SPARSE IS NOT ENOUGH: A SCALABLE INFERENCE FRAMEWORK FOR HEATMAP-BASED METHODS

Anonymous authors
Paper under double-blind review

ABSTRACT

Heatmap-based methods have shown strong performance on large-scale vehicle routing problems (VRPs), yet their scalability remains limited by inefficient inference pipeline. We identify two bottlenecks in existing inference pipelines: (1) quadratic intermediate representations during candidate graph construction and heatmap representation. (2) inefficient autoregressive decoding due to substantial CPU-side overhead from launching numerous fine-grained GPU operations. To address these bottlenecks, we propose **SIF**, a scalable sparse inference framework. Specifically, SIF first employs sparse search strategies tailored to different sparsification criteria to construct candidate graphs, and then represents scores directly on the resulting sparse candidate pairs, thereby avoiding quadratic intermediate representations from both candidate graph construction and heatmap representation. To improve decoding efficiency, SIF fuses the entire autoregressive decoding loop into a single GPU-resident kernel, eliminating repeated CPU-side kernel launches. Moreover, a low-overhead rescue mechanism is introduced to mitigate the solution degradation caused by index-biased greedy fallback when sparsification leaves no feasible candidates. Notably, SIF can be integrated into existing heatmap-based solvers without modifying their learned models. Experiments on four representative heatmap-based methods show that SIF consistently extends the maximum solvable problem scale while substantially improving inference efficiency. On a single 24GB GPU, SIF enables representative heatmap-based solvers to process instances with up to 100,000 nodes and achieves up to a $182.11\times$ speedup in inference over the original inference pipelines.

1 INTRODUCTION

Vehicle routing problem (VRPs) constitute a fundamental class of combinatorial optimization problems with broad applications in logistics (Gaur & Fisher, 2004), transportation (Florio et al., 2021), and delivery planning (Voccia et al., 2019). Neural combinatorial optimization (Bengio et al., 2021) provides a data-driven paradigm that employs neural networks to learn solution policies for such problems. Among these approaches, heatmap-based methods (Ye et al., 2024; Pan et al., 2025; Kim et al., 2025; Zhang & Cao, 2025) are particularly attractive because they can leverage graph structure to generate a heatmap of pairwise scores, which provides global guidance for solution construction. However, these methods incur substantial memory overhead due to quadratic intermediate representations. Moreover, their CPU-controlled decoding loops introduce significant runtime overhead, limiting the scalability from training on small-scale instances to inference on large-scale instances.

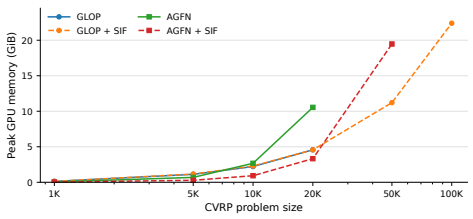


Figure 1: Scaling to 100K Nodes with SIF framework. Peak GPU memory usage of GLOP with and without SIF on CVRP. On a single 24 GB GPU, SIF increases the maximum solvable instance size from 20K to 100K.

To evaluate the scalability of heatmap-based methods, we test the GLOP (Ye et al., 2024) and AGFN (Zhang et al., 2025) on large-scale capacitated VRP (CVRP) instances using an NVIDIA GeForce RTX 3090 Ti GPU with 24GB memory, as shown by the solid lines in Fig. 1. It is shown

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.

that GLOP and AGFN can solve problem instances with up to N=20K nodes, but fail at larger scales due to out-of-memory.

The quadratic intermediate representations arise mainly at two stages of the inference on large-scale instances. First, the graph sparsification of heatmap-based method requires a dense $N \times N$ pairwise matrix to identify the sparse candidate graph. Second, the scores predicted by the neural model for candidate pairs are subsequently stored in a dense $N \times N$ heatmap. Consequently, the resulting quadratic growth in memory consumption becomes a critical bottleneck to scale larger instances. In addition, the decoder of the heatmap-based method builds the solution through a CPU-controlled loop, resulting in a large runtime overhead. On one hand, in each step, these operations launch multiple fine-grained GPU kernels and exchange intermediate results through the GPU global memory. On the other hand, across steps, the CPU repeatedly dispatches the kernels to advance the decoding process. Therefore, when scaling from small-scale instance training to large-scale instance inference, intermediate results access and repeated kernels launches become significant bottlenecks in the decoding process.

To address these issues, we propose SIF, a Scalable Inference Framework by optimizing intermediate representations and accelerating the execution of decoding. As shown by the dashed lines in Fig. 1, SIF can enable GLOP and AGFN to solve instances with up to 100K and 50K nodes, respectively. Specifically, SIF employs a memory-efficient search strategy to select candidate sparse graphs and stores the predicted scores only for the retained candidate node pairs, thereby aligning the heatmap representation with the sparse decision space. SIF further introduces a GPU-resident fused decoding mechanism, which executes the entire decoding loop within a single GPU kernel, eliminating repeated CPU-side kernel launches and reducing intermediate results accesses. Moreover, SIF uses a low-overhead geometry-based method to preserve solution quality when no heatmap-guided candidate remains feasible.

Our main contributions are summarized as follows.

- We propose SIF, a scalable inference framework that improves the scalability of heatmap-based methods by optimizing intermediate representations and decoding execution.
- We develop a memory-efficient search strategy that constructs candidate graphs without dense pairwise representations and stores predicted scores only for retained candidate node pairs.
- We introduce a GPU-resident fused decoding mechanism to reduce CPU-side launch and intermediate results access overhead.
- We integrate SIF into four typical heatmap-based methods. The experimental results on large-scale instance datasets demonstrate that SIF can extend the problem-solving scale of these methods to 50K or 100K, while significantly improving the inference efficiency.

2 Related Work

Heatmap-based methods have become an important paradigm for large-scale VRP instances, where a neural model predicts edge- or node-level compatibility scores that serve as structural priors for subsequent solution construction. Early approaches directly predict edge-selection likelihoods and combine the resulting heatmaps with search procedures such as Monte Carlo tree search or sampling (Fu et al., 2021; Qiu et al., 2022). Recent methods have substantially diversified the way these learned heatmaps are exploited.

