Supplementary Material for *TopGANN*

1 Overview

This document provides additional material complementing the main paper, including experimental details, ablations, and extended results.

2 Complexity Analysis of Augmented SWAB (SWAB+)

This section complements the theoretical analysis of Step-Wise Adaptive Beam Width (SWAB) presented in §3.2 of the main paper. We briefly recall the key complexity result for SWAB and show that the augmented variant, SWAB+, preserves the same asymptotic indexing cost.

SWAB complexity. In the main paper, we showed that incremental graph construction with a fixed beam width $L = L_{\max}$ has total cost

$$T_{\text{fix}} = O(k\, L_{\max}\, N \log N)\,,$$

while SWAB, which partitions the construction into λ stages with beam widths $L_i = \frac{j+1}{\lambda} L_{\max}$ for stage j, yields

$$T_{\lambda} = O\left(k\, L_{\max}\, N \log N \cdot \frac{\lambda+1}{2\lambda}\right).$$

Consequently, SWAB reduces construction complexity by a factor approaching $2\times$ as λ increases (Theorem 3.3).

Table 2 summarizes the theoretical construction-cost reduction achieved by SWAB relative to fixed beam-width indexing. Each row corresponds to a choice of the step parameter λ, which controls the number of stages over which the beam width is increased during construction. The ratio $T_\lambda/T_{\text{fix}} = (\lambda+1)/(2\lambda)$ is given by Theorem 3.2 and quantifies the total indexing cost under SWAB normalized by the fixed-width baseline. The reported reduction is the relative savings implied by this ratio. For example, $\lambda = 2$ yields a 25% reduction in indexing cost, while larger values of λ provide progressively finer-grained adaptation and increased savings. As λ grows, the ratio converges to 1/2, corresponding to an asymptotic $2\times$ reduction in construction complexity compared to uniform beam-width indexing. “‘

SWAB+ extends SWAB by augmenting candidate *retention* without increasing exploration. During the beam search of insertion i, only the active beam of size $L_i \leq L_{\max}$ is expanded, exactly as in SWAB. In addition, SWAB+ retains up to $L_{\max} - L_i$ extra best-so-far candidates encountered during the same search, which are later considered during neighborhood pruning. Crucially, SWAB+ does *not* increase the beam width used for expansion and does not trigger additional graph traversals or distance evaluations beyond those already performed by SWAB.

λ	T_λ/T_{fix}	Reduction
2	3/4	25%
4	5/8	37.5%
10	11/20	45%

Table 1: Theoretical construction-cost reduction of SWAB relative to fixed beam width

.

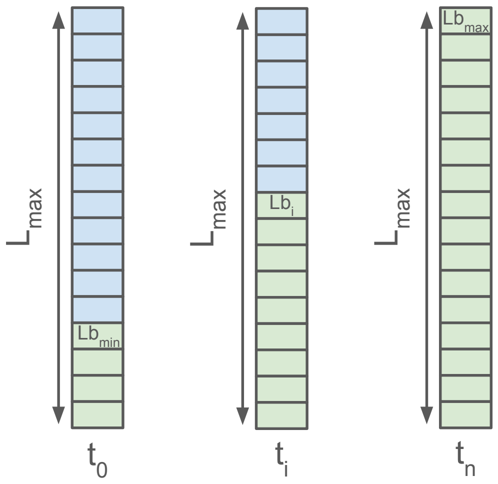


Figure 1: Illustration of SWAB+. The exploration beam is limited to L_i (green), while additional best-so-far candidates (blues) encountered during the same search are retained up to $L_{\max}$ for post-search pruning.

Complexity of SWAB+. Let V_i denote the number of visited nodes (distance evaluations) during the insertion of the i-th point. As in the SWAB analysis, beam search with width L_i on a small-world graph visits

$$V_i = O(L_i D_i) = O(L_i \log i)$$

nodes, where $D_i = O(\log i)$ is the effective search depth. SWAB+ performs the same set of expansions as SWAB and therefore evaluates the same number of distances.

