DAPG: DISTANCE-AWARE PRUNING FOR DYNAMIC GRAPH-BASED APPROXIMATE NEAREST NEIGHBOR SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Approximate nearest neighbor (ANN) search over high-dimensional vector data is a core component of vector databases, recommendation systems, and retrieval-augmented generation. Graph-based indexes provide strong search efficiency, but most are designed for largely static datasets. This implies that under frequent insertions and deletions: (i) graph connectivity may degrade, (ii) maintenance cost can increase, and (iii) redundant edges can further increase query latency and memory usage. To address these challenges, we propose the Distance-Aware Pruned Graph (DAPG), a dynamic graph-based ANN index designed to maintain high-recall search while supporting efficient updates. DAPG constructs an LSH-seeded single-layer proximity graph and applies a two-stage pruning strategy that balances sparsity and navigability. First, a node-specific percentile threshold adapts edge retention to the local distance distribution. Second, global sparsification constrains high-degree nodes and removes redundant connectivity using an adaptive or controlled degree budget. For dynamic workloads, DAPG performs localized maintenance by reapplying pruning only to affected hash buckets, candidate neighborhoods, and adjacency lists, thereby avoiding full index reconstruction. We implement DAPG and evaluate it on seven datasets against eleven representative ANN baselines, including state-of-the-art dynamic methods. In static search, DAPG improves the recall-latency trade-off, reducing query latency by up to 65.6%. For dynamic workloads, DAPG reduces maintenance time by up to $715\times$ relative to rebuild-based baselines. On MS-MARCO1M, specifically, DAPG achieves average Recall@100 of 0.9680 compared to 0.9485 for Wolverine++, while providing $11.0\times$ higher deletion throughput.

1 INTRODUCTION

High-dimensional vector search is a core operation in modern data systems, supporting information retrieval, recommendation, multimedia search, and retrieval-augmented generation (Monir et al., 2024; Huang et al., 2019; Monir et al., 2025; Lewis et al., 2020). Recent large language model (LLM)-based search systems further emphasize the need for high-recall, low-latency ANN retrieval, since both excessive retrieval cost and insufficient recall can degrade end-to-end system efficiency (Yang et al., 2026). Approximate similarity search has been widely studied using tree-based, hashing-based, quantization-based, and graph-based indexes (Qin et al., 2020; Li et al., 2019; Gionis et al., 1999; Jégou et al., 2011a; Wang et al., 2021; Aumüller et al., 2020). Graph-based ANN indexes provide high recall and efficient search, while recent geometry-aware methods exploit local intrinsic structure to improve retrieval efficiency (Zhao, 2026). However, frequent insertions and deletions remain challenging (Xu et al., 2022; 2023; Liu et al., 2025).

Dynamic vector search systems must support frequent insertions and deletions while preserving high-recall search and low query latency. In practice, indexed collections continuously evolve as new documents, products, or media items arrive and obsolete entries are removed. Graph-based ANN indexes, although highly effective in static settings (Fu et al., 2019; Pan et al., 2023), can incur substantial update cost because insertions and deletions may disrupt navigational paths and require graph repair (Xu et al., 2022; 2023; Chung et al., 2025). This motivates dynamic ANN indexes that preserve traversal quality while controlling both query and maintenance cost.

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.

Dynamic maintenance remains challenging for graph-based and hybrid ANN indexes because insertions and deletions can degrade search quality, navigability, and graph structure while increasing update cost (Xu et al., 2022; Singh et al., 2021; Xu et al., 2023; Liu et al., 2025). FreshDiskANN (Singh et al., 2021) supports streaming updates through dynamic graph maintenance, SPFresh (Xu et al., 2023) performs localized in-place partition rebalancing, and Wolverine (Liu et al., 2025) repairs monotonic search paths disrupted by deletions. However, these approaches do not directly address how edge selection should adapt to heterogeneous local density during dynamic graph maintenance.

We propose the distance-aware pruned graph (DAPG), a single-layer, geometry-aware ANN index for accurate search and efficient maintenance under dynamic workloads. DAPG introduces a density-aware pruning methodology for dynamic proximity graphs, with LSH used as one concrete candidate-generation backend. Existing graph-pruning methods commonly rely on fixed degree constraints or geometric heuristics that do not explicitly adapt edge selection to the empirical distance distribution of each node's neighborhood. In contrast to NSG's MRNG edge-selection rule (Fu et al., 2019), which evaluates pairwise distance relationships among candidate neighbors, DAPG's pruning rule is node-wise and distributional, with τ_i estimated from each vertex's local candidate-distance distribution. DAPG instead applies a two-stage pruning process. First, it computes a node-specific percentile threshold from local neighbor distances and removes distant candidates relative to each node's neighborhood distribution. Second, it applies an adaptive degree cap T' to control residual high-degree nodes and bound graph sparsity. Together, these stages reduce redundant connectivity while retaining useful navigational links and enabling localized insert/delete maintenance without full-index reconstruction. Detailed construction and pruning procedures are provided in Appendix C.1, while dynamic maintenance is described in Section 4.2.

In contrast to hierarchical graph organization (Malkov & Yashunin, 2020; Jiang et al., 2025) and uniform degree-based pruning (Zhao et al., 2023; Fu et al., 2019), DAPG separates local geometry-aware edge selection from global sparsity control. This allows edge selection to adapt to heterogeneous neighborhood structure before global degree control is applied. We establish degree and sparsity bounds, characterize traversal work under best-first search, and derive a conditional recall-preservation bound under candidate coverage, local edge preservation, and post-pruning reachability assumptions.

This paper makes the following technical contributions:

- We propose DAPG, a geometry-aware proximity graph index for ANN search over dynamic data. DAPG integrates node-local percentile pruning with adaptive global sparsification in a candidate-seeded graph construction process, allowing edge selection to respond to local distance distributions rather than relying solely on fixed-degree constraints. The resulting single-layer graph reduces redundant connectivity while supporting efficient search and localized dynamic maintenance (Section 3).
- We design LSH-seeded graph traversal and localized insert/delete maintenance for DAPG. The resulting algorithms support efficient query processing and dynamic updates without full-index reconstruction. We further analyze graph sparsity, construction and query costs, and conditional recall preservation (Section 4).
- We evaluate DAPG across seven datasets and eleven representative ANN baselines. Results show improved recall-latency trade-offs, competitive build time and index size, high post-update recall, and up to $11.0\times$ higher deletion throughput than a state-of-the-art dynamic ANN method (Section 5).