One line of work uses heatmaps to guide problem decomposition. GLOP (Ye et al., 2024) learns global partitioning structures and solves the resulting subproblems with local construction policies, while HLGP (Pan et al., 2025) further introduces hierarchical partition refinement to mitigate errors introduced by coarse global decomposition. UDC (Zheng et al., 2024) generalizes this paradigm through a unified divide–conquer–reunion framework for large-scale combinatorial optimization. Another line learns more expressive heatmap distributions with generative models. GFACS (Kim et al., 2025) combines GFlowNets with ant-colony sampling, AGFN (Zhang et al., 2025) introduces adversarial training to improve the learned routing distribution, and HBG-GFACS (Zhang & Cao, 2025) further enhances GFlowNet training by jointly exploiting trajectory- and transition-level balance objectives. Diffusion-based approaches such as DIFUSCO (Sun & Yang, 2023) generate

edge-selection heatmaps through iterative denoising, whereas COExpander (Ma et al., 2025) adopts global structural predictions and progressively expands partial solutions through adaptive variable selection. More recently, MaskCO (Chen et al., 2026) learns solution-conditioned structural representations through masked reconstruction and performs iterative mask-and-reconstruct refinement during inference.

Existing studies primarily focus on improving heatmap quality, sampling diversity, or solution refinement. However, scaling these methods to substantially larger-scale instances requires addressing the memory and runtime costs of the underlying inference pipeline, which remain less explored.

3 PRELIMINARIES

3.1 PROBLEM FORMULATION

We consider routing problems defined on a complete graph $G = (V, E)$, where V denotes the set of nodes and E contains pairwise edges weighted by travel costs. For the travelling salesman problem (TSP), $V = \{v_1, \ldots, v_N\}$ contains N nodes. The objective is to find a minimum-cost Hamiltonian cycle that visits every node exactly once:

$$\min_{\pi} L(\pi) = \sum_{t=1}^{N} d\left(v_{\pi_t}, v_{\pi_{t+1}}\right), \qquad \pi_{N+1} = \pi_1, \tag{1}$$

where π is a permutation of the nodes and $d(\cdot, \cdot)$ denotes the pairwise travel cost, typically the Euclidean distance. The CVRP extends the TSP by introducing a depot v_0, customer demands q_i, and a vehicle capacity Q. Given $V = \{v_0, v_1, \ldots, v_N\}$, the objective is to find a set of minimum-cost depot-to-depot routes $\mathcal{R}$ that visits every customer exactly once while satisfying the capacity constraint:

$$\min_{\mathcal{R}} \sum_{r \in \mathcal{R}} L(r), \qquad \text{s.t.} \quad \sum_{v_i \in r} q_i \leq Q, \ \forall r \in \mathcal{R}, \tag{2}$$

where $L(r)$ denotes the travel cost of route r.

3.2 THE INFERENCE PIPELINE OF HEATMAP-BASED METHODS

Despite differences in model design, heatmap-based methods generally follow a common inference pipeline. In the following, we use CVRP as a running example to illustrate this process.

Sparse graph construction. Given a CVRP instance G, existing methods first construct a dense pairwise matrix using certain geometric criterion, and then apply Top-K selection to identify the candidate neighbor set $\mathcal{N}_K(i)$ for each node i, forming the sparse candidate graph $G_K = (V, E_K)$.

Heatmap generation and representation. The graph neural network takes the sparse candidate graph as input and predicts the scores of the candidate node pairs. Subsequently, these scores are filled into the dense heatmap $\mathbf{H}$: the candidate position H_{ij} stores the corresponding predicted score, while non-candidate positions are set to zero.

Solution construction. Based on the heatmap, the decoder $\mathcal{D}$ constructs an initial solution $\boldsymbol{\pi}$ autoregressively. We describe greedy decoding as a running example. At step t, the decoding state z_t records the current node v_{π_t}, visited customers, and remaining vehicle capacity. Each step consists of four sub-operations: candidate retrieval obtains the candidate neighbors of v_{π_t} and their heatmap scores; feasibility checking identifies candidates that satisfy the problem constraints given z_t; node selection chooses the feasible candidate with the highest score; and state update records the selected node, updates its visitation status, and adjusts the remaining capacity. This decoding step is repeated until the solution is complete. The complete decoding process can be found in the Appendix A.

4 METHODOLOGY

We propose SIF, an efficient inference framework that improves the scalability of heatmap-based methods by addressing their representation and execution bottlenecks. As illustrated in Fig. 2, SIF comprises three complementary components: (1) memory-efficient sparse graph construction, (2) sparse-aligned heatmap representation, and (3) GPU-resident fused decoding.

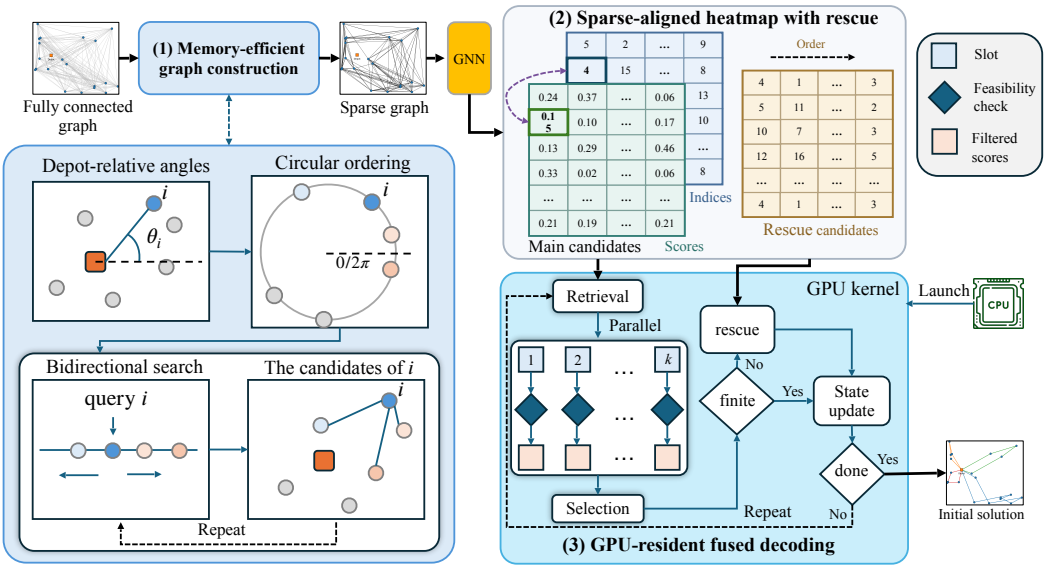


Figure 2: The overview of our proposed SIF.