The only additional cost introduced by SWAB+ is maintaining a capped candidate buffer of size at most $L_{\max}$. Updating this buffer for each visited node incurs at most $O(\log L_{\max})$ overhead, yielding a per-insertion cost

$$C_i^+ = O\big(L_i \log i \cdot (k + \log L_{\max})\big).$$

Since $L_{\max}$ is a fixed construction parameter, $\log L_{\max}$ contributes only a constant factor. Summing over all insertions gives

$$T_\lambda^+ = \sum_{i=1}^{N} C_i^+ = \Theta\left(\sum_{i=1}^{N} k\, L_i \log i\right) = \Theta(T_\lambda).$$

Implication. SWAB+ preserves the same asymptotic indexing complexity as SWAB and therefore inherits the same theoretical reduction factor relative to fixed-width construction:

$$\frac{T_\lambda^+}{T_{\text{fix}}} = \Theta\left(\frac{\lambda+1}{2\lambda}\right).$$

In practice, SWAB+ improves robustness by increasing the likelihood of retaining long-range or high-quality neighbors during early construction stages, while maintaining the same exploration cost and indexing-time behavior as SWAB.

3 Adaptive Beam Width under a Fixed Indexing-Cost Budget

This section studies how step-wise adaptive beam-width schedules can be leveraged to better utilize a fixed indexing-cost budget. We complement the complexity analysis of SWAB and SWAB+ with an experiment designed to isolate the effect of beam-width allocation under approximately matched construction cost, followed by a conditional theoretical explanation of the observed behavior.

3.1 Experimental Setup: Matched Indexing Cost

Let T denote a target indexing-time budget. In incremental graph (IG) construction, the dominant component of T is the cumulative cost of beam searches performed during node insertions, which scales linearly with the beam width used at each insertion.

To compare adaptive and non-adaptive strategies under comparable indexing cost, we proceed as follows. For a SWAB schedule with step parameter λ and maximum beam width $L_{\max}$, we define the average beam width

$$\bar{L} = \frac{1}{N}\sum_{i=1}^{N} L_i = \frac{L_{\max}(\lambda+1)}{2\lambda},$$

which follows directly from the definition of the step-wise schedule. We then construct a baseline IG index using a constant beam width $L = \bar{L}$, yielding approximately matched total indexing cost across methods.

In all experiments, $L_{\max}$ is fixed to a small value ($L_{\max} = 128$ for both LAION10M and T2I10M), deliberately stressing construction under a constrained computational budget.