2 PRELIMINARIES

We introduce the problem setting and notation. Given $(\mathcal{O}, \delta, k)$ and a candidate-generation backend, our goal is to construct a single-layer graph index $G' = (V, E'')$ that supports high Recall@k at low latency, bounded graph sparsity, and insertions and deletions without full reconstruction. We assume that the backend provides candidate coverage ρ_{cand}, that $\delta(\cdot, \cdot)$ is a metric, and that per-node candidate-distance percentiles are well defined. In this work, candidate generation is instantiated with LSH. For node v_i, τ_i denotes its local percentile threshold, $p \in [0, 1]$ the pruning percentile,

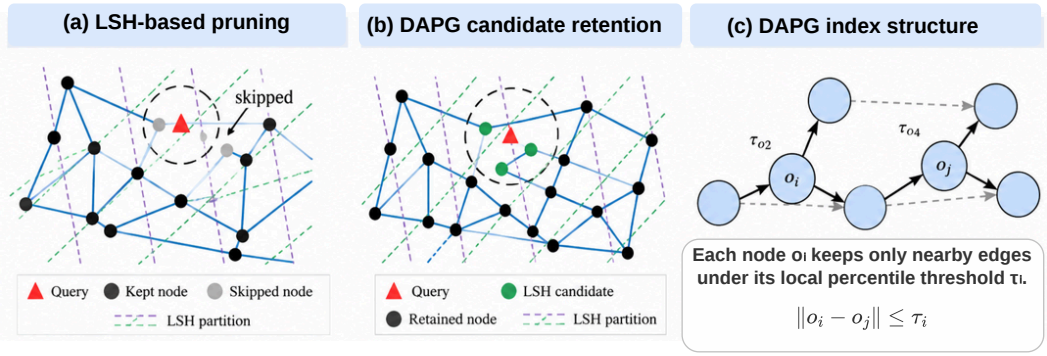


Figure 1: DAPG overview. (a) LSH-based candidate retrieval for the query, where some candidates are skipped. (b) Distance-aware pruning retains locally relevant candidates to form a sparse proximity graph. (c) The resulting single-layer DAPG index applies node-specific thresholds τ_i to support efficient search and localized dynamic updates.

and T' the adaptive global degree cap. The pruning operators $\mathcal{P}_{\text{local}}$ and $\mathcal{P}_{\text{global}}$, together with the conditional quantities ρ_{loc} and ρ_{reach}, are defined later in Proposition 3.

Let $\mathcal{O} = \{o_1, \ldots, o_n\} \subset \mathbb{R}^d$ be a set of n d-dimensional vectors, $q \in \mathbb{R}^d$ a query, and $\delta(\cdot, \cdot)$ the search metric. For $k \geq 1$, let $N_k(q)$ denote the k nearest neighbors of q in $\mathcal{O}$. We represent the index as a directed proximity graph $G = (V, E)$ with $V \equiv \mathcal{O}$, where vertex $v_i \in V$ corresponds to data object $o_i \in \mathcal{O}$. For $v \in V$, $N(v)$ denotes its out-neighborhood, $\deg(v) = |N(v)|$, and $\bar{d} = |V|^{-1} \sum_{v \in V} \deg(v)$ denotes the average out-degree.

LSH (Indyk & Motwani, 1998; Andoni & Indyk, 2008; Datar et al., 2004) is used as the candidate-generation backend, with ρ_{cand} denoting candidate coverage. Additional notation and background are provided in Appendix B.1 and Appendix B.2.

3 Distance-Aware Pruned Graph Index

Figure 1(a) illustrates LSH-seeded candidate generation, where relevant neighbors may fall outside the retrieved candidate region (Gionis et al., 1999; Zhao et al., 2023). Given the retrieved candidates, Figure 1(b) shows how DAPG applies distance-aware graph pruning to construct locally adaptive neighborhoods. For each node v_i, a node-specific percentile threshold τ_i is derived from its local candidate-distance distribution and used to retain locally relevant neighbors. This allows edge selection to adapt to local distance structure rather than relying solely on uniform degree constraints. After local pruning, DAPG applies adaptive global sparsification to control residual high-degree neighborhoods. The two stages provide complementary forms of adaptation: τ_i controls node-level edge retention, while the global degree cap controls graph-level sparsity. Together, they reduce redundant connectivity while retaining useful navigational links across heterogeneous regions.

Figure 1(c) summarizes the resulting DAPG architecture. Starting from LSH-seeded candidate neighborhoods, DAPG applies local distance-aware pruning followed by global sparsification to construct a sparse single-layer proximity graph. Query processing and dynamic updates operate directly on this pruned graph. Insertions and deletions are handled through localized maintenance of affected candidate neighborhoods and adjacency lists rather than full index reconstruction. The complete construction procedure is provided in Algorithm 2 in Appendix C.1. The local and global pruning mechanisms are described in Section 3.1 and Section 3.2, respectively.

3.1 Local Distance-Aware Pruning

The first stage of DAPG performs local distance-aware pruning to retain relevant neighbors while removing redundant edges. Given a graph $G = (V, E)$ and the neighborhood $N(v_i)$ of a node $v_i \in V$, let

$$\mathcal{D}(v_i) = \{\delta(v_i, v_j) \mid v_j \in N(v_i)\}$$

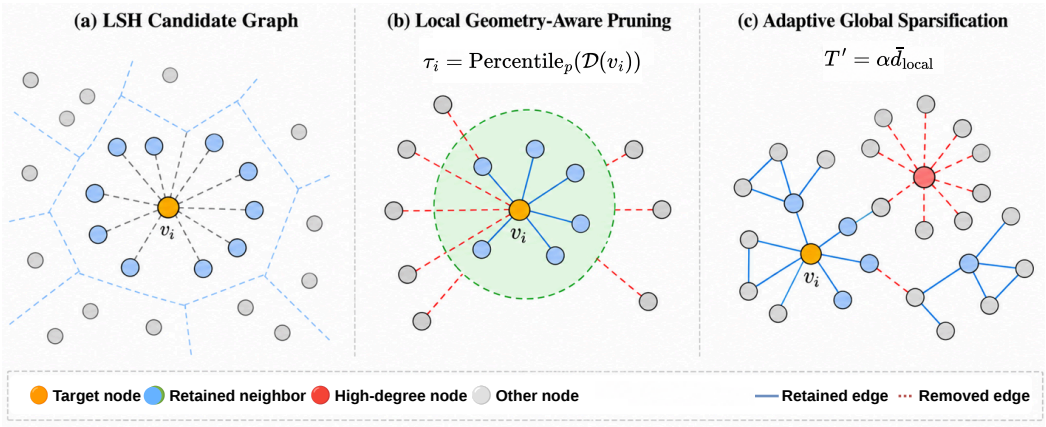


Figure 2: Construction of DAPG. (a) LSH provides a candidate neighborhood for node v_i. (b) Local distance-aware pruning retains edges according to the node-specific percentile threshold τ_i. (c) Adaptive global sparsification applies the degree cap $T' = \alpha \bar{d}_{\text{local}}$ to remove residual high-degree connections, producing the final sparse proximity graph.

denote the distribution of distances from v_i to its candidate neighbors. For each node, DAPG computes the percentile threshold

$$\tau_i = \text{Percentile}_p(\mathcal{D}(v_i)),$$

where $p \in [0, 1]$ controls the fraction of locally close neighbors retained. The pruned neighborhood is then

$$N'(v_i) = \{v_j \in N(v_i) \mid \delta(v_i, v_j) \leq \tau_i\}.$$

We denote this local pruning operator by $\mathcal{P}_{\text{local}}$, such that

$$N'(v_i) = \mathcal{P}_{\text{local}}(N(v_i)).$$

Unlike a fixed global degree rule, the threshold τ_i is derived from each node's local distance distribution. Dense neighborhoods tend to produce smaller thresholds, whereas sparse neighborhoods may retain more distant neighbors. Consequently,

$$|N'(v_i)| \approx p\,|N(v_i)|.$$

This node-specific adaptation reduces redundant edges in dense regions while retaining useful navigational links in sparse regions, thereby lowering the average post-pruning degree. Proposition 1 establishes the resulting sparsity bound, while Proposition 3 gives the corresponding conditional recall-preservation analysis.