4.1 Memory-efficient Exact Sparse Graph Construction

Existing heatmap-based methods rely on dense pairwise matrices to construct sparse graphs, so the sparsification process itself still requires $\mathcal{O}(N^2)$ representations. These methods typically use directional cosine similarity or Euclidean distance as the sparsification criterion. To avoid quadratic intermediate representations, we adopt memory-efficient graph construction strategies tailored to the two sparsification criterion, as described below.

Angular-order sparsification. For the cosine similarity criterion, we identify candidate neighbors through an equivalent angular search. Let $\mathbf{x}_0$ denote the depot coordinate and $\mathbf{d}_i = \mathbf{x}_i - \mathbf{x}_0$ the depot-relative position of customer i, with $\|\mathbf{d}_i\|_2 > 0$. The original criterion selects the K other customers with the highest depot-relative cosine similarity:

$$s_{ij} = \frac{\mathbf{d}_i^\top \mathbf{d}_j}{\|\mathbf{d}_i\|_2 \|\mathbf{d}_j\|_2}. \tag{3}$$

Let $\theta_i \in [0, 2\pi)$ denote the polar angle of $\mathbf{d}_i$. We define the circular angular distance between customers i and j as

$$d_{\text{ang}}(i,j) = \min\left\{|\theta_i - \theta_j|, 2\pi - |\theta_i - \theta_j|\right\}. \tag{4}$$

The following proposition establishes that candidate selection by cosine similarity can be expressed in terms of circular angular distance.

Proposition 1 (Equivalence of directional and angular ranking)**.** *For any customer i, ranking the other customers by descending depot-relative cosine similarity is equivalent to ranking them by ascending circular angular distance.*

Proof. For any two customers i and j, their depot-relative position vectors can be expressed in polar coordinates as

$$\mathbf{d}_i = r_i(\cos\theta_i, \sin\theta_i), \qquad \mathbf{d}_j = r_j(\cos\theta_j, \sin\theta_j), \tag{5}$$

where $r_i = \|\mathbf{d}_i\|_2 > 0$ and $r_j = \|\mathbf{d}_j\|_2 > 0$. Substituting these expressions into the definition of cosine similarity gives

$$\begin{aligned} s_{ij} &= \frac{\mathbf{d}_i^\top \mathbf{d}_j}{\|\mathbf{d}_i\|_2 \|\mathbf{d}_j\|_2} = \frac{r_i r_j(\cos\theta_i \cos\theta_j + \sin\theta_i \sin\theta_j)}{r_i r_j} \\ &= \cos(\theta_i - \theta_j) = \cos(d_{\text{ang}}(i,j)). \end{aligned}$$

The third equality follows from the cosine difference identity, and the final equality follows from the evenness and 2π-periodicity of cosine. Since $d_{\text{ang}}(i,j) \in [0,\pi]$ and cosine is strictly decreasing on this interval, for any two candidate customers j and k, we have

$$s_{ij} \geq s_{ik} \quad \Longleftrightarrow \quad d_{\text{ang}}(i,j) \leq d_{\text{ang}}(i,k).$$

Thus, ranking customers by descending cosine similarity is equivalent to ranking them by ascending circular angular distance. □

Based on this equivalence, we sort the customers once by polar angle. For each customer i, we search in both directions along the resulting circular order, accounting for wraparound at 0 and 2π, to identify the K nearest other customers under d_{ang}. This procedure constructs $\mathcal{N}_K(i)$ and hence E_K without materializing an $N \times N$ similarity matrix.

Blocked KNN sparsification. For the Euclidean distance criterion, the original sparsification strategy first constructs a distance matrix $\mathbf{D}$, where $D_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$. It then selects the K smallest entries in each row, excluding the diagonal, to obtain the candidate neighbors:

$$\mathcal{N}_K(i) = \text{TopK}_{j \neq i}(-D_{ij}). \tag{6}$$

We avoid this quadratic representations by processing candidate nodes in blocks of size B. For each block, we compute distances from all nodes to its candidates, excluding self-pairs. Each node merges these candidates with its retained neighbors and keeps the K closest. After each update, the retained neighbors are the closest among all candidates processed so far, ensuring exact KNN selection once all blocks have been processed.

4.2 Sparse-Aligned Heatmap Representation with rescue

Although scores are predicted only for candidate node pairs, the existing heatmap representation still allocates representations for all node pairs, including zero-valued entries for non-candidate pairs, thereby requiring $\mathcal{O}(N^2)$ representations. To avoid this redundant representation, we directly represent the heatmap on the sparse candidate graph by retaining only candidate indices and their associated scores. For each node i, we store the candidate indices and predicted scores in two aligned vectors:

$$\mathbf{C}_i = (c_{i1}, \ldots, c_{iK}), \qquad \mathbf{S}_i = (s_{i1}, \ldots, s_{iK}), \tag{7}$$

where c_{ik} denotes the index of the k-th candidate neighbor of node i, and s_{ik} stores its corresponding predicted score. We refer to the set of neighbors of node i in the sparse candidate graph as the main candidate set $\mathcal{N}_K(i)$. Note that the above transformation only changes the representations method of the heatmap, while the model's predicted scores for the candidate node pairs remain unchanged.