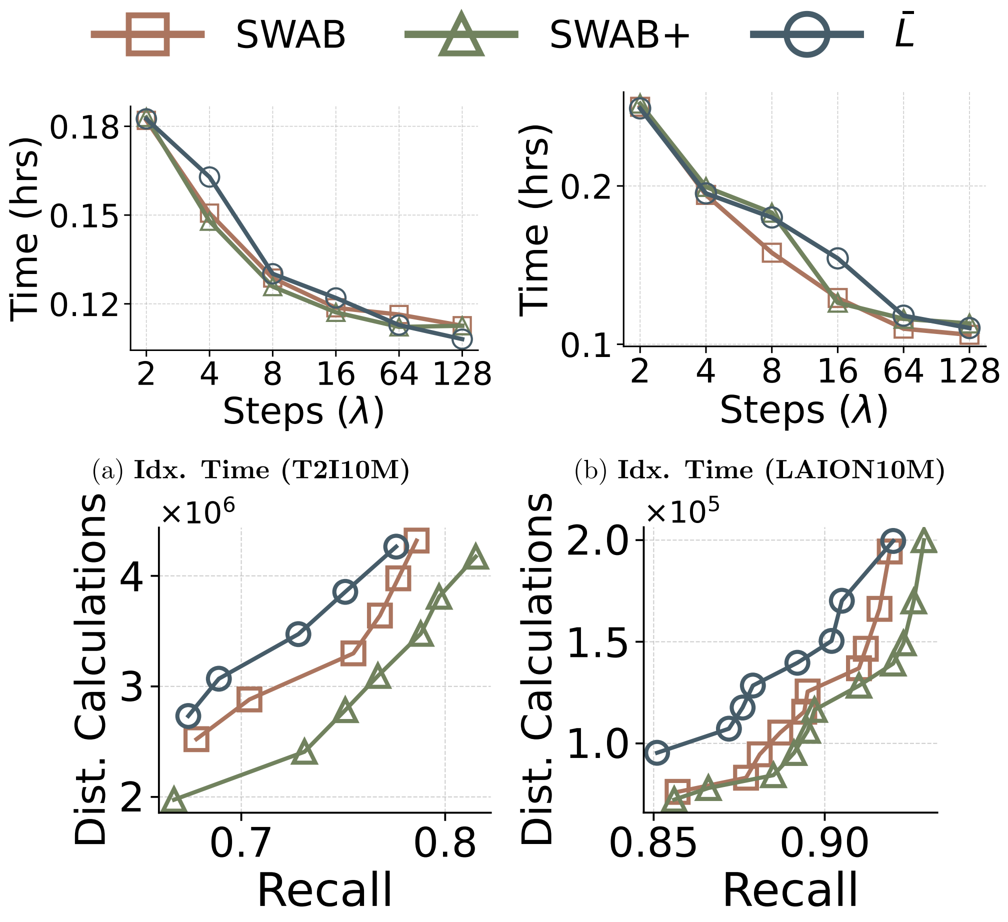
3.2 Empirical Results

Figure 2 reports indexing time and search performance under this matched-cost setting.

On T2I10M (Fig. 2a), the uniform-beam baseline incurs slightly higher indexing time (up to 20%), whereas on LAION10M (Fig. 2b) the indexing times are nearly identical. This difference is consistent with dataset hardness. On harder datasets, using a uniform beam width introduces many long-range edges during early insertions, when the graph is still sparse. These early long edges increase synchronization and update overhead, leading to higher effective construction cost.

Despite comparable indexing time, adaptive strategies consistently achieve better search performance. As shown in Figs. 2c and 2d, SWAB outperforms the uniform-beam baseline under the same average cost. Augmenting candidate retention (SWAB+) further improves search efficiency, achieving up to 1.38× and 1.22× gains compared to the uniform-beam baseline and standard SWAB, respectively.

These results indicate that, under a fixed indexing budget, the temporal allocation of computation matters. A uniform beam width allocates the same computational effort to all insertions, regardless of the maturity of the partial graph. In contrast, SWAB allocates smaller beams early, when candidate quality is limited by graph sparsity, and progressively larger beams later, when the graph has densified and additional exploration is more likely to yield useful neighbors. SWAB+ further mitigates early-stage losses by retaining additional high-quality candidates encountered during search, without increasing exploration cost.



(a) **Idx. Time (T2I10M)** (b) **Idx. Time (LAION10M)**

(c) **Search Performance (T2I10M)** (d) **Search Performance (LAION10M)**

Figure 2: Step-wise Adaptive vs. Constant Average Beam Width under constrained $L_{\max}$.

3.3 Theoretical Perspective (Conditional Guarantee)

We now provide a conservative theoretical explanation for the observed behavior. A fully unconditional proof is not possible without specifying a generative model linking beam width, candidate discovery, and final graph quality. Instead, we state a conditional result under explicit assumptions.

Setup. Let $Q_i(L)$ denote a scalar proxy of neighborhood quality obtained when inserting node x_i using beam width L. This quantity may represent, for example, the expected recall@k of the selected neighbor set or the expected reduction in distance to true nearest neighbors. We consider construction schedules $\{L_i\}_{i=1}^{N}$ with fixed total budget

$$\sum_{i=1}^{N} L_i = N\bar{L}.$$

The uniform-beam baseline corresponds to $L_i \equiv \bar{L}$, while SWAB corresponds to a non-decreasing schedule $L_1 \leq \cdots \leq L_N$ (up to discretization).

Assumption A1 (Diminishing returns in beam width). For each insertion index i, the function $Q_i(L)$ is discrete concave in L, i.e., the marginal gain

$$\Delta_i(L) \triangleq Q_i(L+1) - Q_i(L)$$

is non-increasing in L.

