO can also be trained without expert labels, and Appendix I reports results with label-free self-improvement. For each problem, we generate $1,000,000$ random training instances and obtain high-quality reference solutions using established OR solvers. Concretely, we use LKH (Helsgaun, 2017) for ATSP and CVRP, HGS (Vidal, 2022) for CVRPTW, EA4OP (Kobeaga et al., 2018) for OP, and OR-Tools (Perron & Didier) for JSSP. Training instances have size $N = 100$ for routing problems and $10 \times 10$ for JSSP. We follow GOAL (Drakulic et al., 2025) to generate the train and test instances.

**Setting of GECO.** GECO is trained separately on each of the five tasks for 400 epochs on a single NVIDIA RTX 3090 GPU. The network has 9 layers, the first of which is the warm-up attention layer. The optimizer, model dimensions, and remaining hyperparameters are listed in Appendix B.

**Baselines.** We consider a set of representative constructive neural models as baselines, including AM (Kool et al., 2019), MDAM (Xin et al., 2021), POMO (Kwon et al., 2020), Sym-NCO (Kim et al., 2022), BQ-NCO (Drakulic et al., 2023), LEHD (Luo et al., 2023), ICAM (Zhou et al., 2025a), MatNet (Kwon et al., 2021), GOAL (Drakulic et al., 2025), GAT (Veličković et al., 2017), and L2D (Zhang et al., 2020). All baseline models are trained in a single-task setting, and all of them are trained only on the training size except ICAM, which is trained on sizes 100–500, and L2D, which is trained separately at each test size.

**Evaluation settings.** All the results are reported under a greedy rollout, except for models trained with multi-trajectory strategies (Kwon et al., 2020; Kim et al., 2022; Kwon et al., 2021), for which multi-trajectory inference is retained at test time. ICAM is evaluated with a single trajectory and without augmentation. Inference time for all neural solvers is measured using the same batch size on a single NVIDIA RTX 3090 GPU.

### 4.1 Main Results

Table 2 reports the main comparison across five graph-based CO problems. Overall, GECO achieves strong performance across both routing and scheduling tasks, while maintaining substantially lower inference time than existing edge-aware heavy-decoder architectures. On ATSP, where asymmetric

Table 2: Main results on 5 graph-based CO problems.

| Method | ATSP100 Gap | ATSP100 Time | ATSP200 Gap | ATSP200 Time | ATSP500 Gap | ATSP500 Time | ATSP1000 Gap | ATSP1000 Time |
|---|---|---|---|---|---|---|---|---|
| LKH | 0.00% | 3m | 0.00% | 1m | 0.00% | 5m | 0.00% | 10m |
| GAT⋆ | 1.15% | 1m | 1.66% | 7s | 1.87% | 41s | 2.63% | 3m |
| MatNet⋆ | 3.35% | 31s | 142.41% | 1s | - | | - | |
| MatNet ×128⋆ | 0.97% | 2h | 131.42% | 6m | - | | - | |
| BQ-NCO⋆ | 1.63% | 2m | 1.90% | 10s | 4.29% | 1m | 8.09% | 6m |
| ICAM⋆‡ | 5.15% | – | 6.74% | – | 10.71% | – | 21.59% | – |
| GOAL⋆ | 0.56% | 10m | 0.88% | 58s | 1.25% | 6m | 1.67% | 31m |
| GECO⋆ | **0.31%** | 2m | **0.55%** | 9s | **0.81%** | 59s | **0.87%** | 5m |
| Method | CVRP100 Gap | CVRP100 Time | CVRP200 Gap | CVRP200 Time | CVRP500 Gap | CVRP500 Time | CVRP1000 Gap | CVRP1000 Time |
| LKH | 0.00% | 12m | 0.00% | 2h | 0.00% | 5h | 0.00% | 7h |
| AM | 7.11% | 1s | 10.16% | 1s | 17.2% | 2s | 72.95% | 3s |
| MDAM | 4.84% | 1s | 7.54% | 2s | 12.94% | 15s | 32.25% | 1m |
| POMO | **1.21%** | 1s | 5.94% | 2s | 31.36% | 17s | 284.58% | 1m |
| Sym-NCO | 3.33% | 2s | 8.42% | 2s | 37.58% | 1m | 481.53% | 30m |
| BQ-NCO | 2.79% | 2s | 2.81% | 8s | **2.73%** | 2m | 5.71% | 12m |
| LEHD | 3.61% | 1s | 3.55% | 5s | 3.04% | 1m | 5.14% | 8m |
| ICAM⋆‡ | 3.64% | 1s | 2.84% | 1s | 1.86% | 2s | 5.22% | 18s |
| GOAL⋆ | 2.41% | 9s | 2.86% | 1m | 4.27% | 15m | 7.27% | 2h |
| GECO⋆ | 2.37% | 3s | **2.37%** | 12s | 3.36% | 2m | **4.79%** | 22m |
| Method | CVRPTW100 Gap | CVRPTW100 Time | CVRPTW200 Gap | CVRPTW200 Time | CVRPTW500 Gap | CVRPTW500 Time | CVRPTW1000 Gap | CVRPTW1000 Time |
| HGS | 0.00% | 10m | 0.00% | 26m | 0.00% | 2h | 0.00% | 2h |
| GOAL⋆ | 4.66% | 9s | 7.19% | 1m | 18.48% | 15m | 26.34% | 2h |
| GECO⋆ | **4.21%** | 2s | **5.37%** | 12s | **5.55%** | 2m | **2.26%** | 15m |
| Method | OP100 Gap | OP100 Time | OP200 Gap | OP200 Time | OP500 Gap | OP500 Time | OP1000 Gap | OP1000 Time |
| EA4OP | 0.00% | 1m | 0.00% | 3m | 0.00% | 11m | 0.00% | 45m |
| AM | 5.35% | 1s | 11.05% | 1s | 22.29% | 2s | 30.73% | 2s |
| MDAM | 2.88% | 35s | 5.55% | 5s | 18.31% | 29s | 30.05% | 45s |
| SymNCO | 2.03% | 3s | 6.60% | 17s | 19.81% | 12m | 31.79% | 2h |
| BQ-NCO | 0.22% | 2s | 1.02% | 7s | 7.32% | 1m | 13.12% | 7m |
| GOAL⋆ | **-0.34%** | 8s | 1.78% | 1m | 6.69% | 11m | 16.57% | 1h |
| GECO⋆ | **-0.34%** | 2s | **0.27%** | 9s | **2.98%** | 2m | **5.11%** | 15m |
| Method | JSSP 10x10 Gap | JSSP 10x10 Time | JSSP 15x15 Gap | JSSP 15x15 Time | JSSP 20x20 Gap | JSSP 20x20 Time | JSSP 30x20 Gap | JSSP 30x20 Time |
| ORTools | 0.00% | 47s | 0.00% | 2m | 0.00% | 3h | 0.00% | 80h |
| L2D⋆† | 22.3% | 1s | 26.7% | 10s | 29.0% | 24s | 29.2% | 46s |
| GOAL⋆ | 2.51% | 10s | 6.89% | 2m | 12.08% | 7m | 102.01% | 22m |
| GECO⋆ | **2.14%** | 5s | **6.00%** | 43s | **7.13%** | 2m | **29.12%** | 4m |

**Note.** ⋆ denotes architectures with edge information. † L2D model is trained on each size respectively. ‡ICAM is trained on sizes 100–500 and is excluded from bolding, which marks the best model trained only on the training size. "–" denotes results that were not measured.

edge costs are central to the problem, GECO clearly outperforms GAT, MatNet, BQ-NCO, and GOAL across all tested sizes. Although GECO requires longer inference time than lightweight baselines such as GAT, the additional cost brings substantial improvements in solution quality. Meanwhile, compared with GOAL, the state-of-the-art generic edge-aware baseline, GECO achieves both better solution quality and much lower inference time, reducing the gap from $1.67\%$ to $0.87\%$ on ATSP1000 while reducing inference time from $31$ minutes to $5$ minutes. This demonstrates that GECO improves the quality-efficiency trade-off for edge-aware constructive inference. GECO also obtains lower ATSP gaps than ICAM at every size, although ICAM is trained on sizes up to 500. Similar trends are observed on the remaining tasks. On Euclidean CVRP, GECO outperforms LEHD, GOAL, and ICAM at three of the four sizes. On CVRP500, LEHD and ICAM obtain lower gaps, and this size lies within the training range of ICAM. Together, these results show that GECO provides a robust, generic, and efficient architecture across heterogeneous graph-based CO tasks, with the largest advantage on problems whose pairwise attributes are intrinsic.

### 4.2 Ablation Study

**Effectiveness of SMA.** We isolate the attention module with two controlled replacements (Table 3). GOAL-SMA replaces the attention module of GOAL with SMA, and GECO-MatNet replaces SMA in GECO with MatNet attention while keeping all other components, including the support-size factor of Eq. 4, without which it performs poorly. SMA improves GOAL at every ATSP size and

Table 3: Attention ablation on ATSP and CVRP (gap, %). "–" denotes results that were not measured.

| | ATSP | | | | CVRP | | | |
|---|---|---|---|---|---|---|---|---|
| Method | 100 | 200 | 500 | 1000 | 100 | 200 | 500 | 1000 |
| GOAL | 0.56 | 0.88 | 1.25 | 1.67 | 2.41 | 2.86 | 4.27 | 7.27 |
| GOAL-SMA | 0.36 | 0.66 | 0.90 | 1.00 | – | – | – | – |
| GECO-MatNet | 1.01 | 1.20 | 1.26 | 1.19 | 2.60 | 2.82 | 4.43 | 6.05 |
| GECO | **0.31** | **0.55** | **0.81** | **0.87** | **2.37** | **2.37** | **3.36** | **4.79** |

reduces the ATSP1000 gap from $1.67\%$ to $1.00\%$. GECO outperforms GECO-MatNet on both ATSP and CVRP while being $1.8$–$2.1\times$ faster with $2.9$–$4.2\times$ lower peak memory (Sec. 4.3). These controlled comparisons support refining the multi-head score tensor jointly as a compact node–edge compatibility representation.