3.2 Global Sparsification Strategy

Let $\bar{d}_{\text{local}}$ denote the average out-degree after local pruning. We set the adaptive global degree cap to

$$T' = \alpha\, \bar{d}_{\text{local}}, \tag{1}$$

where $\alpha \in (0, 1]$ controls the target sparsity level.

DAPG therefore incorporates two levels of adaptation. As described in Section 3.1, local pruning derives the node-specific percentile threshold τ_i from the distance distribution of the LSH-seeded candidate neighborhood to provide density-aware edge selection. Global sparsification then adapts the degree cap to the graph's post-pruning structure according to Equation (1).

For each node v_i, neighborhoods with more than T' retained neighbors are pruned according to edge distance:

$$N''(v_i) = \begin{cases} N'(v_i), & |N'(v_i)| \leq T', \\ \text{Top}_{T'}(N'(v_i)), & |N'(v_i)| > T', \end{cases} \tag{2}$$

where $\mathrm{Top}_{T'}$ retains the T' closest neighbors according to $\delta(v_i, \cdot)$. We denote this operation by

$$N''(v_i) = \mathcal{P}_{\mathrm{global}}(N'(v_i)) .$$

Thus, $|N''(v_i)| \leq T'$, which bounds the local branching factor and yields $\bar{d}_{\mathrm{DAPG}} \leq T'$. Proposition 1 establishes the resulting sparsity bound, while Proposition 2 characterizes its effect on traversal work.

As shown in Figure 2, DAPG applies local distance-aware pruning followed by global sparsification to produce the final sparse proximity graph.

The complete LocalPrune and GlobalPrune procedures are provided in Appendix C.1.

4 Query Processing and Dynamic Maintenance

4.1 LSH-Seeded Query Traversal

After constructing the pruned graph $G' = (V, E'')$, DAPG performs LSH-seeded best-first traversal. For a query q, the LSH backend forms a candidate set C_q and selects a seed set $S_q \subseteq C_q$ as traversal entry points. Search expands retained neighborhoods $N''(u)$ using a frontier ordered by $\delta(q, \cdot)$ and a bounded candidate set of size ef. The complete query procedure is provided in Appendix C.6.

To study entry-point selection, we evaluate four additional traversal configurations on the same pruned graph. DAPG-Pseudo uses multiple pseudo entry points, DAPG-Aware reranks LSH candidates using collision counts, DAPG-Anchor uses precomputed anchors, and DAPG-Learned uses a lightweight learned selector for query-specific anchors. Detailed definitions are provided in Appendix D.3.

4.2 Localized Insert/Delete Maintenance

To support evolving datasets, DAPG performs localized maintenance while preserving the local and global pruning constraints. For an inserted object o_{new}, the LSH backend first retrieves a candidate set

$$C_{\mathrm{new}} = \{\, v_i \in V \mid h(o_{\mathrm{new}}) = h(v_i) \text{ for some } h \in \mathcal{H} \,\}, \tag{3}$$

where $\mathcal{H}$ denotes the LSH hash family. An initial neighborhood $N(o_{\mathrm{new}})$ is formed from the top-k candidates with the smallest distances to o_{new}. DAPG then applies $\mathcal{P}_{\mathrm{local}}$ followed by $\mathcal{P}_{\mathrm{global}}$, so that the inserted node satisfies the density-aware local pruning rule and global degree constraint.

Algorithm 1: DAPG-Update (Insert / Delete)

Input: Mode $\in \{\textsc{Insert}, \textsc{Delete}\}$, pruning percentile p, degree cap T'
Output: Updated pruned graph G'

if *Mode* = $\textsc{Insert}(o_{\mathrm{new}})$ **then**
  Obtain candidate set C_{new} via LSH;
  Construct $N(o_{\mathrm{new}}) \leftarrow \text{Top-}k(C_{\mathrm{new}})$;
  Apply local pruning: $N'(o_{\mathrm{new}}) \leftarrow \mathcal{P}_{\mathrm{local}}(N(o_{\mathrm{new}}))$;
  Apply global pruning: $N''(o_{\mathrm{new}}) \leftarrow \mathcal{P}_{\mathrm{global}}(N'(o_{\mathrm{new}}))$;
  Add symmetric edges between o_{new} and $v_j \in N''(o_{\mathrm{new}})$;
  Re-apply $\mathcal{P}_{\mathrm{global}}$ to affected neighbors exceeding T';
else if *Mode* = $\textsc{Delete}(v_d)$ **then**
  Remove v_d and all incident edges from G';
  foreach *affected neighbor* u **do**
    Recompute $N'(u) \leftarrow \mathcal{P}_{\mathrm{local}}(N(u))$;
    Recompute $N''(u) \leftarrow \mathcal{P}_{\mathrm{global}}(N'(u))$;
return updated graph G';

For insertion, DAPG incrementally applies the local and global pruning operators to affected neighborhoods, avoiding full graph reconstruction. For deletion, the target vertex and its associated edges are removed before affected neighborhoods are locally re-pruned.

The amortized update cost is

$$\mathbb{E}[T_{\text{upd}}] = O\big(d\, C_Q + \bar{d}_{\text{DAPG}} \log n\big)\,, \tag{4}$$

where d is the embedding dimensionality and C_Q the expected number of candidates examined by the LSH-seeded retrieval.

Thus, Algorithm 1 performs localized maintenance by reapplying $\mathcal{P}_{\text{local}}$ and $\mathcal{P}_{\text{global}}$ only to affected neighborhoods, preserving the pruning and sparsity constraints without full-index reconstruction.

To study different trade-offs between update cost and post-update search quality, we consider three extensions to the base DAPG procedure. DAPG-Lazy marks deleted vertices inactive and defers edge repair. DAPG-Batch amortizes cleanup and re-pruning over an update block. DAPG+Hybrid augments DAPG with an auxiliary HNSW candidate layer and reranks merged candidates by exact distance to improve candidate coverage during updates.

4.3 Theoretical Analysis

We establish three theoretical properties of DAPG: (i) bounded graph sparsity under two-stage pruning, (ii) a decomposition of traversal work, and (iii) conditional recall preservation. Detailed proofs and construction-complexity analysis are provided in Appendix C.

Proposition 1 (Degree and Sparsity Bound). *Let $G' = (V, E'')$ be the graph obtained after applying $\mathcal{P}_{\text{global}} \circ \mathcal{P}_{\text{local}}$. For every $v_i \in V$,*

$$|N''(v_i)| \le \min\{|N'(v_i)|, T'\} \le T'. \tag{5}$$

Consequently, $\bar{d}_{\text{DAPG}} \le T'$ and $|E''| \le |V|T'$.

The bound guarantees sparsity but does not by itself guarantee graph connectivity. We therefore evaluate post-pruning reachability empirically.