Assumption A2 (Increasing marginal utility over time). For any fixed $L \geq 1$ and any $1 \leq i < j \leq N$,

$$\Delta_i(L) \leq \Delta_j(L).$$

This assumption captures the idea that increasing the beam width becomes more beneficial as the partial graph densifies.

Budget reallocation under increasing returns:

Under Assumptions A1–A2, consider any construction schedule $\{L_i\}$ with fixed budget $\sum_i L_i = N\bar{L}$. Let $i < j$ with $L_i \geq 2$ and define a new schedule by transferring one unit of beam budget: $L'_i = L_i - 1$, $L'_j = L_j + 1$, and $L'_\ell = L_\ell$ for $\ell \notin \{i, j\}$. Then

$$\sum_{\ell=1}^{N} Q_\ell(L'_\ell) \;\geq\; \sum_{\ell=1}^{N} Q_\ell(L_\ell).$$

Consequently, reallocating beam budget from earlier to later insertions cannot decrease the total expected neighborhood quality.

Proof. Only the terms corresponding to indices i and j are affected:

$$\sum_\ell Q_\ell(L'_\ell) - \sum_\ell Q_\ell(L_\ell) = \big(Q_i(L_i - 1) - Q_i(L_i)\big) + \big(Q_j(L_j + 1) - Q_j(L_j)\big) = -\Delta_i(L_i - 1) + \Delta_j(L_j).$$

By Assumption A1, marginal gains are non-increasing in L, hence

$$\Delta_j(L_j) \geq \Delta_j(\min\{L_j, L_i - 1\}) \quad \text{and} \quad \Delta_i(L_i - 1) \leq \Delta_i(\min\{L_j, L_i - 1\}).$$

Applying Assumption A2 at the common beam level $\min\{L_j, L_i - 1\}$ yields

$$\Delta_j(\min\{L_j, L_i - 1\}) \geq \Delta_i(\min\{L_j, L_i - 1\}),$$

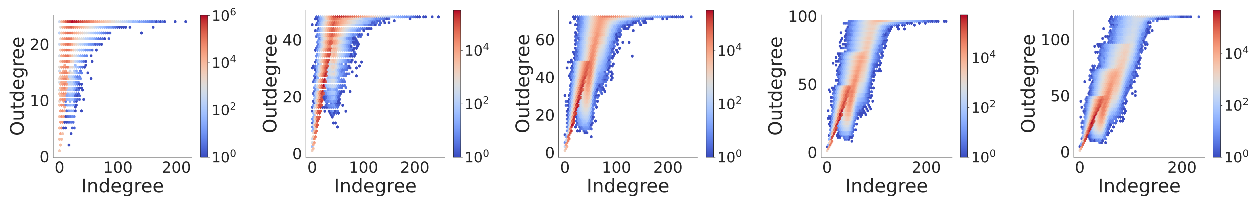
which implies $-\Delta_i(L_i - 1) + \Delta_j(L_j) \geq 0$. Therefore, the beam-budget transfer cannot decrease the total expected neighborhood quality. $\square$

Proposition above does not claim optimality of SWAB, nor does it assert that Assumptions A1–A2 hold universally. Rather, it formalizes sufficient conditions under which adaptive beam-width schedules can dominate uniform ones under the same average indexing cost. These conditions are consistent with the empirical behavior observed in Fig. 2.

Finally, SWAB+ can be interpreted as further reducing the loss incurred by small early beams by retaining additional best-so-far candidates during search, while preserving the exploration cost governed by $\{L_i\}$ (see §2).

4 Impact of Step-wise Adaptive Out-Degree (SAO)

We analyze the structural effects induced by Step-wise Adaptive Out-Degree (SAO) on the base layer of the graph constructed on DEEP10M. We fix $\Delta_R = 24$, $\alpha_{\max} = 1.2$, and use a single step factor κ controlling both the out-degree cap growth and pruning relaxation, i.e., $\kappa = \kappa_R = \kappa_\alpha$. We vary $\kappa \in \{1, 2, 3, 4, 5\}$ while keeping all other construction parameters unchanged.