**Impact of Attention Score Calibrators.** We ablate the two parts of ScoreNorm. For the additive correction, Table 9 in the Appendix compares score statistics computed along different axes on ATSP. Several variants that improve ATSP100 degrade substantially on ATSP500, and statistic-based multiplicative factors are unreliable under size extrapolation. The query-axis maximum shift gives the best ATSP500 result and remains competitive at ATSP100, so we use it in ScoreNorm. For the multiplicative factor, the average row-wise maximum attention probability drops as the problem size grows (Fig. 2 in the Appendix). Removing the $\log N_k$ factor hurts size generalization (Table 4), increasing the CVRPTW1000 gap from $2.26\%$ to $6.58\%$ and the ATSP1000 gap from $0.87\%$ to $1.12\%$. The full analysis is given in Appendix D.

Table 4: Effect of the support-size factor $\log N_k$ (gap, %).

| ATSP | 100 | 200 | 500 | 1000 |
|---|---|---|---|---|
| GECO w/o $\log N_k$ | 0.42 | 0.66 | 0.97 | 1.12 |
| GECO | **0.31** | **0.55** | **0.81** | **0.87** |
| CVRPTW | 100 | 200 | 500 | 1000 |
| GECO w/o $\log N_k$ | **4.08** | 5.63 | 7.87 | 6.58 |
| GECO | 4.21 | **5.37** | **5.55** | **2.26** |

Table 5: Node initialization ablation (gap, %).

| ATSP | 100 | 200 | 500 | 1000 |
|---|---|---|---|---|
| GECO-KNN | 0.44 | 0.69 | 0.82 | 0.91 |
| GECO-Random | **0.23** | **0.52** | 0.89 | 0.98 |
| GECO | 0.31 | 0.55 | **0.81** | **0.87** |
| OP | 100 | 200 | 500 | 1000 |
| GECO-KNN | −0.27 | 0.74 | 5.68 | 9.71 |
| GECO-Random | −0.16 | 0.65 | 4.96 | 10.58 |
| GECO | **−0.34** | **0.27** | **2.98** | **5.11** |

**Effectiveness of the Warm-up Layer.** We compare the proposed warm-up layer with two node-initialization alternatives: KNN-based distance packing and random node identifiers (Son et al., 2025; Zhou et al., 2025a). For KNN, we concatenate the distances to the nearest $100$ nodes and project them into the embedding space. For random initialization, we assign each node a random scalar identifier as additional node feature (Drakulic et al., 2025). Table 5 shows that the warm-up layer provides the most robust performance under size generalization. On ATSP, random identifiers are slightly better at the training size, but degrade more on larger instances, while GECO achieves the best result on ATSP500 and ATSP1000. The advantage is more pronounced on OP, where GECO substantially outperforms both KNN and random identifiers at larger sizes, reducing the gap to $2.98\%$ on OP500 and $5.11\%$ on OP1000, see Appendix E for more analysis. These results suggest that learnable edge aggregation in the warm-up layer provides a more transferable initialization than either hand-crafted local edge packing or input-agnostic random identifiers.

**On the Design Choices of Feature Normalizer.** Table 11 in the Appendix varies the normalization frequency and the $\mathrm{Stat}$–$\mathrm{Agg}$ operators on CVRP and OP, two problems with heterogeneous feature interactions. One-shot normalization computes the scaling factors from the initial instance only, while step-wise normalization re-applies the normalizer to the current construction state at each decoding step. Step-wise normalization gives more robust size generalization on both problems and reduces the OP1000 gap from over $10\%$ to around $5\%$, because constructive decoding produces a sequence of subproblems whose feature scales change along the trajectory. It also makes the results much less sensitive to the choice of $\mathrm{Stat}$ and $\mathrm{Agg}$, whereas the one-shot variants vary widely, especially on OP.

Table 6: Inference-time resource consumption on ATSP. We report wall-clock time and peak GPU memory on a single RTX 3090.

| Method | ATSP100 | | ATSP200 | | ATSP500 | | ATSP1000 | |
|---|---|---|---|---|---|---|---|---|
| | Time (s) | Mem (MiB) | Time (s) | Mem (MiB) | Time (s) | Mem (MiB) | Time (s) | Mem (MiB) |
| GOAL | 603 | 4944 | 58 | 18498 | 349 | 14426 | 1838 | 14398 |
| GECO-MatNet | 212 | 1928 | 19 | 6522 | 113 | 5098 | 596 | 5070 |
| GECO | 112 | 672 | 9 | 1550 | 59 | 1242 | 337 | 1222 |

**Explicit Edges versus Coordinates on Euclidean Routing.** On Euclidean CVRP, the distance matrix carries no information beyond the node coordinates. We therefore compare GECO with a coordinate-based variant that receives the same information through node features and node-only attention (Appendix H). The edge-based GECO reduces the gap from $3.22\%$ to $2.37\%$ on CVRP100 and from $3.09\%$ to $2.37\%$ on CVRP200, and both variants perform similarly at larger sizes. Explicit pairwise modeling therefore helps near the training scale even when edges are recoverable from nodes. We also observe a runtime trade-off. The inference time of GECO grows faster than that of the node-based BQ-NCO from CVRP500 to CVRP1000 (2m to 22m, versus 2m to 12m in Table 2). The two architectures differ in several respects, so this comparison does not isolate the cost of explicit edge inputs. On problems such as ATSP, whose pairwise costs cannot be recovered from node features, an edge-based interface is necessary.

### 4.3 Efficiency Analysis

We finally evaluate whether GECO improves the computational efficiency of edge-aware constructive inference. Table 6 reports the inference time and peak GPU memory on ATSP, and Appendix G reports the FLOPs and training cost (Tables 12 and 13). Overall, GECO consistently reduces the resource consumption of edge-intensive decoder architectures. On ATSP, GECO is $5.4$–$6.4\times$ faster than GOAL and uses $7.4$–$11.9\times$ less peak GPU memory under the same batch size. Compared with GECO-MatNet, which replaces SMA with MatNet-style score mixing, GECO is $1.8$–$2.1\times$ faster and uses $2.9$–$4.2\times$ less memory. The FLOPs analysis further confirms that the gain comes from the proposed score-space design: at $n = 1000$, GECO reduces total single-step FLOPs from $3.0 \times 10^{11}$ to $4.8 \times 10^{9}$ compared with GOAL, and reduces attention FLOPs from $3.3 \times 10^{10}$ to $4.0 \times 10^{8}$. GECO also lowers training cost on CVRP, reducing training time from 230 hours to 74.6 hours and peak memory from 17GB to 5.3GB measured with the same batch size. These results support the main efficiency argument: edge-aware interaction can be modeled in compact attention-score space without the high computational and memory overhead of high-dimensional pairwise feature transformations.

## 5 Conclusion

In this paper, we studied how to build constructive neural solvers that are both generic across graph-based CO problems and efficient under edge-aware step-wise inference. The main insight is that edge-aware interaction need not be performed through expensive high-dimensional pairwise feature transformations: by moving the interaction into compact attention-score space, SMA provides an efficient way to refine node-edge compatibility while preserving strong modeling capacity. Complementarily, the warm-up layer and instance-preserving normalizer reduce two common sources of problem-specific redesign. Experiments across five routing and scheduling problems show that these design choices lead to consistent improvements in solution quality, size generalization, and computational efficiency. More broadly, our results suggest that many performance gaps in generic NCO solvers come not only from insufficient model capacity, but also from under-specified architectural components whose effects become amplified under problem and size transfer. GECO provides one step toward systematically instantiable neural solvers for edge-aware graph-based CO. One limitation is that although SMA substantially reduces the cost of edge-aware attention compared with existing methods, GECO still operates on dense pairwise score tensors. Scaling to larger graphs may require combining the proposed principle with sparse candidate selection, decomposition, or hierarchical inference, we leave such exploration to future work. Another important direction for future work is to further characterize which architectural biases remain transferable across broader CO families, and where problem-specific structure must still be introduced.

## Ethics Statement

This work studies neural architectures for combinatorial optimization on synthetic routing and scheduling benchmarks. It involves no human subjects, personal data, or sensitive information. We are not aware of direct negative societal impacts beyond those generally associated with improved optimization tools in logistics and scheduling.

## Reproducibility Statement

The GECO architecture is specified in Sec. 3.1–3.3, with complete layer-level details in Appendix B. The formulations of all five problems are given in Table 7. Training and test instances are generated following the protocol of GOAL (Drakulic et al., 2025), reference solutions are obtained with the solvers listed in Sec. 4, and all training hyperparameters are reported in Appendix B. The code will be released upon publication.