Proposition 2 (Traversal-Work Decomposition). *Consider best-first search on $G' = (V, E'')$ initialized from LSH-generated seeds. Let $\bar{d}_{\text{DAPG}}$ denote the expected out-degree and $\beta(\ell)$ the expected number of expansions required to reach the relevant k-NN region under probe budget ℓ. Then*

$$C_Q^{\text{DAPG}} = \mathcal{O}\big(\bar{d}_{\text{DAPG}}\beta(\ell)\big)\,. \tag{6}$$

Equation (6) separates traversal work into branching factor and expansion count. Reducing $\bar{d}_{\text{DAPG}}$ lowers the work per expansion, while the effect on $\beta(\ell)$ is evaluated empirically.

Proposition 3 (Conditional Recall Preservation). *Let $N_k(q)$ denote the true top-k neighbors and $S(q, G')$ the result returned by best-first search. Define E_{cand} as candidate access to the relevant top-k region, E_{loc} as preservation of the required edges after local pruning, and E_{reach} as the existence of a traversal path from a search seed to the relevant k-NN region. Let*

$$\rho_{\text{cand}} = \Pr[E_{\text{cand}}], \qquad \rho_{\text{loc}} = \Pr[E_{\text{loc}} \mid E_{\text{cand}}], \qquad \rho_{\text{reach}} = \Pr[E_{\text{reach}} \mid E_{\text{cand}} \cap E_{\text{loc}}].$$

If $E_{\text{cand}} \cap E_{\text{loc}} \cap E_{\text{reach}}$ is sufficient to recover $N_k(q)$, then

$$\Pr[S(q, G') = N_k(q)] \ge \rho_{\text{cand}}\rho_{\text{loc}}\rho_{\text{reach}}. \tag{7}$$

Proposition 3 provides a conditional recovery guarantee. The effects of p, T', and candidate-neighborhood structure are evaluated empirically in Section 5. DAPG has build cost $\mathcal{O}(ndC_Q + nk\log k + n\bar{d}_{\text{DAPG}})$, query cost $\mathcal{O}(\bar{d}_{\text{DAPG}}\beta(\ell))$, and graph space $\mathcal{O}(n\bar{d}_{\text{DAPG}})$. A detailed comparison with representative ANN methods is provided in Appendix C.5.

5 Evaluations

This section evaluates DAPG across static search and update settings. We describe the experimental setup and answer three research questions (RQs):

- RQ1: How does DAPG's two-stage distance-aware pruning affect recall, query latency, and construction efficiency across pruning settings?
- RQ2: How does DAPG compare with state-of-the-art static ANN methods in recall-latency trade-off, build time, and index size, and how do its query-time variants affect matched-recall search efficiency?
- RQ3: How does DAPG compare with state-of-the-art dynamic ANN methods in post-update recall, search throughput, and update efficiency under insert/delete workloads?

5.1 Experimental Setup

Datasets. We evaluate DAPG on seven high-dimensional datasets spanning different scales and data modalities. Small-scale benchmarks include MNIST (Lecun et al., 1998) (784-D image vectors) and Audio (RSIA-LIESMARS-WHU, 2025) (192-D audio features). Medium-scale datasets include Deep1M (Babenko & Lempitsky, 2016) (96-D embeddings), SIFT1M (Jégou et al., 2011b) (128-D descriptors), and Text2Image1M (Simhadri et al., 2022) (200-D multimodal embeddings). We additionally include MS-MARCO1M (Bajaj et al., 2018) as a neural text-retrieval embedding workload in our dynamic evaluation. For large-scale evaluation, we use SIFT100M (Jégou et al., 2011b), which contains 100M 128-D vectors. Dynamic experiments are conducted across multiple datasets and update protocols, with detailed dataset statistics and experimental setting provided in Appendix D.1.

Baselines. We compare DAPG against eleven representative ANN baselines spanning LSH-based, graph-based, query-adaptive, and dynamic methods. LSH-based methods include DB-LSH (Tian et al., 2024) and LSH-APG (Zhao et al., 2023). Graph-based methods include HNSW (Malkov & Yashunin, 2020), HCNNG (Vargas Muñoz et al., 2019), NSG (Fu et al., 2019), FastNSG (Yang et al., 2025), FastHNSW (Yang et al., 2025), and RNNDescent (Ono & Matsui, 2023). We also include GATE (Ruan et al., 2025) as a query-adaptive baseline and Wolverine++ (Liu et al., 2025) and FreshDiskANN (Singh et al., 2021) as state-of-the-art dynamic ANN baselines.

Evaluation metrics. We measure search quality using Recall@k and efficiency using query latency (QT, ms/query), index size (MB), and build time (s). For matched-recall comparisons, Rt@0.95 denotes query latency at Recall@100 $= 0.95$, and QPS@R denotes query throughput at target recall R. For dynamic experiments, we report Recall@100 on the active set, Search OPS, Insert OPS, and Delete OPS. Maintenance time denotes per-window rebuilding for rebuild-based baselines and localized maintenance for DAPG. We additionally report recall-aware cost efficiency, $\mathrm{CE} = R \cdot \mathrm{QPS@R/Maintenance}$.

Parameter settings and dynamic evaluation. We use tuned configurations for DAPG and the compared baselines, following recommended settings from official implementations where available. The node-specific threshold τ_i provides density adaptation, while T' controls the final degree budget. We use $T' = \lceil \alpha \bar{d}_{\mathrm{local}} \rceil$ when adaptive and fix $T' = 48$ in the reported experiments for controlled sparsity. Dynamic evaluation includes FIFO sliding-window and random delete-and-new-insert workloads on SIFT1M, and a FIFO sliding-window workload on MS-MARCO1M. For SIFT1M, we maintain a $900K$ active set from $1M$ vectors and run five rounds, each deleting $10K$ active vectors and inserting $10K$ previously unseen vectors. For MS-MARCO1M, we initialize with $900K$ passages and use the remaining $100K$ as a 10-round update stream, replacing $10K$ passages per round. We report post-update recall, search throughput, and update throughput. Rebuild-based baselines are reconstructed after each update window, while DAPG performs localized maintenance. Full configurations and implementation details are provided in Appendix D.1.

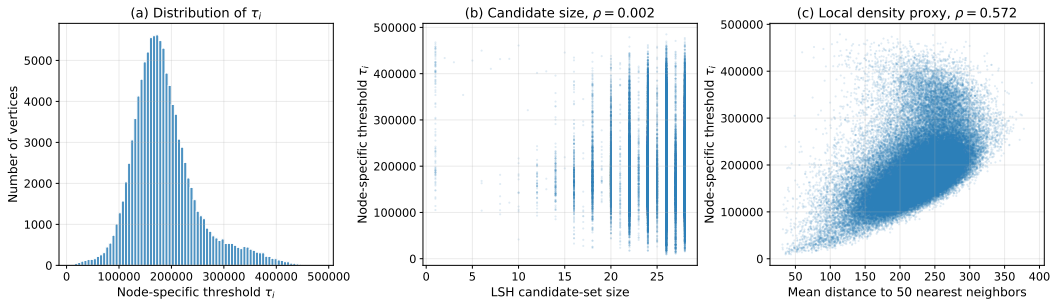
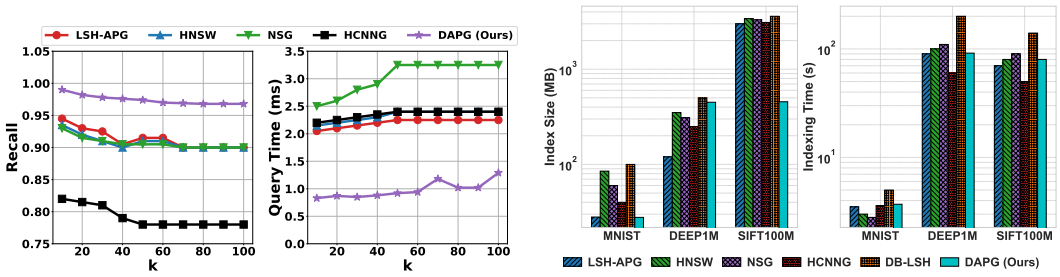


Figure 3: Node-specific pruning thresholds on SIFT1M. Panels show the threshold distribution, candidate-set size, and local 50-NN distance.