(a) $\kappa = 1$ (b) $\kappa = 2$ (c) $\kappa = 3$ (d) $\kappa = 4$ (e) $\kappa = 5$

Figure 3: Joint indegree–outdegree distributions for increasing κ.

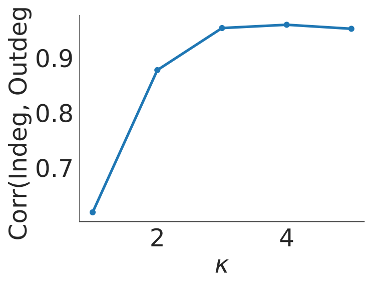


Figure 4: Correlation between indegree and outdegree as a function of κ.

5 Impact of Step-wise Adaptive Out-Degree (SAO)

Experimental setting. We study the structural effects of SAO on the base layer built on Deep10M. We fix $\Delta_R = 24$, $\alpha_{\max} = 1.2$, and use a single step factor $\kappa = \kappa_R = \kappa_\alpha$. We vary $\kappa \in \{1, 2, 3, 4, 5\}$ and keep all other parameters unchanged.

5.1 Indegree vs. Outdegree: Selective Reinforcement

For $\kappa = 1$, outdegree is effectively capped and largely independent of indegree, indicating a static regime. As κ increases, a clear positive dependency emerges: nodes with larger indegree progressively acquire higher outdegree, while low-indegree nodes remain near the baseline cap. This behavior reflects *selective reinforcement*. SAO does not densify the graph uniformly; instead, it allocates additional outgoing capacity to nodes that attract sustained incoming edges during construction. The effect strengthens smoothly with κ, without abrupt regime changes.

Figure 4 quantifies the trend observed in Figure 3. The correlation increases monotonically with κ, confirming that SAO progressively aligns outgoing capacity with incoming traffic. This monotonic behavior indicates stable adaptation rather than overfitting to a small subset of nodes. The correlation remains bounded, consistent with controlled degree growth.

Figure 5 shows that SAO induces bounded degree growth. Across all values of κ, the median outdegree increases only marginally and the interquartile range remains narrow. Degree expansion is driven by a limited upper tail, corresponding to the same high-indegree nodes identified earlier. Because the bulk of nodes retain near-baseline degree, the total number of edges grows slowly. This directly explains why SAO does not incur significant indexing-time or memory overhead, despite improving navigability through targeted reinforcement.

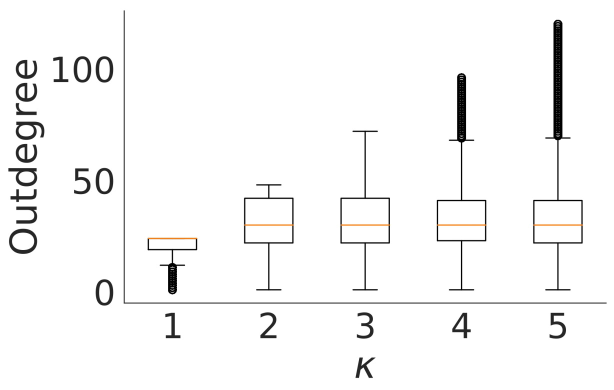


Figure 5: Outdegree boxplots for increasing κ. The median grows slowly and the upper tail expands moderately.

6 Insertion Order: Does it Matter?

Graph-based ANN methods typically construct the index by processing nodes in a random order, either through incremental insertion or by pruning an existing base graph. To the best of our knowledge, prior work rarely discusses the impact of node processing order, and most methods implicitly assume random processing.

However, the order of processing in incremental insertion construction can significantly affect both indexing and search performance. Consider a set of mutually similar nodes that are inserted concurrently: since the graph is constructed incrementally, the partial graph at time t does not yet contain these nodes, preventing them from discovering one another as neighbors during candidate retrieval. This may degrade connectivity and recall in the final graph structure.