Because candidate selection ignores problem constraints, all retained candidates for a node may become infeasible during solution construction. To provide alternative guidance when no main candidate is feasible, we construct a geometry-based rescue list during graph sparsification. Geometric proximity provides a local preference for route construction: Euclidean proximity (Santana et al., 2022) favors nodes with lower connection costs, while angular proximity (Gillett & Miller, 1974) favors customers in similar directions from the depot. These criteria guide candidate selection when model scores are unavailable. For each node i, we select up to R geometric neighbors outside its main candidate set $\mathcal{N}_K(i)$, excluding the node itself, and store their indices in ascending order of geometric distance under the corresponding sparsification criterion:

$$\mathbf{Q}_i = (q_{i1}, \ldots, q_{iR_i}), \qquad R_i \leq R. \tag{8}$$

Rescue candidates do not participate in model computation. By reusing the geometric search performed during graph sparsification, the rescue mechanism incurs little additional construction overhead.

4.3 GPU-resident Fused Decoding

To improve decoding efficiency, we propose GPU-resident fused decoding, which builds on the sparse-aligned heatmap to organize decoding as a sequence of fused state transitions within a single GPU kernel. Each transition fuses candidate retrieval, feasibility checking, node selection, and state update, while the kernel itself controls iteration and termination. The aligned candidate indices

and scores provide a bounded set of candidate slots that can be processed in parallel within each transition.

Formally, we represent a decoding step as a fused state transition:

$$(a_t, z_{t+1}) = \mathcal{F}_{\text{fused}}\left(z_t; \mathcal{H}_{SA}\right), \tag{9}$$

where $\mathcal{H}_{SA}$ denotes sparse-aligned heatmap, a_t denotes selected node and z_{t+1} denotes the decoding state at step $t+1$.

Specifically, given the current node v_{π_t}, the kernel first locates the corresponding row in the sparse-aligned heatmap $\mathcal{H}_{\text{SA}}(\pi_t)$, and then obtains the aligned main candidate indices and their scores from it. Since the feasibility of each candidate depends only on its demand and the current state z_t, the K candidate slots can be evaluated in parallel. For the k-th slot, the decoder reads candidate node $v_{c_{\pi_t k}}$ and its predicted score $s_{\pi_t k}$, and defines the filtered score as

$$\widetilde{s}_{\pi_t k} = \begin{cases} s_{\pi_t k}, & v_{c_{\pi_t k}} \text{ is feasible under } z_t, \\ -\infty, & \text{otherwise.} \end{cases} \tag{10}$$

After the parallel feasibility evaluation, the kernel performs a parallel maximum reduction over the filtered scores:

$$k_t^\star = \arg\max_{1 \le k \le K} \widetilde{s}_{\pi_t k}, \tag{11}$$

where $k_t^\star$ denotes the highest-scoring feasible slot in the main candidate row. If the maximum filtered score is finite, the corresponding node $v_{c_{\pi_t k_t^\star}}$ is selected. Otherwise, the kernel activates the rescue mechanism to select next node. If neither candidate tier produces a feasible node, the decoder returns to the depot or uses the global fallback, which selects a feasible node according to the default node-index order. Once the next node a_t is determined, the kernel updates the decoding state to obtain z_{t+1}.

After each state update, the kernel checks whether all customers have been served. If unserved customers remain, it proceeds directly to the next decoding step using the updated state. Otherwise, it completes the final depot return and output the constructed solution.

5 Experiments

5.1 Experimental Setup

Baseline. We integrate SIF into four representative heatmap-based methods from two categories: decomposition-based methods GLOP and HLGP, and GFlowNet-based methods AGFN and HBG-GFACS. We evaluate solution quality under the original benchmark settings of each method and assess scalability on larger instances. Each method and its SIF-enhanced counterpart are evaluated on the same instances under identical problem settings.

Datasets. For CVRP, we follow the original evaluation settings of the corresponding methods and use different benchmark scales accordingly. We evaluate solution quality on CVRP200–CVRP1000 for AGFN and HBG-GFACS, and on CVRP5K–CVRP20K for GLOP and HLGP. Scalability is further evaluated on CVRP50K and CVRP100K. For TSP, solution quality is evaluated on TSP200–TSP1000, while scalability is assessed on TSP50K and TSP100K. The large-scale CVRP50K, CVRP100K, TSP50K, and TSP100K instances are taken from the datasets provided by (Chen et al., 2025). Detailed dataset configurations are provided in the Appendix B.

Metrics. For profiling, we distinguish two stages: *graph construction* and *solution construction*, with latencies denoted by T_{graph} and T_{solve}, respectively. Here, T_{graph} measures sparse graph construction, while T_{solve} includes heatmap prediction and representation, and decoding for an initial solution. We report end-to-end inference time (Time) and peak GPU memory usage (Mem.), covering both stages and the method-specific solution-improvement stage. Moreover, we measure solution quality by the objective value (Obj.), defined as the total route length.

Inference. All experiments use one NVIDIA RTX 3090 Ti (24GB) with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz. To assess the maximum solvable instance scale under a fixed GPU memory budget, we set the batch size to 1. For each method, we use the released checkpoint and the default decoding and solution-improvement configurations from its original implementation. The detail test configurations can be found in the Appendix C.

Table 2: Solution quality and efficiency comparison on CVRP5K, 10K and 20K. Δ_{Obj} denotes the absolute reduction in objective value, where a positive value indicates improved solution quality. $\mathrm{Spd.} = \mathrm{Time}_{\mathrm{Ori.}}/\mathrm{Time}_{\mathrm{SIF}}$ denotes the inference speedup over the original inference framework.

Method	CVRP5K				CVRP10K				CVRP20K			
	Obj.	Δ_{Obj}	Time(s)	Spd.	Obj.	Δ_{Obj}	Time(s)	Spd.	Obj.	Δ_{Obj}	Time(s)	Spd.
GLOP	141.50	0.31	6.50	2.74×	266.60	1.38	10.48	2.50×	597.35	25.59	30.22	2.87×
GLOP+SIF	141.19		2.37		265.22		4.19		571.76		10.52	
HLGP	130.28	0.01	32.14	3.29×	244.28	1.37	55.58	3.71×	549.16	24.05	101.96	3.53×
HLGP+SIF	130.27		9.78		242.91		14.99		525.11		28.90	

Table 3: Solution quality and efficiency comparison on TSP and CVRP instances.