5.2 RQ1. Recall-Latency Trade-off of Distance-Aware Pruning

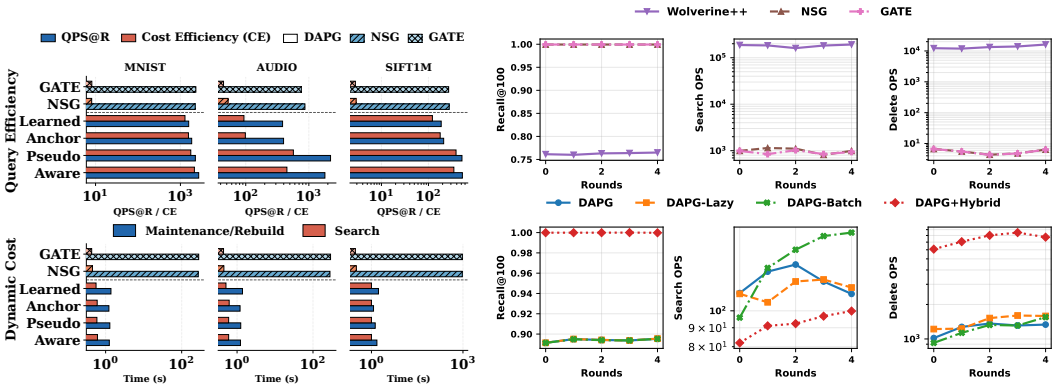
Figure 3 shows that DAPG's thresholds τ_i vary widely across SIFT1M, rather than forming a single global cutoff. Their positive correlation with mean 50-NN distance ($\rho = 0.572$ on a 100K-



(a) Recall and query latency on SIFT100M.

(b) Index size and build time.

Figure 4: Search and index-construction efficiency of DAPG. (a) Recall and query latency on SIFT100M. (b) Index size and build time across datasets.



(a) Search throughput and cost efficiency.

(b) Comparison with Wolverine++, NSG, and GATE.

Figure 5: Update efficiency of DAPG. (a) Query and dynamic efficiency across datasets. Higher QPS@R and CE are better. Lower maintenance and search times are better. (b) Post-update recall and update throughput on SIFT1M.

node sample) indicates smaller thresholds in dense regions and larger thresholds in sparse regions, providing direct evidence of density-aware local pruning. The corresponding sensitivity results in Appendix D.2 show that this adaptation preserves high recall while maintaining low query latency across threshold settings.

5.3 RQ2. DAPG vs. State-of-the-Art ANN

We compare DAPG against leading graph- and LSH-based baselines to evaluate its recall-latency trade-off, build time, index size, and traversal efficiency. Compared with LSH-APG, DAPG improves the recall-latency trade-off across the evaluated datasets. The largest gain occurs on SIFT1M, where DAPG achieves higher recall while reducing query latency by 65.6%. Detailed numerical comparisons with LSH-APG are reported in Appendix D.3.

Figure 4(a) shows that DAPG maintains recall above 0.97 across all k on SIFT100M with sub-millisecond to low-millisecond query latency, while HNSW (Malkov & Yashunin, 2020), NSG (Fu et al., 2019), and LSH-APG (Zhao et al., 2023) converge near 0.90 recall under comparable conditions. Figure 4(b) compares index size and build time against LSH-APG, HNSW, NSG, HCNNG, and DB-LSH on MNIST, DEEP1M, and SIFT100M. DAPG substantially reduces memory footprint relative to HNSW and NSG. Additional search and traversal-efficiency results are provided in Appendix D.3.

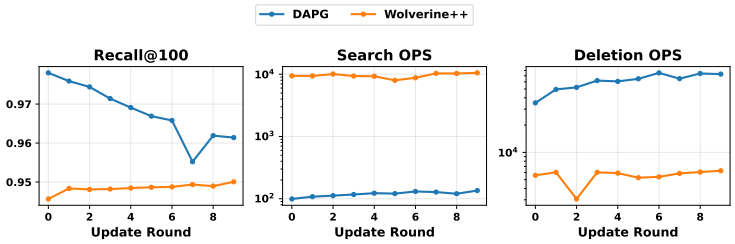


Figure 6: Dynamic maintenance on MS-MARCO1M passage embeddings.

5.4 RQ3. Dynamic Maintenance Efficiency

Figure 5(a) and Figure 5(b) compare DAPG with rebuild-based and dynamic ANN baselines. DAPG configurations maintain high post-update recall while avoiding full index reconstruction. In particular, DAPG-Pseudo reduces maintenance time by $42.7\times$ to $715\times$ relative to NSG and GATE rebuilds while maintaining competitive matched-recall search performance. Wolverine++ provides higher update throughput under the evaluated setting but lower Recall@100, whereas DAPG maintains substantially higher post-update recall. This recall gap is consistent with the different maintenance strategies. Wolverine++ repairs disrupted monotonic search paths, whereas DAPG re-prunes affected neighborhoods after each update to preserve its local graph structure.

We further evaluate DAPG on MS-MARCO1M in Figure 6. Over 10 update rounds, DAPG attains higher average and final Recall@100 than Wolverine++ ($0.9680/0.9614$ vs. $0.9485/0.9501$) and about $11.0\times$ higher deletion throughput. DAPG+Hybrid further improves recall at additional insertion cost. Further dynamic results appear in Appendix D.4.

6 Related Work

Graph-based ANN methods include hierarchical HNSW (Malkov & Yashunin, 2020) and sparse MRNG-inspired NSG (Fu et al., 2019). Graph construction can be costly, with brute-force kNN requiring $\mathcal{O}(n^2)$ comparisons and NN-Descent reducing this cost through iterative refinement (Dong et al., 2011). Recent work improves construction and traversal through LSH seeding, hierarchies, learned probing, and query-aware routing (Zhao et al., 2023; Yang et al., 2025; Vargas Muñoz et al., 2019; Jiang et al., 2025; Zheng et al., 2023; Ruan et al., 2025). PACMANN (Zhou et al., 2025) targets privacy-preserving graph-based ANN search but does not address dynamic updates.