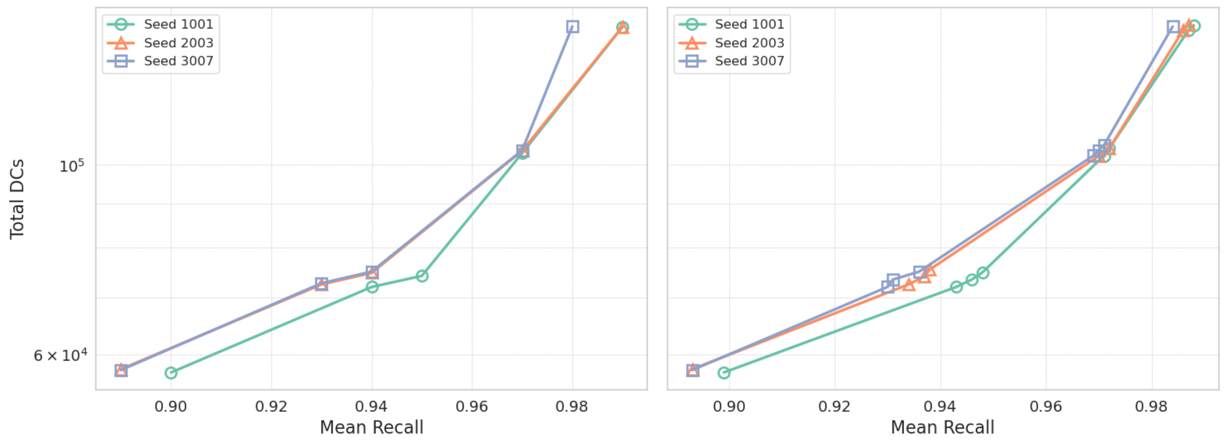


Figure 6: Search performance under random insertion order across different random seeds.

6.1 Formalization of Insertion Orders

We investigate the effect of insertion order in incremental graph construction by defining three canonical strategies for a dataset $\mathcal{D} = \{v_1, v_2, \ldots, v_n\}$ of d-dimensional vectors:

Random Order ($\mathcal{O}_R$). Nodes are permuted such that the expected distance between consecutive vectors v_i, v_{i+1} approximates the expected pairwise distance between uniformly sampled points in $\mathcal{D}$:

$$\mathbb{E}[d(v_i, v_{i+1})] \approx \mathbb{E}_{u,w\sim\mathcal{D}}[d(u, w)]. \tag{1}$$

Homogeneous Order ($\mathcal{O}_H$). Consecutive nodes are highly similar. Formally, the ordering minimizes the total distance between successive vectors:

$$\mathcal{O}_H = \arg\min_{\pi} \sum_{i=1}^{n-1} d(v_{\pi(i)}, v_{\pi(i+1)}), \tag{2}$$

where π is a permutation of $\{1, 2, \ldots, n\}$.

Heterogeneous Order ($\mathcal{O}_T$). Consecutive nodes are dissimilar, maximizing the total distance between successive vectors:

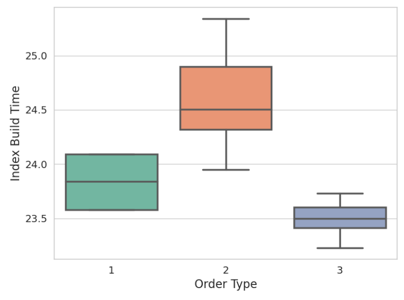
$$\mathcal{O}_T = \arg\max_{\pi} \sum_{i=1}^{n-1} d(v_{\pi(i)}, v_{\pi(i+1)}). \tag{3}$$

Both $\mathcal{O}_H$ and $\mathcal{O}_T$ correspond to NP-hard problems [?], equivalent to the shortest and longest Hamiltonian path variants of the Traveling Salesman Problem, respectively.