## AI Use Statement

In this work, we used generative AI tools for editing and polishing the writing of this manuscript. We have not used generative AI tools for other activities requiring disclosure under the ICLR 2027 AI Policy for Authors. The formulation of the research problem, development of the method, experimental design, analysis of results, and interpretation of findings are entirely the authors' own work. We take full responsibility for the content of this paper.

## References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pp. 1352–1361. PMLR, 2021.

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.

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

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. *arXiv preprint arXiv:2202.12172*, 2022.

Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. *Advances in Neural Information Processing Systems*, 37:105528–105551, 2024.

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

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

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

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems. *arXiv preprint arXiv:2503.00753*, 2025.

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

Gorka Kobeaga, María Merino, and Jose A Lozano. An efficient evolutionary algorithm for the orienteering problem. *Computers & Operations Research*, 90:42–59, 2018.

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

Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34:5138–5149, 2021.

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

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

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

Fu Luo, Xi Lin, Mengyuan Zhong, Fei Liu, Zhenkun Wang, Jianyong Sun, and Qingfu Zhang. Learning to insert for constructive neural vehicle routing solver. *arXiv preprint arXiv:2505.13904*, 2025b.

Ken M Nakanishi. Scalable-softmax is superior for attention. *arXiv preprint arXiv:2501.19399*, 2025.

Wenzheng Pan, Hao Xiong, Jiale Ma, Wentao Zhao, Yang Li, and Junchi Yan. UniCO: On unified combinatorial optimization via problem reduction to matrix-encoded general TSP. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yEwakMNIex.

Laurent Perron and Frédéric Didier. Cp-sat. URL https://developers.google.com/optimization/cp/cp_solver/.

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

Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 7025–7041, 2022.

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

Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19 (2):1600–1610, 2022.

A Vaswani. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. Flexible job shop scheduling via dual attention network-based reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3091–3102, 2023.

Xuan Wu, Di Wang, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L Maskell, and You Zhou. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *arXiv preprint arXiv:2406.00415*, 2024.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 12042–12049, 2021.

Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10819–10829, 2022.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33:1621–1632, 2020.

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. In *Advances in Neural Information Processing Systems*, 2024.

Changliang Zhou, Xi Lin, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. ICAM: Rethinking instance-conditioned adaptation in neural vehicle routing solver, 2025a. URL https://openreview.net/forum?id=gyTkfVYL45.

Changliang Zhou, Canhong Yu, Shunyu Yao, Xi Lin, Zhenkun Wang, Yu Zhou, and Qingfu Zhang. Urs: A unified neural routing solver for cross-problem zero-shot generalization. *arXiv preprint arXiv:2509.23413*, 2025b.

Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Chi Xu. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *International Conference on Machine Learning*, 2024.

# Towards Generic and Efficient Architectures for Graph Neural Combinatorial Optimization (Appendix)

## A PRELIMINARY

### A.1 FIVE TASKS OF CO

We study five graph-based combinatorial optimization (CO) problems across routing and scheduling: the Asymmetric Traveling Salesman Problem (ATSP), Capacitated Vehicle Routing Problem (CVRP), Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), Orienteering Problem (OP), and Job-Shop Scheduling Problem (JSSP). The detailed problem formulations are summarized in Table 7.

Table 7: Unified problem definitions of five graph-based combinatorial optimization problems.

| **Problem** | **Given** | **Constraints** | **Minimize** |
|---|---|---|---|
| ATSP | 1) Asymmetric cost matrix | 1) Each node is visited exactly once;<br>2) Forms a single Hamiltonian cycle | Total routing cost |
| CVRP | 1) Depot coordinates;<br>2) Customers coordinates;<br>3) Customer demands;<br>4) Vehicle capacity | 1) Each customer is visited exactly once;<br>2) Each route starts and ends at the depot;<br>3) Total demand on each route does not exceed vehicle capacity | Total routing cost |
| CVRPTW | 1) CVRP Features;<br>2) Time window for each customer;<br>3) Service time | 1) CVRP constraints;<br>2) Serve customers within their time windows | Total routing cost |
| OP | 1) Nodes with rewards;<br>2) Edge costs;<br>3) Travel budget | 1) Each node is visited at most once;<br>2) Total travel cost does not exceed the budget | Negative collected reward |
| JSSP | 1) Operation execution time;<br>2) Job availability time;<br>3) Machine availability time | 1) Operations of the same job must follow the predefined order<br>2) Each machine can process at most one operation at a time<br>3) Once started, an operation cannot be interrupted | Makespan |

### A.2 FORMULATION OF CONSTRUCTIVE NEURAL SOLVERS

Constructive neural solvers address CO problems by incrementally constructing a solution through a sequence of decisions. Given a problem instance $\mathcal{I}$, a solution is represented as an ordered sequence of actions

$$\pi = (a_1, a_2, \ldots, a_T), \tag{10}$$

where each action $a_t$ corresponds to selecting a feasible element (e.g., a node, customer, or operation) at step $t$. At each step $t$, the model conditions on the current construction state $s_t$, which encodes the partial solution through feasibility information and problem-specific attributes (e.g., remaining vehicle

capacity), and assigns a score to each feasible action. The next action is selected by normalizing these scores over the feasible action set:

$$p(a_t \mid s_t, \mathcal{I}) = \mathrm{softmax}_{a \in \mathcal{A}(s_t)}\big(f_\theta(s_t, \mathcal{I}, a)\big), \tag{11}$$

where $f_\theta(\cdot)$ denotes a parameterized scoring function and $\mathcal{A}(s_t)$ denotes the set of feasible actions in state $s_t$. We write $\mathcal{A}$ for feasible *actions* to distinguish it from the set $\mathcal{F}(x)$ of feasible *solutions* used in Sec. 3.3. The construction state is updated deterministically according to the problem dynamics after each action selection,

$$s_{t+1} = \tau(s_t, a_t; \mathcal{I}), \tag{12}$$

where $\tau(\cdot)$ denotes a problem-specific state transition function. This update reflects the evolution of the partial solution and implicitly captures all dependencies induced by previously selected actions. Feasibility constraints are enforced by masking infeasible actions before normalization.

### A.3 Edge in Attention

We introduce two commonly used approaches for incorporating edge information into the attention mechanism in NCO. We follow the notation of Sec. 3.1, where $H_q \in \mathbb{R}^{d \times N}$ and $H_k \in \mathbb{R}^{d \times M}$ are the query-side and key-side node embeddings and $E \in \mathbb{R}^{d_e \times N \times M}$ are the edge features, with $e_{ij} \in \mathbb{R}^{d_e}$ the feature of pair $(i, j)$. Both designs first project the node embeddings into $C$ heads,

$$Q = W_q H_q \in \mathbb{R}^{C \times d_s \times N}, \qquad K = W_k H_k \in \mathbb{R}^{C \times d_s \times M}, \tag{13}$$

with $W_q, W_k \in \mathbb{R}^{C \times d_s \times d}$ and per-head query/key dimension $d_s$.

**MatNet attention.** MatNet (Kwon et al., 2021) incorporates edge information directly at the attention-logit level by mixing node-based dot-product scores with edge features through head-specific MLPs. For each head $c \in \{1, \dots, C\}$, MatNet first computes the standard scaled dot-product logits

$$S^{\mathrm{node},(c)} = \frac{(Q^{(c)})^\top K^{(c)}}{\sqrt{d_s}} \in \mathbb{R}^{N \times M}. \tag{14}$$

To inject edge information, each scalar node-based logit $S_{ij}^{\mathrm{node},(c)}$ is concatenated with the corresponding edge feature $e_{ij}$, and the resulting $(1 + d_e)$-dimensional vector is passed through a head-specific two-layer MLP:

$$\tilde{S}_{ij}^{(c)} = w_2^{(c)} \phi\left( W_1^{(c)} \begin{bmatrix} S_{ij}^{\mathrm{node},(c)} \\ e_{ij} \end{bmatrix} \right), \tag{15}$$

where $W_1^{(c)} \in \mathbb{R}^{h_s \times (1+d_e)}$, $w_2^{(c)} \in \mathbb{R}^{1 \times h_s}$, $\phi(\cdot)$ denotes the ReLU activation, and $h_s$ is the hidden dimension of the score-mixing MLP. Stacking the mixed logits over all heads gives $\tilde{S} \in \mathbb{R}^{C \times N \times M}$, which is then normalized by a softmax over the key dimension. However, because the score-mixing MLP is applied separately for each head and each query–key pair, a direct vectorized implementation needs to materialize an intermediate activation tensor of size $C \times h_s \times N \times M$. Compared with vanilla attention logits of size $C \times N \times M$, this introduces an $h_s$-fold increase in intermediate score-level storage. Since MatNet typically sets $h_s = 16$, this design can lead to substantial memory overhead, especially for large graphs.

**GOAL attention.** GOAL (Drakulic et al., 2025) incorporates edge features into the attention mechanism by augmenting the query and key representations with edge-dependent terms. It projects the edge features into query-side and key-side augmentation tensors:

$$E^q = W_{e,q} E, \qquad E^k = W_{e,k} E, \tag{16}$$

where $W_{e,q}, W_{e,k} \in \mathbb{R}^{C \times d_s \times d_e}$. Therefore, $E^q, E^k \in \mathbb{R}^{C \times d_s \times N \times M}$. The attention logits are then computed by an edge-augmented dot product. For each head $c$ and query–key pair $(i, j)$,

$$S_{ij}^{(c)} = \frac{1}{\sqrt{d_s}} \left\langle Q_{:,i}^{(c)} + E_{:,i,j}^{q,(c)},\, K_{:,j}^{(c)} + E_{:,i,j}^{k,(c)} \right\rangle, \tag{17}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product over the feature dimension. Stacking the logits over all heads gives $S \in \mathbb{R}^{C \times N \times M}$, which is then normalized by softmax over the key dimension.