For dynamic vector search, SPFresh (Xu et al., 2023) uses local partition rebalancing, FreshDiskANN (Singh et al., 2021) combines in-memory update staging with an SSD-resident graph, and Wolverine (Liu et al., 2025) repairs monotonic search paths disrupted by deletions. In contrast, DAPG combines density-aware sparsification with localized insert and delete maintenance in an in-memory proximity graph, reducing redundant edges and update cost while preserving high-recall search.

7 Conclusion

We propose DAPG for efficient dynamic approximate nearest neighbor search. First, DAPG applies node-specific percentile pruning to adapt edge retention to local neighborhood geometry. Second, adaptive global sparsification limits high-degree nodes and reduces redundant connectivity. Third, localized maintenance updates only affected neighborhoods during insertions and deletions, avoiding full graph reconstruction. Experiments across seven datasets and eleven representative ANN baselines show that DAPG improves the recall-latency trade-off, reduces maintenance cost relative to rebuild-based baselines, and achieves up to $11.0\times$ higher deletion throughput than a state-of-the-art dynamic ANN baseline while maintaining high post-update recall. DAPG shows that node-specific, distribution-aware pruning can improve the recall-efficiency trade-off of LSH-seeded graph search. Future work will study incremental repair strategies that distribute graph maintenance across update batches while maintaining high post-update recall.

REFERENCES

Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 29–38, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783405. URL https://doi.org/10.1145/2783258.2783405.

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327494. URL https://doi.org/10.1145/1327452.1327494.

Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2019.02.006. URL https://www.sciencedirect.com/science/article/pii/S0306437918303685.

Artem Babenko and Victor Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset, 2018. URL https://arxiv.org/abs/1611.09268.

Dmitry Baranchuk, Matthijs Douze, Yash Upadhyay, and I Zeki Yalniz. Dedrift: Robust similarity search under content drift. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11026–11035, 2023.

Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, May 1990. ISSN 0163-5808. doi: 10.1145/93605.98741. URL https://doi.org/10.1145/93605.98741.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL https://doi.org/10.1145/361002.361007.

Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 97–104, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143857. URL https://doi.org/10.1145/1143844.1143857.

Jun Woo Chung, Huawei Lin, and Weijie Zhao. Locality-sensitive indexing for graph-based approximate nearest neighbor search. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, pp. 2418–2428, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715921. doi: 10.1145/3726302.3730028. URL https://doi.org/10.1145/3726302.3730028.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pp. 253–262, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138857. doi: 10.1145/997817.997857. URL https://doi.org/10.1145/997817.997857.

Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pp. 577–586, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306324. doi: 10.1145/1963405.1963487. URL https://doi.org/10.1145/1963405.1963487.

Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, January 2019. ISSN 2150-8097. doi: 10.14778/3303753.3303754. URL https://doi.org/10.14778/3303753.3303754.

Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pp. 541–552, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312479. doi: 10.1145/2213836.2213898. URL https://doi.org/10.1145/2213836.2213898.

Thanaa M Ghanem, Moustafa A Hammad, Mohamed F Mokbel, Walid G Aref, and Ahmed K Elmagarmid. Incremental evaluation of sliding-window queries over data streams. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):57–72, 2007.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pp. 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606157.

Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2): 47–57, June 1984. ISSN 0163-5808. doi: 10.1145/971697.602266. URL https://doi.org/10.1145/971697.602266.

Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. Fibinet: combining feature importance and bilinear feature interaction for click-through rate prediction. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, pp. 169–177, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3347043. URL https://doi.org/10.1145/3298689.3347043.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pp. 604–613, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919629. doi: 10.1145/276698.276876. URL https://doi.org/10.1145/276698.276876.

Mengxu Jiang, Zhi Yang, Fangyuan Zhang, Guanhao Hou, Jieming Shi, Wenchao Zhou, Feifei Li, and Sibo Wang. Digra: A dynamic graph indexing for approximate nearest neighbor search with range filter. *Proc. ACM Manag. Data*, 3(3), June 2025. doi: 10.1145/3725399. URL https://doi.org/10.1145/3725399.

Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011a. doi: 10.1109/TPAMI.2010.57.

Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 861–864, 2011b. doi: 10.1109/ICASSP.2011.5946540.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2019.

Dawei Liu, Bolong Zheng, Ziyang Yue, Fuhao Ruan, Xiaofang Zhou, and Christian S. Jensen. Wolverine: Highly efficient monotonic search path repair for graph-based ann index updates. *Proc. VLDB Endow.*, 18(7):2268–2280, March 2025. ISSN 2150-8097. doi: 10.14778/3734839.3734860. URL https://doi.org/10.14778/3734839.3734860.

Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pp. 950–961. VLDB Endowment, 2007. ISBN 9781595936493.

Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45: 61–68, 2014. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2013.10.006. URL https://www.sciencedirect.com/science/article/pii/S0306437913001300.

Solmaz Seyed Monir, Irene Lau, Shubing Yang, and Dongfang Zhao. Vectorsearch: Enhancing document retrieval with semantic embeddings and optimized search, 2024. URL https://arxiv.org/abs/2409.17383.

Solmaz Seyed Monir, Dongfang Zhao, and Yan Bai. Nexusindex: A self-optimizing multimodal framework for fake news detection with dynamic indexing and retrieval. In *2025 IEEE 8th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, pp. 62–68, 2025. doi: 10.1109/MIPR67560.2025.00018.

Naoki Ono and Yusuke Matsui. Relative nn-descent: A fast index construction for graph-based approximate nearest neighbor search. In *Proceedings of the 31st ACM International Conference on Multimedia*, MM '23, pp. 1659–1667, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701085. doi: 10.1145/3581783.3612290. URL https://doi.org/10.1145/3581783.3612290.

James Jie Pan, Jianguo Wang, and Guoliang Li. Survey of vector database management systems, 2023. URL https://arxiv.org/abs/2310.14021.

Jianbin Qin, Wei Wang, Chuan Xiao, and Ying Zhang. Similarity query processing for high-dimensional data. *Proc. VLDB Endow.*, 13(12):3437–3440, August 2020. ISSN 2150-8097. doi: 10.14778/3415478.3415564. URL https://doi.org/10.14778/3415478.3415564.

RSIA-LIESMARS-WHU. Lshbox audio dataset. https://github.com/RSIA-LIESMARS-WHU/LSHBOX-sample-data, 2025. Accessed: 2025-07-29.

Jiancheng Ruan, Tingyang Chen, Renchi Yang, Xiangyu Ke, and Yunjun Gao. Empowering graph-based approximate nearest neighbor search with adaptive awareness capabilities. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, pp. 2444–2454, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714542. doi: 10.1145/3711896.3736930. URL https://doi.org/10.1145/3711896.3736930.

Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. In Douwe Kiela, Marco Ciccone, and Barbara Caputo (eds.), *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, volume 176 of *Proceedings of Machine Learning Research*, pp. 177–189. PMLR, 06–14 Dec 2022. URL https://proceedings.mlr.press/v176/simhadri22a.html.

Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search, 2021. URL https://arxiv.org/abs/2105.09613.

Yao Tian, Xi Zhao, and Xiaofang Zhou. Db-lsh 2.0: Locality-sensitive hashing with query-based dynamic bucketing. *IEEE Transactions on Knowledge and Data Engineering*, 36(3):1000–1015, 2024. doi: 10.1109/TKDE.2023.3295831.