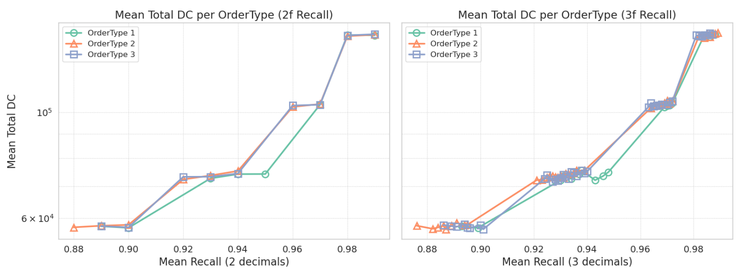
6.2 Approximation Strategy

Given the computational intractability of exact solutions, we approximate the orderings as follows:

1. Apply k-means clustering to partition $\mathcal{D}$ into k clusters $\{C_1, \ldots, C_k\}$ with centroids $\{\mu_1, \ldots, \mu_k\}$.
2. For homogeneous order: sort clusters by centroid norm $\|\mu_i\|_2$, then sort vectors within each cluster by distance to centroid.
3. For heterogeneous order with p parallel threads: partition the homogeneous sequence into p contiguous blocks and interleave them, ensuring that concurrently inserted vectors are drawn from different clusters and thus less likely to be similar.



(a) Indexing performance.



(b) Search performance.

As shown in Figure 7a, homogeneous insertion ($\mathcal{O}_H$) leads to longer indexing times, primarily due to synchronization overhead: similar nodes tend to share neighbors, and their concurrent insertion increases contention on neighborhood lists. By contrast, heterogeneous insertion ($\mathcal{O}_T$) reduces synchronization, since concurrently inserted nodes are unlikely to overlap in neighborhood sets. This difference becomes more pronounced in higher dimensions, where distance computations are costlier and locks are held for longer durations.

In query evaluation (Figure 7b), random order ($\mathcal{O}_R$) generally achieves the best recall-performance tradeoff, while homogeneous order shows slightly reduced recall, particularly below 0.9. This reduction arises because promising edges between similar neighbors may be missed when such nodes are inserted concurrently.

These results should be interpreted cautiously. Both insertion order and thread scheduling affect performance. In our experiments, OpenMP threads were scheduled dynamically, which may conflict with the imposed ordering. A more controlled scheduling policy, aligned with the insertion strategy, or single-threaded indexing is required to validate and confirm these preliminary findings.

6.3 Parameter Settings

We report the main hyper-parameters used by each index. Unless stated otherwise, we tune these parameters to achieve target recall levels and report the corresponding efficiency metrics.

HNSW. We use the standard HNSW parameters: (i) M, the maximum number of neighbors selected for a newly inserted node at each layer, and (ii) L (often denoted efConstruction), the construction beam width used for candidate neighbor search during insertion.

NSG. We use the standard NSG parameters: (i) R, the maximum out-degree of the final NSG graph, (ii) C, the candidate pool size used during neighbor selection/pruning, and (iii) L, the construction/search list size controlling candidate exploration during building NSG is constructed on top of an initial graph produced by EFANNA; therefore, we report the EFANNA parameters that govern the graph initialization and refinement. Specifically, EFANNA is configured by: (i) `mlevel`, the maximum graph level, (ii) `trees`, the number of random projection trees, (iii) `iterations`, the number of refinement iterations, (iv) L, the beam width used during search, (v) R, the maximum number of connections per node, (vi) K, the number of nearest neighbors considered during refinement, and (vii) S, the candidate pool size used in refinement.

Vamana. We control Vamana using: (i) R, the out-degree bound of the graph, (ii) C, the candidate set size used in pruning/neighbor selection, (iii) L, the construction/search list size controlling exploration, and (iv) α, the pruning relaxation parameter (often used in robust/greedy pruning rules) that determines how aggressively candidates are discarded.

HCNNG. HCNNG is configured by: (i) the number of clusters, (ii) the cluster size (or an equivalent capacity parameter), and (iii) `mstout`, which controls the number of outgoing edges in the MST.

TopGANN. TopGANN is configured by: (i) Δ_R, the step increment controlling how the allowable out-degree budget grows, (ii) $L_{\max}$, the maximum beam width (upper bound on candidate exploration), (iii) λ, the number of stages in the step-wise beam schedule (SWAB), (iv) κ_R, the multiplicative step factor controlling step-wise out-degree relaxation (SAO), (v) κ_α, the multiplicative step factor controlling step-wise pruning relaxation, and (vi) $\alpha_{\max}$, the maximum pruning relaxation reached by the adaptive schedule.