While this design injects edge information directly into the core query–key interaction, it is computationally and memory intensive. In particular, Eq. (16) requires constructing two edge-dependent tensors, $E^q$ and $E^k$, each of size $C \times d_s \times N \times M$. Moreover, Eq. (17) performs the inner product over pairwise query–key representations of the same size. Compared with vanilla attention, which computes $Q^\top K$ using efficient batched matrix multiplication without explicitly materializing a $C \times d_s \times N \times M$ pairwise feature tensor, GOAL introduces substantially larger intermediate storage and memory movement. This overhead can become significant for large graphs.

## B Detail Model Architecture

This section provides a step-by-step description of the GECO computation from inputs to outputs. GECO follows the general input-adapter–backbone–output-adapter architecture of GOAL (Drakulic et al., 2025), while replacing or augmenting several key components. Specifically, GECO adds an instance-preserving feature normalizer before the input adapter, replaces the GOAL mixed-attention module with ScoreMixer-in-Attention (SMA), uses a warm-up attention layer to initialize edge-aware node embeddings, and replaces the ReZero residual block in GOAL with a ReZero+RMSNorm residual block.

**Training and model hyperparameters.** GECO is trained separately on each of the five tasks using supervised learning (Drakulic et al., 2023; Luo et al., 2023; Drakulic et al., 2025). We adopt the AdamW optimizer with an initial learning rate of 0.0005, which is decayed by a factor of 0.97 every 10 epochs. The weight decay is set to $1 \times 10^{-6}$. The main supervised GECO models are trained on a single NVIDIA RTX 3090 GPU with a batch size of 256 for a total of 400 training epochs, while the training-cost comparison in Appendix G uses batch size 128 for all methods. The final model is selected based on performance on fixed validation sets. The network consists of 9 layers in total, where the first layer serves as the warm-up attention layer. Node embedding dimension $d$ is set to 128. The number of attention heads $C$ is set to 8, with query, key, and value vectors have dimension $d_s = d_v = 16$. The score mixer MLP in SMA has a hidden expansion ratio $r = 0.5$.

**Step-wise constructive interface.** GECO is used as a constructive neural solver. Given a problem instance $\mathcal{I}$ and the current construction state $s_t$, the model scores feasible actions in $\mathcal{A}(s_t)$ and selects the next action by normalizing the scores over the feasible action set. After an action is selected, the state is updated by the problem-specific transition function $\tau(\cdot)$, and the model is invoked again on the updated state. Therefore, GECO is not applied only once to the initial instance; instead, it is repeatedly applied to the state-induced subproblem at each construction step.

At step $t$, the problem interface provides node features, edge features, and a feasible-action mask:

$$X_t \in \mathbb{R}^{d_x \times N}, \qquad E_t \in \mathbb{R}^{d_e^{\text{in}} \times N \times M}, \qquad \mathcal{A}(s_t). \tag{18}$$

Here, $X_t$ denotes the node-side features of the current subproblem, $E_t$ denotes pairwise edge features, and $\mathcal{A}(s_t)$ denotes the feasible action set under the current state, and $d_x$ and $d_e^{\text{in}}$ are the raw node and edge feature dimensions. The dimensions $N$ and $M$ denote the query-side and key-side object numbers used by attention. For standard routing problems, usually $N = M$; for bipartite or multi-type problems, the same notation also covers cross-attention between different object sets.

**State-wise feature normalizer.** Before neural processing, GECO applies the instance-preserving feature normalizer introduced in Sec. 3.3 to the current state-induced features:

$$\widetilde{X}_t, \widetilde{E}_t = \text{Normalizer}(X_t, E_t). \tag{19}$$

The normalizer follows the feature-grouping and unified-scaling procedure in Eq. 9, where each feature tensor $x_i$ of Sec. 3.3 is one node-feature row of $X_t$ or one edge-feature channel of $E_t$. Features that appear additively in the same objective term or constraint expression share the same positive scaling factor. In constructive decoding, this normalization is applied in a step-wise manner. After each selected action changes the state $s_t$, the current subproblem features are reconstructed and normalized again before being sent to the neural architecture.

**Input adapter.** The normalized node and edge features are mapped into the hidden dimensions expected by the backbone:

$$H_t^0 = P_x(\widetilde{X}_t) \in \mathbb{R}^{d \times N}, \qquad \bar{E}_t = P_e(\widetilde{E}_t) \in \mathbb{R}^{d_e \times N \times M}, \tag{20}$$

where $P_x$ and $P_e$ denote the node and edge input adapters. The node adapter produces the initial hidden node embeddings $H_t^0$, while the edge adapter produces the edge representation $\bar{E}_t$ used by SMA, which is the edge feature tensor $E \in \mathbb{R}^{d_e \times N \times M}$ of Sec. 3.1. When raw node features are absent or identical, the input adapter alone may not produce distinguishable node embeddings. GECO addresses this issue through the warm-up layer, which injects edge information into node embeddings in the first backbone layer.

**GECO backbone.** The GECO backbone contains $L$ layers ($L = 9$ in all experiments). It follows the GOAL-style repeated attention and feed-forward structure, but replaces the GOAL mixed-attention module with SMA. The first layer is a warm-up attention layer:

$$U_t^1 = \mathrm{SMAA}_{\mathrm{wu}}(H_t^0, H_t^0, \bar{E}_t). \tag{21}$$

The remaining layers use standard SMA:

$$U_t^\ell = \mathrm{SMAA}(H_t^{\ell-1}, H_t^{\ell-1}, \bar{E}_t), \qquad \ell = 2, \ldots, L. \tag{22}$$

Here, $\mathrm{SMAA}(\cdot)$ denotes the SMA attention output defined in Eq. 6. Inside each SMA module, the initial score tensor is computed by Eq. 2, refined by the channel mixer in Eq. 3, calibrated by the score normalizer in Eq. 4, and normalized into attention weights by Eq. 5. Compared with the GOAL mixed-attention module in Eq. 16–Eq. 17, SMA avoids materializing high-dimensional pairwise query and key augmentation tensors and instead performs edge-aware interaction in compact score space.

The warm-up attention layer $\mathrm{SMAA}_{\mathrm{wu}}(\cdot)$ uses the same score calculation as SMA, but modifies the value branch according to Eq. 7. This allows edge information to flow into node embeddings through value aggregation, rather than only affecting attention logits inside the softmax. This edge-to-value flow is only activated in the first layer, which is re-applied at every construction step, so GECO obtains structure-aware initial node embeddings without introducing the same additional value-branch computation in every layer.

**Factorized warm-up value aggregation.** Conceptually, the warm-up layer aggregates an edge-augmented value tensor obtained by broadcasting the standard values $V = W_v H_k \in \mathbb{R}^{C \times d_v \times M}$ along the query dimension and the scalar edge values $B^v = W_{v,e} E \in \mathbb{R}^{C \times N \times M}$ along the value dimension:

$$\widehat{V}^E = \mathrm{Broad}_N(V) + \mathrm{Broad}_{d_v}(B^v) \in \mathbb{R}^{C \times d_v \times N \times M}. \tag{23}$$

Aggregating $\widehat{V}^E$ with the attention weights gives, for head $c$, value channel $k$, and query $i$, $\sum_j A_{c,i,j}\big(V_{c,k,j} + B^v_{c,i,j}\big) = (VA^\top)_{c,k,i} + \sum_j A_{c,i,j} B^v_{c,i,j}$, which is exactly the factorized form

$$O^{\mathrm{wu}} = VA^\top + \mathrm{Broad}_{d_v}\left(\mathrm{Sum}_M(A \odot B^v)\right) \in \mathbb{R}^{C \times d_v \times N} \tag{24}$$

used in Eq. (7). The factorized form only involves tensors of size $C \times N \times M$, so the $C \times d_v \times N \times M$ tensor $\widehat{V}^E$ is never materialized.

**ReZero+RMSNorm residual blocks.** Each GECO layer applies a ReZero+RMSNorm residual update after the attention branch and after the feed-forward branch. For a hidden representation $H$ and a branch output $U = F(H)$, we write this residual update as

$$\mathrm{RZN}(H, U) = H + \alpha\, \mathrm{RMSNorm}(U), \tag{25}$$

where $\alpha$ is a learnable ReZero coefficient (Bachlechner et al., 2021) and $\mathrm{RMSNorm}$ denotes root-mean-square normalization (Zhang & Sennrich, 2019). This form preserves the identity mapping when $\alpha = 0$, while using RMSNorm to stabilize the branch output. Therefore, the complete update of the first layer is

$$\widehat{H}_t^1 = \mathrm{RZN}^1_{\mathrm{attn}}(H_t^0, U_t^1), \tag{26}$$

$$H_t^1 = \mathrm{RZN}^1_{\mathrm{ffn}}\left(\widehat{H}_t^1, \mathrm{FFN}_1(\widehat{H}_t^1)\right). \tag{27}$$

For the remaining layers, the update is

$$\widehat{H}_t^\ell = \mathrm{RZN}^\ell_{\mathrm{attn}}(H_t^{\ell-1}, U_t^\ell), \qquad \ell = 2, \ldots, L, \tag{28}$$

$$H_t^\ell = \mathrm{RZN}^\ell_{\mathrm{ffn}}\left(\widehat{H}_t^\ell, \mathrm{FFN}_\ell(\widehat{H}_t^\ell)\right), \qquad \ell = 2, \ldots, L. \tag{29}$$

Thus, GECO preserves the high-level GOAL backbone layout, while replacing the attention computation and residual normalization design.