Javier Vargas Muñoz, Marcos A. Gonçalves, Zanoni Dias, and Ricardo da S. Torres. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recogn.*, 96(C), December 2019. ISSN 0031-3203. doi: 10.1016/j.patcog.2019.106970. URL https://doi.org/10.1016/j.patcog.2019.106970.

Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, July 2021. ISSN 2150-8097. doi: 10.14778/3476249.3476255. URL https://doi.org/10.14778/3476249.3476255.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, Peng Cheng, and Mao Yang. Spfresh: Incremental in-place update for billion-scale vector search. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 545–561, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613166. URL https://doi.org/10.1145/3600006.3613166.

Zhaozhuo Xu, Weijie Zhao, Shulong Tan, Zhixin Zhou, and Ping Li. Proximity graph maintenance for fast online nearest neighbor search, 2022. URL https://arxiv.org/abs/2206.10839.

Shuo Yang, Jiadong Xie, Yingfan Liu, Jeffrey Xu Yu, Xiyue Gao, Qianru Wang, Yanguo Peng, and Jiangtao Cui. Revisiting the index construction of proximity graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 18(6):1825–1838, February 2025. ISSN 2150-8097. doi: 10.14778/3725688.3725709. URL https://doi.org/10.14778/3725688.3725709.

Tiannuo Yang, Zebin Yao, Bowen Jin, Lixiao Cui, Yusen Li, Gang Wang, Xiaoguang Liu, and Willie Neiswanger. Demystifying and enhancing the efficiency of large language model based search agents. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 41432–41453, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/44bed3ea6eb269aa71ef1f9630c0139b-Paper-Conference.pdf.

Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '93)*, pp. 311–321, Austin, TX, USA, 1993. Society for Industrial and Applied Mathematics.

Dongfang Zhao. Mcgi: Manifold-consistent graph indexing for billion-scale disk-resident vector search, 2026. URL https://arxiv.org/abs/2601.01930.

Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proc. VLDB Endow.*, 16(8):1979–1991, April 2023. ISSN 2150-8097. doi: 10.14778/3594512.3594527. URL https://doi.org/10.14778/3594512.3594527.

Bolong Zheng, Ziyang Yue, Qi Hu, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. Learned probing cardinality estimation for high-dimensional approximate nn search. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pp. 3209–3221, 2023. doi: 10.1109/ICDE55515.2023.00246.

Mingxun Zhou, Elaine Shi, and Giulia Fanti. Pacmann: Efficient private approximate nearest neighbor search. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 22791–22811, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/391d50b3fe1c59b3e2b8b644e0c8fe81-Paper-Conference.pdf.

APPENDIX

CONTENTS

1 INTRODUCTION **1**

2 PRELIMINARIES **2**

3 Distance-Aware Pruned Graph Index **3**

3.1 Local Distance-Aware Pruning 3

3.2 Global Sparsification Strategy 4

4 Query Processing and Dynamic Maintenance **5**

4.1 LSH-Seeded Query Traversal 5

4.2 Localized Insert/Delete Maintenance 5

4.3 Theoretical Analysis 6

5 EVALUATIONS **6**

5.1 Experimental Setup 7

5.2 RQ1. Recall-Latency Trade-off of Distance-Aware Pruning 7

5.3 RQ2. DAPG vs. State-of-the-Art ANN 8

5.4 RQ3. Dynamic Maintenance Efficiency 9

6 Related Work **9**

7 Conclusion **9**

A Additional Related Work **16**

B Notation and ANN Definitions **16**

B.1 Notation 16

B.2 Additional LSH and Graph-Based ANN Background 16

C Additional Method and Theoretical Details **17**

C.1 Detailed Construction and Pruning Procedures 17

C.2 Proof of the Degree and Sparsity Bound 18

C.3 Traversal-Work Decomposition 18

C.4 Conditional Recall Preservation 18

C.5 Complexity Analysis 18

C.6 DAPG Query Procedure 19

D Additional Experimental Details **19**

D.1 Experimental Settings 19
D.2 Pruning Ablation and Threshold Sensitivity 21
D.3 Search Sensitivity and Query-Efficiency Analysis 21
D.4 Additional Dynamic Evaluation 22

E Reproducibility Statement **24**

F AI Use Statement **24**

A Additional Related Work

High-dimensional similarity search includes tree-based indexes, hashing, quantization, and proximity graphs (Qin et al., 2020). Tree-based methods such as k-d trees, vp-trees, R-trees, and cover trees can lose pruning effectiveness in high dimensions (Bentley, 1975; Yianilos, 1993; Guttman, 1984; Beckmann et al., 1990; Beygelzimer et al., 2006). Hashing-based ANN methods use locality-sensitive hashing for sublinear candidate retrieval (Indyk & Motwani, 1998), with extensions based on p-stable projections (Datar et al., 2004), collision counting (Gan et al., 2012), and query-dependent bucketing (Tian et al., 2024). Achieving high recall may require additional hash tables or probing, increasing memory or query cost (Lv et al., 2007; Tian et al., 2024).

B Notation and ANN Definitions

B.1 Notation

Table 1 summarizes the notation used throughout the paper, including graph structure, candidate generation, pruning, traversal, recall, and dynamic-update terms.

Table 1: Notation summary

Symbol	Description
$\mathcal{O}$, n, d	Dataset, size, and dimensionality; $\mathcal{O} \subset \mathbb{R}^d$.
q, δ	Query and distance metric (e.g., ℓ_2).
k, $N_k(q)$	Target k and true Top-k neighbors.
$G = (V, E)$, $G' = (V, E'')$	Original proximity graph and pruned DAPG graph.
$N(v)$, $\deg(v)$, $\bar{d}$	Out-neighborhood, node degree, and average degree.
$\mathcal{H}$, C_q	LSH family and candidate set for q.
$\mathcal{P}_{\text{local}}$, p, τ_i	Local pruning operator, pruning percentile, and node-specific threshold.
$\mathcal{P}_{\text{global}}$, T'	Global pruning operator and adaptive degree cap.
$N'(v)$, $N''(v)$	Neighborhoods after local and global pruning.
$\bar{d}_{\text{local}}$, $\bar{d}_{\text{DAPG}}$	Average degree after local pruning and final DAPG pruning.
$\beta(\ell)$	Expected number of expansion steps to reach the relevant k-NN region under probe budget ℓ.
C_Q	Expected number of candidates examined by LSH-seeded retrieval.
C_Q^{DAPG}	Expected DAPG traversal cost, $\mathcal{O}\big(\bar{d}_{\text{DAPG}}\,\beta(\ell)\big)$.
ρ_{cand}, ρ_{loc}, ρ_{reach}	Candidate coverage, local-pruning preservation, and post-pruning reachability terms.
T_{build}, T_{upd}	Build time and amortized update time.

B.2 Additional LSH and Graph-Based ANN Background

We assume that the candidate-generation backend provides candidate coverage ρ_{cand}, that $\delta(\cdot,\cdot)$ is a metric, and that per-node candidate-distance percentiles are well defined.