Task	Method	200				500				1000			
		Obj.	Δ_{Obj}	Time(s)	Spd.	Obj.	Δ_{Obj}	Time(s)	Spd.	Obj.	Δ_{Obj}	Time(s)	Spd.
TSP	AGFN	11.85	0.00	0.48	48.00×	18.78	-0.03	1.25	125.00×	27.08	0.02	2.35	117.50×
	AGFN+SIF	11.85		0.01		18.81		0.01		27.06		0.02	
	HBG-GFACS	12.24	0.00	34.45	5.45×	22.44	0.01	72.69	5.54×	37.64	0.00	133.14	6.22×
	HBG-GFACS+SIF	12.24		6.32		22.43		13.12		37.64		21.41	
CVRP	AGFN	30.74	0.14	0.87	14.50×	69.45	0.06	2.04	29.14×	129.64	0.03	4.13	29.50×
	AGFN+SIF	30.60		0.06		69.39		0.07		129.61		0.14	
	HBG-GFACS	32.28	0.01	16.07	69.87×	71.93	0.01	39.54	127.55×	133.84	0.01	81.95	182.11×
	HBG-GFACS+SIF	32.27		0.23		71.92		0.31		133.83		0.45	

5.2 Main Results

5.2.1 Scalability Analysis

We evaluate the scalability of four representative heatmap-based methods under a fixed GPU memory budget of 24 GB. As shown in Table 1, the largest instances these methods can solve under their original inference frameworks contain 20K nodes, while larger tested instances encounter out-of-memory (OOM) errors.

Table 1: Scalability improvements enabled by SIF across four representative heatmap-based methods. OOM denotes out-of-memory.

Problem	Method	50K		100K	
		Obj.	Time (s)	Obj.	Time (s)
CVRP	GLOP	OOM		OOM	
	GLOP + SIF	1264.93	46.58	2137.94	131.14
	HLGP	OOM		OOM	
	HLGP + SIF	1155.71	67.80	2101.69	175.71
	AGFN	OOM		OOM	
	AGFN + SIF	10754.97	95.11	OOM	
	HBG-GFACS	OOM		OOM	
	HBG-GFACS + SIF	1681.22	37.06	3380.93	138.05
TSP	AGFN	OOM		OOM	
	AGFN + SIF	206.51	13.21	303.35	48.05
	HBG-GFACS	OOM		OOM	
	HBG-GFACS + SIF	936.36	149.01	1582.43	296.53

For CVRP, SIF enables GLOP, HLGP, and HBG-GFACS to solve CVRP100K and AGFN to solve CVRP50K, increasing their supported CVRP instance sizes by factors of 5 and 2.5, respectively. The OOM failures of the original methods at 50K and 100K preclude direct runtime comparisons at these scales. We therefore compare against INViT (Fang et al., 2024) and TTPL (Chen et al., 2025), which can solve instances at these scales. As shown in Table 4, GLOP and HLGP equipped with SIF achieve a better trade-off between runtime and solution quality than INViT and TTPL. The results demonstrate the advantages of SIF inference framework for improving the scalability of heatmap-based methods.

5.2.2 Solution Quality and efficiency

We examine whether the scalability gain is achieved without degrading solution quality. As shown in Tables 2 and 3, SIF preserves or slightly improves the solution quality of all evaluated solvers on scales where the original implementations remain executable. For GLOP and HLGP, the improvement becomes more evident on CVRP20K, reducing the objective from 597.35 to 571.76 and from 549.16 to 525.11, respectively. We attribute this improvement to the rescue mechanism in SIF, which avoids degenerate masked decoding where tied scores may be resolved arbitrarily. For AGFN and HBG-GFACS, SIF yields nearly identical results on TSP200–1000 and CVRP200–1000. Overall, these results show that SIF substantially improves scalability while preserving the solution quality of the original methods.

Table 4: The performance of INViT and TTPL on the instances of the CVRP50K and 100K problem scales.

Method	CVRP50K		CVRP100K	
	Obj.	Time	Obj.	Time
INViT-3V	1331.1	1.5h	2683.4	5.8h
TTPL	1105.6	7.0m	2157.5	7.7m

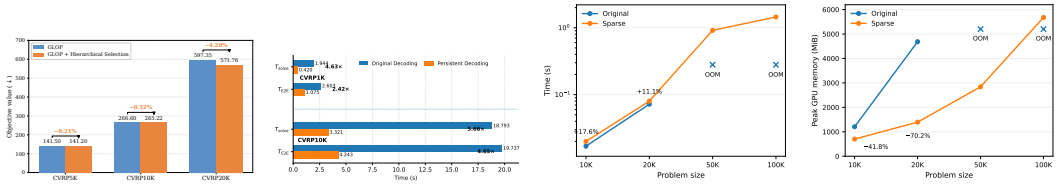
5.2.3 Stage-wise Runtime Analysis

We further analyze the computational efficiency of the proposed framework by separately profiling graph construction and solution construction across different problem scales, as shown in Table 5. On CVRP, SIF consistently reduces the cost of solution construction and substantially improves end-to-end efficiency. For GLOP, T_{solve} is reduced from 9.55s to 3.25s on CVRP10K and from 29.67s to 9.93s on CVRP20K, yielding end-to-end speedups of $2.50\times$ and $2.87\times$, respectively. The gain is even more pronounced for AGFN: T_{E2E} decreases from 37.43s to 4.13s on CVRP10K and from 92.68s to 12.78s on CVRP20K, corresponding to $9.07\times$ and $7.25\times$ speedups. Detailed analyses of other models can be found in the Appendix D.

Table 5: Inference efficiency and peak memory usage across tasks and scales.