**Output adapter and feasible-action masking.** After the final backbone layer, the output adapter produces the scoring function used by the constructive policy:

$$f_\theta(s_t, \mathcal{I}, a) = \mathrm{OutputAdapter}(H_t^L, \bar{E}_t, s_t, a), \qquad a \in \mathcal{A}(s_t). \tag{30}$$

Infeasible actions are masked before normalization. The action distribution is then computed over the feasible action set:

$$p(a_t \mid s_t, \mathcal{I}) = \mathrm{softmax}_{a \in \mathcal{A}(s_t)} \big(f_\theta(s_t, \mathcal{I}, a)\big). \tag{31}$$

During inference, the next action is selected greedily from this distribution in our main evaluation. During supervised training, the model is optimized to assign high probability to the reference action at each construction step.

**Autoregressive decoding loop.** Once the next action $a_t$ is selected, the construction state is updated as

$$s_{t+1} = \tau(s_t, a_t; \mathcal{I}). \tag{32}$$

The updated state $s_{t+1}$ induces the next subproblem, and GECO repeats the same pipeline: state-wise normalization, input adaptation, SMA-based backbone computation, output adaptation, feasible-action masking, and action selection. In this sense, GECO can be viewed as a GOAL-style constructive architecture whose main changes are localized to the feature normalization stage, the edge-aware attention module, the first-layer warm-up value branch, and the residual normalization block.

# C Complexity and Expressivity Analysis of SMA

Per query–key pair, SMA costs $O(C(d_s + d_e + rC))$ time and stores only $(1 + r)C$ values, while GOAL and MatNet store $O(Cd_s)$ and $O(Ch_s)$ values, where $h_s$ is the width of the MatNet score MLP (Sec. C.1 and Table 8). With our default $C = 8$ and $r = 0.5$, its mixer uses $rC = 4$ hidden activations per pair, compared with $Cd_s = 128$ pairwise hidden values in GOAL and $Ch_s = 128$ in MatNet ($d_s = h_s = 16$). These counts refer to the pairwise hidden values of the score computation and not to the total memory of the model. SMA keeps the $O(NM)$ pairwise dependence of dense edge-aware attention and reduces the cost of each pair. The compact design also affects which score functions SMA can represent, as analyzed in Sec. C.2. Since edge information enters additively, two inputs with the same fused score vector receive the same refined scores (Prop. C.1). At the same time, each refined score can depend nonlinearly on all heads, which no head-separable MatNet scorer can represent (Prop. C.3). GOAL, in turn, can represent multiplicative node–edge terms that neither SMA nor MatNet can (Prop. C.4), so the three designs cannot be ranked by expressivity alone. With matching ReLU activations and a mixer of width at least $Ch_s$, a concatenation variant of SMA strictly contains MatNet (Prop. C.5). Score mixing itself therefore does not limit expressivity relative to MatNet, and the gap of the default design comes from its additive fusion and its narrow mixer. We evaluate the default design empirically through controlled module replacements (Sec. 4.2) and resource measurements (Sec. 4.3).

## C.1 Time and Memory Complexity

We count the dominant cost of computing the pre-softmax scores for a single query–key pair, in one dense edge-aware attention layer at one construction step. The totals in Table 8 multiply these per-pair costs by $NM$. We omit the operations shared by all three designs, namely the node projections $W_q H_q$ and $W_k H_k$ (cost $O((N + M)Cd_s d)$), value aggregation, feed-forward layers, and storage of the raw edge input. The complexities describe the direct dense implementations given by Eqs. (15)–(17) and Eqs. (2)–(4). They are not lower bounds that hold for every implementation.

**SMA.** The node compatibility $Q^\top K$ costs $Cd_s$ multiply–adds per pair, the edge projection $W_e e_{ij}$ costs $Cd_e$, and the two mixer layers $W_1 \in \mathbb{R}^{rC \times C}$ and $W_2 \in \mathbb{R}^{C \times rC}$ cost $2rC^2$. ScoreNorm and softmax are $O(C)$ per pair (the query-axis maximum is one pass over the score tensor). The per-pair time is therefore $O(C(d_s + d_e + rC))$. The only pairwise activations are the $C$ fused scores and the $rC$ mixer hidden units, which gives $O((1 + r)C)$ memory per pair.

**MatNet.** Besides $Q^\top K$ ($Cd_s$), each of the $C$ head-specific MLPs maps a $(1 + d_e)$-dimensional input to $h_s$ hidden units and back to one score, costing $h_s(d_e + 2)$. The per-pair time is $O(C(d_s + h_s(d_e + 1)))$, and storing the $C$ hidden activations of width $h_s$ requires $O(Ch_s)$ memory per pair.

Table 8: Dominant pairwise cost of one dense edge-aware attention layer at one construction step. Operations shared by all designs (node projections, value aggregation, feed-forward layers, and raw edge inputs) are omitted. $C$ is the number of heads, $d_s$ the per-head query/key width, $d_e$ the edge feature width, $h_s$ the MatNet score-MLP width, and $rC$ the SMA mixer width.

| Attention | Pairwise time | Pairwise activation memory |
|---|---|---|
| GOAL (Drakulic et al., 2025) | $O\big(NMC\,d_s(d_e+1)\big)$ | $O(NMC\,d_s)$ |
| MatNet (Kwon et al., 2021) | $O\big(NMC\,(d_s+h_s(d_e+1))\big)$ | $O(NMC\,h_s)$ |
| SMA (ours) | $O\big(NMC\,(d_s+d_e+rC)\big)$ | $O\big(NMC\,(1+r)\big)$ |

**GOAL.** Constructing $E^q_{:,i,j}$ and $E^k_{:,i,j}$ costs $2Cd_sd_e$, and the edge-augmented inner product costs $O(Cd_s)$, giving $O(Cd_s(d_e+1))$ per pair. The two augmentation tensors store $2Cd_s$ values per pair, which is $O(Cd_s)$.

**Trajectory cost.** All three designs have the same dependence on the number of pairs. A constructive solver re-applies the policy to the state-induced subproblem, so the total cost over a trajectory is $\sum_t N_tM_t$ times the per-pair cost. When the policy re-encodes the remaining $n-t$ candidates at each of $n$ steps, this sum is $\Theta(n^3)$ for every dense edge-aware design. SMA lowers the per-pair constant and leaves this dependence unchanged.

### C.2 Expressivity of the Score Map

We compare the score functions that the three designs can represent for a single query–key pair before ScoreNorm. The query-axis maximum in ScoreNorm couples different pairs, so we leave it out of this local analysis. Let $q,k\in\mathbb{R}^d$ be the query and key embeddings of the pair and $e\in\mathbb{R}^{d_e}$ its edge feature. Each entry of the node-compatibility vector $a(q,k)\in\mathbb{R}^C$ is a bilinear form $a_c(q,k)=q^\top M_ck$ with $M_c=W_q^{(c)\top}W_k^{(c)}/\sqrt{d_s}$. We consider the following function classes, each ranging over all parameter values for fixed $C$ and widths:

$$\mathcal{S}_{\mathrm{SMA}}: f(q,k,e)=u+W_2\,\sigma(W_1u),\quad u=a(q,k)+W_ee, \tag{33}$$

$$\mathcal{S}_{\mathrm{Mat}}: [f(q,k,e)]_c=w_2^{(c)}\,\phi\big(W_1^{(c)}[a_c(q,k);\,e]\big),\quad c=1,\dots,C, \tag{34}$$

$$\mathcal{S}_{\mathrm{GOAL}}: [f(q,k,e)]_c=\big\langle W_q^{(c)}q+W_{e,q}^{(c)}e,\ W_k^{(c)}k+W_{e,k}^{(c)}e\big\rangle/\sqrt{d_s}, \tag{35}$$

$$\mathcal{S}_{\mathrm{Concat}}: f(q,k,e)=Pz+W_2\,\sigma(W_1z),\quad z=[a(q,k);\,e]\in\mathbb{R}^{C+d_e}, \tag{36}$$

where $\sigma$ is the activation of the SMA mixer (SiLU in GECO) and $\phi$ is ReLU as in MatNet. $\mathcal{S}_{\mathrm{Concat}}$ is a generalized variant of SMA that feeds the concatenation of compatibility scores and edge features to the head mixer, with a linear residual $P\in\mathbb{R}^{C\times(C+d_e)}$.

**Proposition C.1** (Additive fusion)**.** *Every $f\in\mathcal{S}_{\mathrm{SMA}}$ depends on $(q,k,e)$ only through the fused vector $u=a(q,k)+W_ee$. In particular, $f(q,k,e)=f(q',k',e')$ whenever $a(q,k)+W_ee=a(q',k')+W_ee'$. For example, $f(q,k,e)=f(q,k,e+\delta)$ for every $\delta\in\ker W_e$.*

*Proof.* Immediate from the definition, since $u$ is the only argument of the mixer. □

Proposition C.1 shows which information the additive fusion of SMA discards. These are edge variations that do not change the $C$ fused scores, and changes of the compatibility and edge terms that leave their sum unchanged.