6.4 Method-specific Parameter Settings

Dataset	**Size**	$\boldsymbol{M}$	$\boldsymbol{L}$
Deep	1M	30	300
Deep	10M	32	400
Deep	100M	32	400
T2I	1M	38	500
T2I	10M	38	600
T2I	100M	48	700
LAION	1M	32	300
LAION	10M	32	400
LAION	100M	32	400
SpaceV	1M	32	400
SpaceV	10M	40	500
SpaceV	100M	40	500

Table 2: HNSW parameter settings per dataset and scale.

Dataset	**Size**	$\boldsymbol{\Delta_R}$	$\boldsymbol{L_{\max}}$	$\boldsymbol{\lambda}$	$\boldsymbol{\kappa_R}$	$\boldsymbol{\kappa_\alpha}$	$\boldsymbol{\alpha_{\max}}$
Deep	1M	30	300	4	2	10	1.2
Deep	10M	32	400	4	3	10	1.2
Deep	100M	32	400	3	3	10	1.2
T2I	1M	38	500	4	3	5	1.2
T2I	10M	38	700	4	3	3	1.2
T2I	100M	48	700	4	3	4	1.6
LAION	1M	32	300	4	3	5	1.2
LAION	10M	32	400	4	3	5	1.2
LAION	100M	32	400	4	3	5	1.2
SpaceV	1M	38	400	4	2	6	1.2
SpaceV	10M	40	500	4	2	4	1.1
SpaceV	100M	40	500	4	3	4	1.1

Table 3: TopGANN parameter settings per dataset and scale.

Dataset	**Size**	$\boldsymbol{R}$	$\boldsymbol{C}$	$\boldsymbol{L}$	$\boldsymbol{\alpha}$
Deep	1M	60	300	300	1.15
Deep	10M	64	400	400	1.15
Deep	100M	64	400	400	1.15
T2I	1M	68	400	500	1.2
T2I	10M	68	400	600	1.2
T2I	100M	74	400	700	1.2
LAION	1M	60	300	300	1.15
LAION	10M	60	300	300	1.15
LAION	100M	60	300	300	1.15
SpaceV	1M	64	400	400	1.18
SpaceV	10M	64	400	500	1.18
SpaceV	100M	64	400	500	1.18

Table 4: Vamana parameter settings per dataset and scale.

Dataset	**Size**	**mlevel**	**trees**	**iterations**	***K***	***S***	***R***	***L***	***C***
Deep	1M	8	8	8	60	20	60	300	300
Deep	10M	8	8	8	60	20	62	400	400
Deep	100M	8	8	8	60	20	62	400	400
T2I	1M	8	8	8	60	20	72	500	500
T2I	10M	8	8	8	60	20	72	600	600
T2I	100M	8	8	8	60	20	74	700	700
Laion	1M	8	8	8	60	20	60	300	300
Laion	10M	8	8	8	60	20	62	400	400
Laion	100M	8	8	8	60	20	62	400	400
Spacev	1M	8	8	8	60	20	64	400	400
Spacev	10M	8	8	8	60	20	66	500	500
Spacev	100M	8	8	8	60	20	66	500	500

Table 5: NSG parameter settings, reported via + EFANNA parameters used for graph initialization and refinement, per dataset and scale.

Dataset	**Size**	**#clusters**	**Cluster size**	**mstout**
Deep	1M	30	1000	3
Deep	10M	30	1000	3
Deep	100M	30	1000	3
T2I	1M	38	1000	3
T2I	10M	38	1000	3
T2I	100M	38	1000	3
LAION	1M	32	1000	3
LAION	10M	32	1000	3
LAION	100M	32	1000	3
SpaceV	1M	32	1000	3
SpaceV	10M	32	1000	3
SpaceV	100M	32	1000	3

Table 6: HCNNG parameter settings per dataset and scale.