Task	Scale	Method	Latency (s)			Mem. (GiB)
			T_{graph}	T_{solve}	T_{E2E}	
CVRP	10K	GLOP	0.02	9.55	10.48	2.22
		GLOP+SIF	0.02	3.25	4.19	2.29
	20K	GLOP	0.07	29.67	30.22	4.58
		GLOP+SIF	0.08	9.93	10.52	4.57
	50K	GLOP	OOM	OOM	OOM	OOM
		GLOP+SIF	0.91	41.48	46.59	11.21
	100K	GLOP	OOM	OOM	OOM	OOM
		GLOP+SIF	1.45	123.92	131.14	22.40
TSP	5K	AGFN	0.02	8.85	8.99	0.84
		AGFN+SIF	0.02	0.65	0.95	0.75
	10K	AGFN	0.05	19.32	19.56	2.29
		AGFN+SIF	0.05	0.82	1.21	1.50
	50K	AGFN	OOM	OOM	OOM	OOM
		AGFN+SIF	0.34	11.74	13.21	7.46
	100K	AGFN	OOM	OOM	OOM	OOM
		AGFN+SIF	1.20	44.64	48.05	14.91

5.3 Ablation Study



(a) Rescue (b) Decoding (c) Graph time (d) GPU memory

Figure 3: Ablation study of SIF on GLOP: (a) rescue with $K = 200$; (b) original versus GPU-resident decoding using identical sparse inputs; (c, d) dense versus sparse graph construction.

Effect of rescue mechanism. To isolate the contribution of the rescue mechanism, we incorporate it into the original GLOP decoder while keeping the rest of the inference pipeline unchanged. Fig. 3a reports the results on CVRP5K, CVRP10K, and CVRP20K with a fixed sparsification factor of $K = 200$. Rescue mechanism consistently improves solution quality across all problem scales, reducing the objective from 141.50 to 141.19 on CVRP5K, from 266.60 to 265.22 on CVRP10K, and from

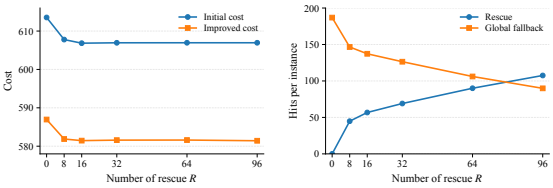
597.35 to 571.76 on CVRP20K. The improvement becomes more pronounced as the problem scale increases, suggesting that rescue mechanism provides more robust decision guidance for large-scale decoding.

Effect of memory-efficient sparse graph construction. To evaluate the proposed sparse graph construction, we use GLOP as a representative model and compare it with the original dense construction scheme, as shown in Figs. 3c and 3d. Our method introduces only minor candidate-selection overhead while reducing peak GPU memory by 41.8% ($\frac{1.18-0.68}{1.18} \times 100\%$) and 70.3%($\frac{4.57-1.34}{4.57} \times 100\%$) on CVRP10K and CVRP20K, respectively. Moreover, the dense scheme runs out of memory at CVRP50K, whereas ours scales to CVRP100K, demonstrating substantially improved scalability.

Effect of GPU-resident fused decoding. To isolate the effect of the decoder, we adopt the same sparse graph and sparse heatmap representation and compare two decoding schemes: the original decoding and our proposed GPU-resident decoding. As shown in Fig. 3b, GPU-resident fused decoding achieves 4.63× and 5.66× decoding speedups on CVRP1K and CVRP10K, respectively, while reducing end-to-end runtime by 58.72% and 78.50%. These results show that the original decoding is a major runtime bottleneck and that GPU-resident decoding substantially improves efficiency.

5.4 Additional Analysis

Space complexity analysis. For Angular-order sparsification, it requires $\mathcal{O}(N + NK)$ space for the angular ordering and retained candidates. Blocked KNN requires $\mathcal{O}(NB + NK)$ peak space for the current distance block and retained candidates. For sparse-aligned heatmap, the aligned candidate indices and scores require $\mathcal{O}(NK)$ representations. Overall, the space complexity of our method is $\mathcal{O}(NB + N(K + R))$, the detail complexity analysis can be found in the Appendix E.



(a) Solution quality.

(b) Rescue and global fallback hits.

Figure 4: Sensitivity to the number of rescue candidates R on CVRP20K with $K = 200$. Increasing R progressively replaces global fallback decisions with rescue candidates, while the solution cost quickly saturates.

Sensitivity analysis of R. To study the effect of the rescue size R, we fix $K = 200$ and then evaluate $R \in \{0, 8, 16, 32, 64, 96\}$ on CVRP20K dataset. As shown in Fig. 4a, increasing R from 0 to 16 substantially improves solution quality, while further increasing R yields marginal gains. As shown in Fig. 4b, on CVRP20K with $K = 200$, global fallback hits decrease from 186.88 to 89.88 per instance as R increases from 0 to 96, with rescue hits increasing from 0 to 107.50.

6 Conclusion

We presented SIF, a sparse inference framework that improves the scalability of heatmap-based neural solvers without modifying their learned models. SIF avoids dense pairwise representations through memory-efficient graph construction and sparse-aligned heatmap representation, and reduces repeated kernel launches through GPU-resident fused decoding. A geometry-based rescue mechanism provides alternative guidance when main candidates become infeasible, mitigating the quality degradation caused by index-biased fallback. Experiments on four representative methods demonstrate improved scalability and inference efficiency while maintaining comparable solution quality at evaluated scales. These results show that realizing the benefits of sparsification requires aligning both intermediate representations and decoding execution with the sparse decision space.

However, since our method utilizes blocked KNN for sparsification, the time complexity in calculating pairwise distances is high, resulting in a bottleneck when expanding to larger-scale instances. Future research will explore directly learning sparse candidate graphs from instances and guided by the quality of the final solution to reduce the cost of constructing sparse graphs, thereby achieving large-scale scalability.

REFERENCES

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. doi: 10.1016/j.ejor.2020.07.063.