The following lemma is used in the next two propositions.

**Lemma C.2.** *Let $\sigma\in\{\mathrm{ReLU},\mathrm{SiLU}\}$ and $\alpha,\beta\in\mathbb{R}$. If $t\alpha+\sigma(t\beta)=0$ for all $t\in\mathbb{R}$, then $\alpha=\beta=0$.*

*Proof.* For SiLU, $\sigma(0)=0$, $\sigma'(0)=1/2$, and $\sigma''(0)=1/2$, so the second derivative of $t\mapsto t\alpha+\sigma(t\beta)$ at $t=0$ is $\beta^2/2$, which forces $\beta=0$. Then $t\alpha\equiv 0$ forces $\alpha=0$. For ReLU, if $\beta>0$, evaluating the identity at $t=1$ and $t=-1$ gives $\alpha+\beta=0$ and $-\alpha=0$, which is a contradiction. The case $\beta<0$ is symmetric. Hence $\beta=0$, and then $\alpha=0$. □

**Proposition C.3** (Cross-head dependence). *Let $C \geq 2$, $d \geq 2$, $d_s \geq 1$, and $rC \geq 1$. There exists $f \in \mathcal{S}_{\mathrm{SMA}}$ whose first output is not of the form $h(b(q,k),e)$ for any bilinear form $b$ and any function $h$. Consequently, $f \notin \mathcal{S}_{\mathrm{Mat}}$. The same holds with $\mathcal{S}_{\mathrm{Concat}}$ in place of $\mathcal{S}_{\mathrm{SMA}}$.*

*Proof.* Choose head 1 and head 2 to read the first and second coordinates, so that $a_1(q,k) = q_1 k_1$ and $a_2(q,k) = q_2 k_2$ (e.g., $W_q^{(c)} = W_k^{(c)} = d_s^{1/4}\,\epsilon_1' \epsilon_c^\top$ for $c = 1, 2$, where $\epsilon_1' \in \mathbb{R}^{d_s}$ and $\epsilon_c \in \mathbb{R}^d$ are standard basis vectors, and all other heads zero), set $W_e = 0$, let the first mixer unit read $u_2$, and route it to output 1 with weight 1. Then $[f]_1 = q_1 k_1 + \sigma(q_2 k_2)$. Suppose $[f]_1 = h(b(q,k),e)$ with $b(q,k) = q^\top M k$. Restricting to the first two coordinates, let $M \in \mathbb{R}^{2\times 2}$ denote the corresponding block. Since $b(q,0) = 0$ and $[f]_1(q,0) = \sigma(0) = 0$, $[f]_1$ must vanish on $Z = \{(q,k) : b(q,k) = 0\}$. If $M = 0$, then $Z$ contains $(\epsilon_1, \epsilon_1)$ where $[f]_1 = 1$, a contradiction. Otherwise, for every $q$ with $M^\top q \neq 0$, $Z$ contains the line $\{(q, tw) : t \in \mathbb{R}\}$ with $w = (-(M^\top q)_2, (M^\top q)_1)$, on which $[f]_1 = t\, q_1 w_1 + \sigma(t\, q_2 w_2)$. By Lemma C.2, vanishing for all $t$ requires $q_1 w_1 = q_2 w_2 = 0$. If this held for all $q$ outside the proper subspace $\ker M^\top$, then, by continuity, the polynomial identities $q_1(M_{12}q_1 + M_{22}q_2) \equiv 0$ and $q_2(M_{11}q_1 + M_{21}q_2) \equiv 0$ would hold, giving $M = 0$, a contradiction. Every output of a MatNet scorer has the excluded form with $b = a_c$, so $f \notin \mathcal{S}_{\mathrm{Mat}}$. For $\mathcal{S}_{\mathrm{Concat}}$, the same $[f]_1$ is obtained by letting $P$ copy $a_1$ to output 1 and the first mixer unit read $a_2$. □

**Proposition C.4** (Multiplicative node–edge terms). *Let $d, d_s, d_e \geq 1$. There exists $f \in \mathcal{S}_{\mathrm{GOAL}}$ with $[f(q,k,e)]_1 = q_1 e_1$, and no element of $\mathcal{S}_{\mathrm{SMA}} \cup \mathcal{S}_{\mathrm{Mat}} \cup \mathcal{S}_{\mathrm{Concat}}$ represents this function. Conversely, the SMA scorer constructed in Proposition C.3 does not belong to $\mathcal{S}_{\mathrm{GOAL}}$.*

*Proof.* In head 1 of GOAL, let $W_q^{(1)}$ map $q_1$ to the first coordinate with weight $d_s^{1/4}$, let $W_{e,k}^{(1)}$ map $e_1$ to the first coordinate with weight $d_s^{1/4}$, and set $W_k^{(1)} = 0$ and $W_{e,q}^{(1)} = 0$. Then $[f]_1 = q_1 e_1$. For the other three classes, $a(q,0) = 0$ for every bilinear compatibility, so their outputs at $k = 0$ depend on $e$ only, whereas $q_1 e_1$ at $k = 0$ varies with $q$ whenever $e_1 \neq 0$. For the converse, every output of a GOAL scorer is a polynomial of degree at most two in $(q,k,e)$, whereas $q_1 k_1 + \sigma(q_2 k_2)$ restricted to $q = (0, 1, 0, \dots)$ equals $\sigma(k_2)$, which is not a polynomial for $\sigma \in \{\mathrm{ReLU}, \mathrm{SiLU}\}$. □

**Proposition C.5** (Concatenation variant contains MatNet). *Let $\sigma = \phi = \mathrm{ReLU}$ in $\mathcal{S}_{\mathrm{Concat}}$ and let its mixer width be at least $Ch_s$. Then $\mathcal{S}_{\mathrm{Mat}} \subsetneq \mathcal{S}_{\mathrm{Concat}}$.*

*Proof.* Given a MatNet scorer, use the same $W_q, W_k$ in $\mathcal{S}_{\mathrm{Concat}}$, set $P = 0$, and allocate a block of $h_s$ hidden units to each head $c$. The rows of $W_1$ in block $c$ apply $W_1^{(c)}$ to the coordinates $(a_c, e)$ of $z$ and are zero on the other compatibility coordinates, and row $c$ of $W_2$ equals $w_2^{(c)}$ on block $c$ and zero elsewhere. Then output $c$ equals $w_2^{(c)} \phi(W_1^{(c)}[a_c; e])$, which proves the inclusion. Strictness follows from Proposition C.3. □

In summary, SMA represents cross-head interactions that no MatNet scorer can, and SMA and GOAL are incomparable, with GOAL representing multiplicative node–edge terms that neither SMA nor MatNet represents. With ReLU activations and a mixer of width at least $Ch_s$, replacing the fused input $u$ by the concatenation $z$ recovers MatNet as a head-separable, block-sparse special case, so score mixing itself does not limit expressivity relative to MatNet. The default design differs from this construction in its additive fusion, its SiLU activation, and its narrow mixer. It uses only $rC = 4$ hidden units per query–key pair, each of which reads all $C$ fused head scores and can contribute to all $C$ outputs, while this construction needs $Ch_s = 128$ head-specific units. SMA thus gives up some expressivity to obtain a compact pairwise representation that still captures cross-head interactions. Because expressivity alone does not determine which design performs best, we compare the designs empirically in Sec. 4.2.

# D IMPACT OF ATTENTION SCORE CALIBRATORS

## D.1 IMPACT OF ATTENTION SCORE CALIBRATORS

### D.1.1 SCORE-STATISTIC CALIBRATION

We provide a detailed analysis of score-statistic calibration in ScoreNorm. Recall that ScoreNorm follows the general form

$$\mathrm{ScoreNorm}(\tilde{S}) = \alpha(\tilde{S}) \cdot \left(\tilde{S} - \beta(\tilde{S})\right), \tag{37}$$

where $\alpha(\cdot)$ is a multiplicative factor and $\beta(\cdot)$ is an additive correction. Since the output of ScoreNorm is directly consumed by the row-wise softmax, these operations should be interpreted as attention-logit calibration rather than conventional feature normalization. We therefore evaluate three design axes: whether a score statistic is used as $\alpha(\cdot)$ or $\beta(\cdot)$, which statistic is used, and along which axis the statistic is computed.

Table 9 compares representative score-statistic calibration designs on ATSP validation. These experiments are conducted under a lighter training setting due to computational constraints, so the absolute numbers are not directly comparable with the main results. Instead, the purpose is diagnostic: we use this comparison to understand which types of score-statistic calibration are compatible with size extrapolation.

**Good training-size performance does not necessarily imply good size generalization.** For example, using query-axis RMS as a multiplicative factor achieves the best ATSP100 validation result, reducing the gap from $1.43\%$ to $0.81\%$. However, the same variant degrades substantially on ATSP500, increasing the gap to $4.66\%$. This suggests that score calibrators selected only from the training size can overfit to the logit distribution at that size and fail when the number of candidate actions changes.

**Score-statistic-based multiplicative factors can stabilize training-size validation but are unreliable under size extrapolation.** This does not mean that multiplicative calibration is inherently harmful, as the support-size factor used in ScoreNorm is also multiplicative. The key distinction is that support-size scaling compensates for a known external factor, namely the number of valid keys in the softmax support, whereas score-statistic rescaling normalizes the realized score variation itself. Such rescaling can improve optimization at the training size by keeping logits in a well-conditioned numerical range and preventing overly sharp or overly flat attention distributions. However, the same operation can be harmful for size extrapolation. For a query with $N$ candidate keys, maintaining a similar selection confidence generally requires larger score gaps as $N$ increases, since the best action must compete with more alternatives. In a simplified case where the best key has gap $\Delta$ over the other keys, its softmax probability is approximately

$$p_{\max} \approx \frac{1}{1 + (N-1)\exp(-\Delta)}. \tag{38}$$

Thus, preserving the same confidence requires $\Delta$ to grow on the order of $\log N$ (Nakanishi, 2025), which is exactly the support-size scalar we used. If logits are rescaled by score statistics such as RMS or variance, this necessary growth in score contrast can be suppressed. More broadly, attention-logit variation is not merely a nuisance variable. It can encode useful decision information, including row-wise confidence, candidate distinctiveness, and head specialization. Normalizing this variation may therefore improve training-size performance while removing information that becomes important when the number of candidate actions changes.

**Query-axis statistics are the most influential calibrator.** Computing statistics along the query axis produces one correction value for each key, thereby calibrating how attractive the same key is across different queries. This is complementary to the row-wise softmax: the softmax normalizes competition among keys for each query, but it does not explicitly calibrate the global attractiveness of each key across queries. Query-axis calibration therefore introduces a missing column-wise interaction in attention-score space. The choice of statistic is also important. Query-axis mean improves ATSP100 but degrades on ATSP500, while query-axis maximum achieves the best ATSP500 result among all compared variants. Intuitively, the maximum statistic targets keys that become highly attractive to at

least one query and can act as a conservative correction against overly dominant columns. In contrast, mean statistics are more sensitive to the changing distribution of all query scores, and minimum statistics are dominated by low-compatibility query-key pairs.

Overall, Table 9 suggests a conservative design principle: avoid score-statistic-based rescaling, but use a lightweight shift-only correction along the query axis. We therefore instantiate the statistic-based part of ScoreNorm as a learnable partial subtraction of the query-axis maximum,

$$\tilde{S} - \alpha_1 \odot \mathrm{Max}_q(\tilde{S}), \tag{39}$$

where $\alpha_1$ controls the correction strength for each head. Unlike statistic-based rescaling, this correction never divides the logits by an estimated score magnitude. It only adjusts key-specific offsets and adds a column-wise calibration term that empirically improves size generalization.

### D.2 Support-Size Calibration

**Attention-dilution diagnostic.** Fig. 2 presents the relative attention concentration under size extrapolation. We measure concentration by the average row-wise maximum attention probability after feasibility masking, normalized by its value at the training size $N = 100$. A smaller value indicates that attention becomes more diffuse over the valid candidate keys. The results show a clear attention-dilution phenomenon: when models trained at $N = 100$ are evaluated on larger instances, the maximum attention probability drops substantially as the feasible support grows. GOAL suffers the strongest dilution, retaining only about $41\%$ of its original attention concentration at $N = 1000$. GECO without support-size scaling also exhibits dilution, but retains about $52\%$, suggesting that the proposed score-space attention design is already less sensitive to support-size growth. This is consistent with its better CVRPTW1000 performance compared with GOAL even before applying the $\log N_k$ factor. The full GECO model further preserves attention concentration, indicating that support-size calibration provides an additional correction on top of the proposed architecture. The diagnostic alone does not show that dilution causes the degradation, so the ablation below evaluates the factor directly through solution quality.

**Ablation of support-size scaling.** Table 4 isolates the effect of the $\log N_k$ factor. Keeping a comparable weight on the best candidate requires logit gaps that grow on the order of $\log N_k$ (Appendix D.1.1), which motivates this factor. The variant "GECO w/o $\log N_k$" removes only support-size scaling while keeping the remaining ScoreNorm components unchanged. On ATSP, support-size calibration consistently improves size generalization, reducing the ATSP1000 gap from $1.12\%$ to $0.87\%$. The effect is more pronounced on CVRPTW, where removing the factor increases the CVRPTW1000 gap from $2.26\%$ to $6.58\%$. These results validate that support-size scaling provides a useful multiplicative calibration signal, especially when size extrapolation substantially enlarges the feasible candidate set.

**Inference-time refocusing diagnostics.** We conduct additional inference-time calibration experiments to further examine the attention-dilution diagnosis. The motivation is that logit rescaling is not the only way to increase concentration over promising candidates. If severe size extrapolation hurts performance partly because attention becomes overly diffuse over an enlarged feasible set, then inference-time strategies that reduce or refocus the effective candidate set may also improve large-size performance. We therefore evaluate three simple modifications on GOAL: 100-NN token pruning, which keeps only the nearest 100 candidates as model input; Top-100 Score pruning, which keeps the 100 highest-scoring candidates for each query in attention; and Log-scaling, which rescales logits according to the valid support size.

Table 10 supports this diagnosis on CVRPTW. Compared with the original GOAL model, these refocusing or rescaling strategies substantially improve large-size performance, reducing the CVRPTW1000 gap from $26.34\%$ to much smaller values. This suggests that GOAL's severe degradation on large CVRPTW instances is closely related to insufficient concentration over the growing candidate set. However, the ATSP results show that this should not be interpreted as a general preference for hard pruning. In particular, 100-NN token pruning severely degrades ATSP performance, increasing the ATSP1000 gap from $1.67\%$ to $23.79\%$. Since this strategy removes non-nearest candidates from the model input, it can discard globally important information before the model has a chance to compare them. Thus, the goal is not simply to shrink the candidate set, but to maintain

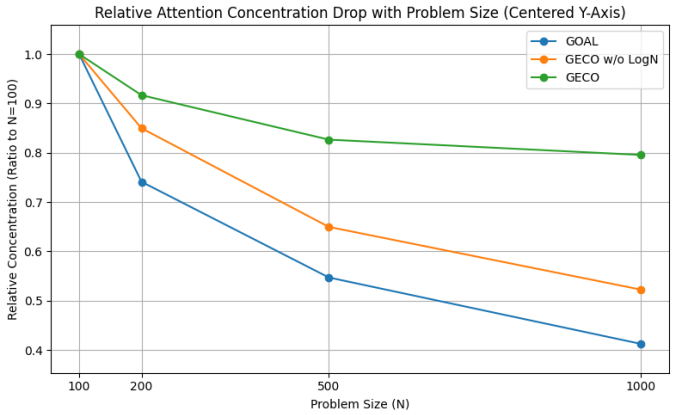


Figure 2: Relative attention concentration under size extrapolation. We report the average row-wise maximum attention probability normalized by its value at $N = 100$. GECO with support-size scaling preserves attention concentration better as problem size increases. Since each curve is normalized by its own value at $N = 100$, the plot shows how much concentration each model retains and does not compare absolute concentration across models.

Table 9: Validation comparison of score-calibration designs on ATSP. Q, K, and H identify the dimension over which the statistic is computed, so reducing over Q produces one correction per key. Columns under $\alpha$ use the statistic as a multiplicative factor, and columns under $\beta$ use it as an additive shift.

| | | $\alpha$ | | | | | $\beta$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | None | QK, RMS | K, RMS | Q, RMS | H, RMS | K, Max | H, Mean | H, Max | Q, Mean | Q, Min | Q, Max |
| **ATSP100** | 1.43% | 1.71% | 1.24% | **0.81%** | 1.34% | 1.58% | 1.46% | 2.05% | 1.26% | 1.40% | 1.29% |
| **ATSP500** | 1.56% | 1.97% | 2.32% | 4.66% | 2.36% | 3.06% | 1.55% | 2.25% | 2.28% | 1.62% | **1.38%** |

calibrated competition over the feasible support. Support-size scaling provides a softer correction: it adjusts the logit scale according to the valid support size while keeping all feasible candidates available to the softmax.

## E Effectiveness of Warm-up Layer

We compare two alternative node embedding initialization methods in Table 5. The gain is particularly clear on OP where node quality depends not only on its reward (node feature) but also on its accessibility under the travel budget (edge feature). Thus, effective initialization should encode how each node is situated in the edge-cost structure. KNN packing provides only a fixed local distance summary, and random identifiers provide distinguishability without relational information. By contrast, the warm-up layer learns to aggregate edge information into node embeddings through attention, producing structure-aware node representations that transfer better when the graph size changes.

## F On the Design Choices of Feature Normalizer

We evaluate the design choices of the proposed feature normalizer along three axes in Table 11, which also includes a no-normalization baseline on OP.