Lvda Chen, Yang Li, and Junchi Yan. MaskCO: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

Yuanyao Chen, Rongsheng Chen, Fu Luo, and Zhenkun Wang. Improving generalization of neural combinatorial optimization for vehicle routing problems via test-time projection learning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. Invit: A generalizable routing problem solver with invariant nested view transformer. In *Forty-first International Conference on Machine Learning*, 2024.

Alexandre M. Florio, Nabil Absi, and Dominique Feillet. Routing electric vehicles on congested street networks. *Transportation Science*, 55(1):238–256, 2021. doi: 10.1287/trsc.2020.1004.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7474–7482, 2021.

Vishal Gaur and Marshall L. Fisher. A periodic inventory routing problem at a supermarket chain. *Operations Research*, 52(6):813–822, 2004. doi: 10.1287/opre.1040.0150.

Billy E. Gillett and Leland R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974. doi: 10.1287/opre.22.2.340.

Minsu Kim, Sanghyeok Choi, Hyeonah Kim, Jiwoo Son, Jinkyoo Park, and Yoshua Bengio. Ant colony sampling with gflownets for combinatorial optimization. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 469–477, 2025.

Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. Coexpander: Adaptive solution expansion for combinatorial optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 42130–42164. PMLR, 2025.

Yuxin Pan, Ruohong Liu, Yize Chen, Cao Zhiguang, and Fangzhen Lin. Hierarchical learning-based graph partition for large-scale vehicle routing problems. In *The 24th International Conference on Autonomous Agents and Multi-Agent Systems*, 2025.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, volume 35, pp. 25531–25546, 2022.

Ítalo Santana, Andrea Lodi, and Thibaut Vidal. Neural networks for local search and crossover in vehicle routing: A possible overkill?, 2022.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 3706–3731, 2023. doi: 10.52202/075280-0164.

Stacy A. Voccia, Ann Melissa Campbell, and Barrett W. Thomas. The same-day delivery problem for online purchases. *Transportation Science*, 53(1):167–184, 2019. doi: 10.1287/trsc.2016.0732.

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

Ni Zhang and Zhiguang Cao. Hybrid-balance gflownet for solving vehicle routing problems. In *Advances in Neural Information Processing Systems*, 2025.

Ni Zhang, Jingfeng Yang, Zhiguang Cao, and Xu Chi. Adversarial generative flow network for solving vehicle routing problems. *The Thirteenth International Conference on Learning Representations*, 2025.

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*, volume 37, pp. 6081–6125, 2024.

A THE CPU-CONTROLLED AUTOREGRESSIVE LOOP DECODING PROCESS

The scalability of heatmap-based methods is limited by decoding efficiency. These methods typically follow an autoregressive decoding paradigm driven by a CPU-controlled loop. This execution scheme introduces overhead at two levels. Within each step, the CPU dispatches multiple fine-grained GPU kernels for candidate retrieval, feasibility checking, node selection, and state update. These kernels exchange intermediate results through GPU global memory, introducing additional overhead from repeated writes and reads. Across decoding steps, the CPU repeatedly launches GPU kernels, causing cumulative launch overhead.

Algorithm 1 Conventional CPU-controlled decoding. $\mathrm{KernelLaunch}$ denotes CPU dispatch of one or more GPU kernels for the specified operation. States and intermediate tensors remain on the GPU.

Input: Heatmap $\mathbf{H}$ and instance
Output: Complete solution $\mathcal{R}$
CPU-controlled execution:
Initialize decoding state z_0 on the GPU; $t \leftarrow 0$
Read the initial completion flag done from the GPU
while $\neg\mathrm{done}$ **do**
  $\mathbf{h}_t \leftarrow \mathrm{KernelLaunch}(\mathrm{Retrieve}, \mathbf{H}, v_t)$
  $\tilde{\mathbf{h}}_t \leftarrow \mathrm{KernelLaunch}(\mathrm{Mask}, \mathbf{h}_t, z_t)$
  $a_t \leftarrow \mathrm{KernelLaunch}(\mathrm{Select}, \tilde{\mathbf{h}}_t)$
  $z_{t+1} \leftarrow \mathrm{KernelLaunch}(\mathrm{Update}, z_t, a_t)$
  Synchronize and read the GPU-computed completion flag into done
  $t \leftarrow t + 1$
end while
return $\mathcal{R}$

B DATASET CONFIGURATIONS

For CVRP, AGFN and HBG-GFACS are evaluated on CVRP200, CVRP500, and CVRP1000, with 128 instances per scale and vehicle capacity $Q = 50$. GLOP and HLGP are evaluated on CVRP5K, CVRP10K, and CVRP20K. The vehicle capacity is set to $Q = 200$ for CVRP5K and CVRP10K and to $Q = 300$ for CVRP20K. For scalability evaluation, we use CVRP50K and CVRP100K, each containing 16 instances with $Q = 300$. These large-scale instances are taken from the datasets provided by TTPL (Chen et al., 2025). For TSP, solution quality is evaluated on TSP200, TSP500, and TSP1000, with 128 instances per scale. For scalability evaluation, we use TSP50K and TSP100K, each containing 16 instances, taken from the datasets provided by TTPL.

C TESTING CONFIGURATION

In this section, we provide a detailed description of the test configuration for each baseline. All experiments were conducted on a NVIDIA GeForce RTX 3090 Ti GPU with 24 GB of video memory. For all the experiments, the batch size was uniformly set to 1.

- GLOP: We use the pretrained cvrp-2000 model released by the authors of GLOP. To ensure a fair comparison, we sample one initial solution for each instance under all evaluated configurations. We set the sparsification factor to $K = 200$. For SIF, we additionally select $R = 16$ rescue candidates for each node. During the improvement stage, we use the Reviser model provided in the original GLOP implementation, with a revision length of 20 and 10 revision iterations.
- HLGP: We use the pretrained cvrp-1000 model released by the authors of HLGP. We disable the data augmentation technique during evaluation because it substantially increases inference time. We set the sparsification factor to $K = 200$. For SIF, we additionally select $R = 16$ rescue candidates for each node. We set the number of local repartition iterations to 5.
- AGFN: We use models pretrained on instances of sizes 200, 500, and 1,000. For each instance, we generate 100 diverse solutions to improve solution quality. We set the sparsification factor to $K = N/5$, where N denotes the number of nodes, and set the hybrid-sampling probability to 0.05. For the scalability experiments, we use the model pretrained on instances of size 1,000 and fix the sparsification factor at $K = 200$.
- HBG-GFACS: We use the model pretrained on instances of size 100. We set the sparsification factor to $K = N/5$. For the scalability experiments, we fix the sparsification factor at $K = 200$. We disable local search in all experiments to exclude its influence on the comparison. We use 100 ants and perform 10 iterations.
- INViT and TTPL: Because the evaluation uses the same datasets, we report the results provided in the TTPL paper.