**Role of** $\mathrm{Stat}$ **and** $\mathrm{Agg}$**.** Instance preservation depends only on the resulting factor $s_p$ being finite, positive, and shared by all features in group $\mathcal{G}_p$, and it does not depend on which $\mathrm{Stat}$ and $\mathrm{Agg}$ are used. These operators determine which notion of group magnitude is normalized. When both operators are positively homogeneous of degree one, as RMS, maximum, and mean are, rescaling all features of a group by a common $\lambda > 0$ rescales $s_p$ by $\lambda$. The normalized features are then invariant

Table 10: Effects of inference time strategy.

| | ATSP500 Gap | ATSP1000 Gap | CVRPTW500 Gap | CVRPTW1000 Gap |
|---|---|---|---|---|
| **GOAL** | 1.25% | 1.67% | 18.48% | 26.34% |
| **- 100-NN** | 18.88% | 23.79% | 9.27% | 7.49% |
| **- Top-100 Score** | 1.12% | 1.18% | 15.28% | 21.53% |
| **- Log-scaling** | 1.19% | 1.70% | 10.07% | 10.37% |

Table 11: Effects of different norm choices. The three-tuple encodes norm frequency, choices of Stat and Agg operators respectively.

| | CVRP100 | CVRP200 | CVRP500 | CVRP1000 |
|---|---|---|---|---|
| (One-shot, Max, Max) | **1.91%** | **2.37%** | 3.98% | 5.37% |
| (One-shot, Max, Mean) | 2.97% | 3.98% | 6.10% | 6.14% |
| (One-shot, RMS, Mean) | 3.23% | 3.05% | 4.25% | 5.09% |
| (Step-wise, Max, Max) | 2.28% | 2.33% | 3.55% | 4.87% |
| (Step-wise, Max, Mean) | 2.39% | 2.55% | 4.38% | 5.02% |
| (Step-wise, RMS, Mean) | 2.37% | **2.37%** | **3.36%** | **4.79%** |
| | OP100 | OP200 | OP500 | OP1000 |
| None | -0.19% | 1.43% | 3.90% | 10.37% |
| (One-shot, Max, Max) | 0.41% | 7.07% | 13.40% | 15.57% |
| (One-shot, Max, Mean) | -0.31% | 1.08% | 6.18% | 10.29% |
| (One-shot, RMS, Mean) | -0.33% | 1.27% | 8.46% | 15.61% |
| (Step-wise, Max, Max) | -0.31% | 0.34% | 2.06% | 4.99% |
| (Step-wise, Max, Mean) | **-0.34%** | **-0.18%** | **1.66%** | **4.23%** |
| (Step-wise, RMS, Mean) | **-0.34%** | 0.27% | 2.98% | 5.11% |

to the units in which the group is expressed, and they satisfy

$$\mathrm{Agg}\left(\{\mathrm{Stat}(\widetilde{x}_i) : x_i \in \mathcal{G}_p\}\right) = 1. \tag{40}$$

As a result, RMS–Mean normalizes the average RMS magnitude within a group to one, Max–Mean normalizes the average per-feature maximum, and Max–Max normalizes the largest maximum. Max-based choices give stronger range control but can be dominated by a single extreme value, while RMS and mean spread the scale estimate over entries and feature tensors. We do not know of an architecture-independent reason for one pair to be optimal in general, and in Table 11 no combination dominates across CVRP and OP. We therefore use RMS–Mean as a stable default. In our experiments, adapting the scale to the current construction state (step-wise normalization) matters more than the exact operator pair.

## G Efficiency Analysis

We provide detailed efficiency measurements for GECO, GECO-MatNet, and GOAL. All measurements are conducted on a single RTX 3090 GPU under the same batch size. GECO-MatNet replaces SMA with MatNet-style score mixing while keeping the remaining GECO components unchanged, so the comparison between GECO and GECO-MatNet isolates the efficiency effect of the proposed score-space attention design.

**Inference consumption.** Table 6 reports wall-clock inference time and peak GPU memory on ATSP. GECO consistently achieves the lowest resource consumption among the compared edge-aware architectures. Compared with GOAL, GECO is $5.4$–$6.4\times$ faster and uses $7.4$–$11.9\times$ less peak GPU memory. Compared with GECO-MatNet, GECO is $1.8$–$2.1\times$ faster and uses $2.9$–$4.2\times$ less memory. This shows that the efficiency improvement comes not only from avoiding the GOAL-style high-dimensional edge fusion, but also from the specific SMA design.

**FLOPs.** Table 12 reports single-step FLOPs, including both total model FLOPs and the FLOPs of a single attention module. GECO substantially reduces computation compared with GOAL, and the

Table 12: Single step FLOPs breakdown (total vs. single attention).

| | Method | n100 | n200 | n500 | n1000 |
|---|---|---|---|---|---|
| Total | GOAL | $3.2 \times 10^9$ | $1.2 \times 10^{10}$ | $7.5 \times 10^{10}$ | $3.0 \times 10^{11}$ |
| | GECO-MatNet | $2.5 \times 10^8$ | $6.3 \times 10^8$ | $2.6 \times 10^9$ | $8.7 \times 10^9$ |
| | GECO | $2.1 \times 10^8$ | $4.8 \times 10^8$ | $1.6 \times 10^9$ | $4.8 \times 10^9$ |
| Attn | GOAL | $3.4 \times 10^8$ | $1.3 \times 10^9$ | $8.3 \times 10^9$ | $3.3 \times 10^{10}$ |
| | GECO-MatNet | $1.4 \times 10^7$ | $4.4 \times 10^7$ | $2.3 \times 10^8$ | $8.3 \times 10^8$ |
| | GECO | $9.9 \times 10^6$ | $2.7 \times 10^7$ | $1.2 \times 10^8$ | $4.0 \times 10^8$ |

Table 13: Training-time resource consumption. We report total training time (validation excluded) and peak GPU memory on a single RTX 3090.

| **Metric** | **GOAL** | **GECO-MatNet** | **GECO** |
|---|---|---|---|
| Time (h) | 230 | 84.8 | 74.6 |
| Peak Mem (MiB) | 17038 | 17148 | 5338 |

reduction becomes larger as the problem size increases. At $n = 1000$, GECO reduces total FLOPs from $3.0 \times 10^{11}$ to $4.8 \times 10^9$, and attention FLOPs from $3.3 \times 10^{10}$ to $4.0 \times 10^8$. Compared with GECO-MatNet, GECO also consistently reduces both total and attention FLOPs. This confirms that SMA lowers the cost of edge-conditioned attention by keeping refinement in the compact $C \times N \times M$ score tensor, instead of materializing high-dimensional pairwise feature tensors.

**Training cost.** Table 13 reports training-time resource consumption on CVRP. Since GOAL cannot be trained with the original batch size on a 24GB RTX 3090, we use batch size 128 for all compared methods. GECO requires 74.6 hours of training, compared with 84.8 hours for GECO-MatNet and 230 hours for GOAL. It also reduces peak memory from about 17GB for GOAL and GECO-MatNet to 5.3GB. Therefore, the efficiency gain of GECO improves not only inference rollout, but also practical trainability.

## H Explicit Edges versus Coordinates

Table 14 reports the comparison discussed in Sec. 4.2. The coordinate-based variant removes the edge-dependent warm-up layer, replaces SMA with standard node-only attention, and receives node coordinates as node features, while the remaining architecture and training configuration are unchanged. Both variants therefore receive equivalent geometric information on Euclidean CVRP but represent and process it through different interfaces. Because this comparison changes the input interface, the attention module, and the warm-up layer together, it evaluates the edge-aware design as a whole and does not isolate a single component. The edge-based interface processes the dense $N \times N$ distance matrix (Appendix C), while coordinate-based inputs describe a Euclidean instance with $N$ vectors. Dense node-only attention also computes $N \times N$ scores, however, so the runtime difference observed against BQ-NCO in Sec. 4.2 cannot be attributed to explicit edges alone.

Table 14: Coordinate-based versus edge-based input interfaces of GECO on Euclidean CVRP (gap).

| | CVRP100 | CVRP200 | CVRP500 | CVRP1000 |
|---|---|---|---|---|
| GECO w/ coordinates | 3.22% | 3.09% | **3.38%** | 4.88% |
| GECO w/ edges | **2.37%** | **2.37%** | 3.40% | **4.79%** |

## I Beyond Supervised Training

Our main experiments use supervised learning so that GECO is compared with the closest heavy-decoder baselines (Drakulic et al., 2023; Luo et al., 2023; Drakulic et al., 2025) under the protocol

used in their own evaluations. This section shows that GECO can also be trained without expert labels.

**Label-free self-improvement.** We train GECO on ATSP by self-improvement (Luo et al., 2025a; Corsini et al., 2024; Pirnay & Grimm, 2024), where the model is supervised by solutions generated by itself through inference-time search. At each epoch, we randomly generate $2048$ ATSP instances and use a frozen teacher, the checkpoint with the best greedy validation performance so far, to generate pseudo-labels. For each instance, the teacher samples $32$ width-$2$ beam-search trajectories, and the best solution is kept. The resulting instance–solution pairs are added to a first-in-first-out buffer of capacity $8192$, from which the student is trained for $400$ epochs with the same cross-entropy objective and per-epoch update budget as in supervised learning. Without any externally generated solutions, the resulting model reaches a gap of $1.27\%$ on ATSP100 and generalizes to larger instances, with a gap of $2.42\%$ on ATSP1000 (Table 15). Supervised GECO remains better by $1.0$ to $1.6$ percentage points. This is expected, since its labels are produced by LKH, and label-free training generally needs more instances or updates than supervised learning to reach the same solution quality.

Table 15: Test gap of GECO on ATSP when trained with LKH labels (supervised) and when trained for 400 epochs by label-free self-improvement.

| | ATSP100 | ATSP200 | ATSP500 | ATSP1000 |
|---|---|---|---|---|
| GECO (supervised) | 0.31% | 0.55% | 0.81% | 0.87% |
| GECO (self-improvement) | 1.27% | 1.74% | 2.10% | 2.42% |