D DETAILED STAGE-WISE ANALYSES OF OTHER BASELINE

Table 6 provides a detailed stage-wise analysis of AGFN on large-scale CVRP instances. From CVRP5K to CVRP20K, SIF reduces T_{solve} from 19.75s, 33.90s, and 82.41s to 0.12s, 0.59s, and 2.50s, corresponding to speedups of $164.6\times$, $57.5\times$, and $33.0\times$, respectively. As a result, the end-to-end latency is reduced by $12.55\times$, $9.06\times$, and $7.25\times$ at the three scales. SIF also lowers peak GPU memory by 59.4%, 65.5%, and 68.5%, with the relative saving increasing as the problem size grows. These results confirm that sparse-aligned heatmap representation and GPU-resident fused decoding effectively remove the major computation and memory bottlenecks in solution construction.

At CVRP50K, the original AGFN pipeline runs out of memory, whereas AGFN+SIF successfully completes inference using 19.47 GiB, extending the maximum solvable scale from 20K to 50K under the same 24 GB memory budget. After accelerating solution construction, graph construction becomes the dominant runtime component, accounting for 75–79% of the end-to-end latency for AGFN+SIF across the evaluated large-scale instances. This bottleneck shift is consistent with blocked KNN sparsification: it avoids materializing the quadratic distance matrix and substantially reduces peak memory, while retaining the pairwise distance computation required for exact neighbor retrieval.

E SPACE COMPLEXITY ANALYSIS

Let N denote the number of nodes, K the number of main candidates per node, B the candidate block size, and R the maximum number of rescue candidates per node. We assume fixed-dimensional node coordinates and constant-size representations for each index and score. The analysis includes retained candidates and temporary construction buffers, but excludes model parameters and neural-network activations.

Table 6: Additional inference efficiency and memory results for CVRP across problem scales. Latency is reported in seconds and peak memory in GiB. OOM denotes out-of-memory.

Task	Scale	Method	Latency(s)			Mem. (GiB)
			T_{graph}	T_{solve}	T_{E2E}	
CVRP	5K	AGFN	1.30	19.75	21.33	0.69
		AGFN+SIF	1.30	0.12	1.70	0.28
	10K	AGFN	3.03	33.90	37.43	2.67
		AGFN+SIF	3.03	0.59	4.13	0.92
	20K	AGFN	9.60	82.41	92.68	10.55
		AGFN+SIF	9.60	2.50	12.78	3.32
	50K	AGFN	OOM	OOM	OOM	OOM
		AGFN+SIF	75.56	17.82	95.11	19.47

Angular-order sparsification. Computing and sorting the polar angles requires storing N angles and their associated node indices, using $\mathcal{O}(N)$ space. The circular search maintains only a constant number of pointers per node, requiring at most $\mathcal{O}(N)$ additional space. Storing the K selected neighbors for each of the N nodes requires $\mathcal{O}(NK)$ space. Therefore, the total space complexity is

$$S_{\text{angular}} = \mathcal{O}(N) + \mathcal{O}(NK) = \mathcal{O}(N + NK). \tag{12}$$

No pairwise similarity matrix is materialized.

Blocked KNN sparsification. For each block of at most B candidate nodes, distances from all N nodes to the current candidates form an $N \times B$ buffer, requiring $\mathcal{O}(NB)$ space. The retained neighbor indices and distances occupy $\mathcal{O}(NK)$ space. Merging the current block with the retained candidates requires at most $\mathcal{O}(N(B+K))$ temporary representations when selection uses workspace linear in the merged input size. The block buffer and selection workspace are reused across iterations, so their representations does not accumulate with the number of blocks. Consequently, peak space is

$$S_{\text{blocked}} = \mathcal{O}(NB) + \mathcal{O}(NK) = \mathcal{O}(NB + NK). \tag{13}$$

Although all node pairs are evaluated, their distances are never stored simultaneously.

Sparse-aligned heatmap representation. The candidate-index array $\mathbf{C}$ and score array $\mathbf{S}$ each contain NK entries, requiring $\mathcal{O}(NK)$ representations. Each node additionally stores at most R rescue indices, contributing $\mathcal{O}(NR)$ representations. Any per-node list lengths require only $\mathcal{O}(N)$ additional space. For $K \geq 1$, the total representation size is therefore

$$S_{\text{representation}} = \mathcal{O}(NK) + \mathcal{O}(NR) + \mathcal{O}(N) = \mathcal{O}(N(K + R)). \tag{14}$$

Rescue candidates require no predicted-score representations because they do not participate in model computation.

Construction with rescue candidates. When rescue lists are built during graph construction, each node retains up to $K + R$ geometric neighbors. Replacing K by $K + R$ in the construction bounds gives $\mathcal{O}(N + N(K + R))$ space for angular-order sparsification and $\mathcal{O}(NB + N(K + R))$ peak space for blocked KNN. For fixed K, R, and B, both construction space and the final representation size grow linearly with N.

F THE USE OF LARGE LANGUAGE MODELS (LLMS)

We employed large language models strictly as copy-editing assistants to improve clarity, readability, and style across the manuscript. The models were used for tasks such as rephrasing sentences, checking grammar, harmonizing terminology, and smoothing the flow of prose. They did not generate technical content, analyses, results, or conclusions, and they did not modify the substance of our claims. All AI-assisted edits were reviewed and approved by the authors, and no confidential data were provided to the models.