For any candidate set S, Top-$k(S)$ denotes the k elements of S closest to the current reference point according to δ. The reference point is q during query processing and v_i during graph construction.

Given an approximation factor $c \geq 1$, the (c, k)-ANN problem returns an approximate neighbor set $\hat{N}_k(q)$ such that, for each $\hat{v} \in \hat{N}_k(q)$, there exists $v^\star \in N_k(q)$ satisfying

$$\delta(q, \hat{v}) \leq c\,\delta(q, v^\star).$$

We report search accuracy using

$$\text{Recall@}k = \frac{|\hat{N}_k(q) \cap N_k(q)|}{k},$$

averaged over all queries. Query efficiency is measured using end-to-end latency, QPS, and the number of graph expansions.

LSH (Indyk & Motwani, 1998; Andoni & Indyk, 2008; Datar et al., 2004) supports sublinear nearest-neighbor search by mapping high-dimensional vectors into hash buckets so that nearby points collide with higher probability than distant points. A hash family $\mathcal{H}$ is (r, cr, p_1, p_2)-sensitive if $\|x-y\| \leq r$

implies $\Pr[h(x) = h(y)] \geq p_1$ and $\|x - y\| > cr$ implies $\Pr[h(x) = h(y)] \leq p_2$, with $p_1 > p_2$. Multiple hash functions and tables improve candidate coverage, although bucket boundaries can separate geometrically nearby vectors and reduce recall (Lv et al., 2007; Gan et al., 2012).

For query q, LSH returns $C_q = \{o_i \in \mathcal{O} \mid h(o_i) = h(q) \text{ for some } h \in \mathcal{H}\}$, optionally using multi-table or multi-probe strategies. In DAPG, LSH serves as the seed mechanism for candidate generation, with its parameters controlling the expected candidate size $|C_q|$ and coverage probability $\rho_{\text{cand}} = \Pr[N_k(q) \subseteq C_q]$.

Graph-based ANN methods construct a proximity graph $G = (V, E)$ whose edges connect approximate neighbors. Queries typically start from one or more entry points and perform greedy or best-first traversal according to $\delta(q, \cdot)$. Representative graph-based methods include NSW (Malkov et al., 2014), HNSW (Malkov & Yashunin, 2020), and NSG (Fu et al., 2019). Fixed degree constraints can retain redundant edges in dense neighborhoods while providing limited flexibility for edge selection in sparse regions (Dong et al., 2011).

C Additional Method and Theoretical Details

C.1 Detailed Construction and Pruning Procedures

This section presents the detailed DAPG construction and pruning procedures, including LSH-seeded candidate generation, node-specific percentile filtering, and global degree control. These procedures are described in DAPG Index Construction, LocalPrune, and GlobalPrune.

Algorithm 2: DAPG Index Construction

Input: Dataset $\mathcal{O}$, LSH parameters, k, percentile p, degree cap T'
Output: Pruned graph $G' = (V, E'')$

Build LSH tables over $\mathcal{O}$;
foreach $o_i \in \mathcal{O}$ **do**
  Retrieve candidate set C_{o_i} from the LSH index;
  $N(o_i) \leftarrow \text{Top}_k(C_{o_i})$;
  $N'(o_i) \leftarrow \text{LocalPrune}(o_i, N(o_i), p)$;
  $N''(o_i) \leftarrow \text{GlobalPrune}(o_i, N'(o_i), T')$;
Construct E'' from $\{N''(o_i)\}_{i=1}^{n}$;
return $G' = (V = \mathcal{O}, E'')$;

Algorithm 3: LocalPrune $\mathcal{P}_{\text{local}}$

Input: Node v_i, neighbors $N(v_i)$, percentile p
Output: Pruned neighborhood $N'(v_i)$

Compute distance set $D \leftarrow \{\delta(v_i, v_j) \mid v_j \in N(v_i)\}$;
Compute threshold $\tau_i \leftarrow \text{Percentile}_p(D)$;
return $N'(v_i) \leftarrow \{v_j \in N(v_i) \mid \delta(v_i, v_j) \leq \tau_i\}$;

Algorithm 4: GlobalPrune $\mathcal{P}_{\text{global}}$

Input: Node v_i, pruned neighbors $N'(v_i)$, degree cap T'
Output: Globally pruned neighborhood $N''(v_i)$

if $|N'(v_i)| \leq T'$ **then**
  return $N''(v_i) \leftarrow N'(v_i)$;
else
  return $N''(v_i) \leftarrow \text{Top}_{T'}(N'(v_i))$ sorted by increasing $\delta(v_i, \cdot)$;

C.2 Proof of the Degree and Sparsity Bound

Proof of Proposition 1. For each node $v_i \in V$, local pruning produces $N'(v_i) = \{v_j \in N(v_i) : \delta(v_i, v_j) \leq \tau_i\}$. Let $r_i = |N'(v_i)|$. If distances are distinct at the percentile boundary, then $r_i \approx p|N(v_i)|$ up to rounding. Ties at τ_i may retain additional neighbors.

Global pruning keeps at most T' outgoing edges, hence $|N''(v_i)| = \min\{r_i, T'\} \leq T'$. Therefore,

$$\bar{d}_{\mathrm{DAPG}} = \frac{1}{|V|} \sum_{v_i \in V} |N''(v_i)| \leq T', \qquad |E''| = \sum_{v_i \in V} |N''(v_i)| \leq |V| T'. \tag{8}$$

□

Relation to the pruning percentile. If ties at τ_i are absent, or deterministically broken to retain at most the prescribed fraction, then $|N'(v_i)| \leq \lceil p|N(v_i)| \rceil$, and thus

$$|N''(v_i)| \leq \min\{\lceil p|N(v_i)| \rceil, T'\}. \tag{9}$$

Let $\bar{d}_{\mathrm{pre}} = |V|^{-1} \sum_{v_i \in V} |N(v_i)|$. Ignoring rounding gives the useful approximation $\bar{d}_{\mathrm{DAPG}} \lesssim \min\{p\bar{d}_{\mathrm{pre}}, T'\}$. The exact guarantees remain those in Equation (8).

C.3 Traversal-Work Decomposition

Proof of Proposition 2. Expanding node v requires scanning $|N''(v)|$ outgoing neighbors, so the expected number of distance computations per expansion is $\mathcal{O}(\bar{d}_{\mathrm{DAPG}})$. If $\beta(\ell)$ is the expected number of expansions needed to reach the relevant k-NN region, then

$$C_Q^{\mathrm{DAPG}} = \mathcal{O}\big(\bar{d}_{\mathrm{DAPG}} \beta(\ell)\big). \tag{10}$$

□

Interpretation. The bound does not assume $\beta(\ell) = \mathcal{O}(\log |V|)$ and therefore does not by itself imply sublinear complexity in dataset size. It isolates the two main contributors to traversal work: average branching factor $\bar{d}_{\mathrm{DAPG}}$ and expansion count $\beta(\ell)$.

C.4 Conditional Recall Preservation

Proof of Proposition 3. Let $E = E_{\mathrm{cand}} \cap E_{\mathrm{loc}} \cap E_{\mathrm{reach}}$. By the sufficiency assumption in the proposition, event E is sufficient for best-first traversal to recover $N_k(q)$. By the probability chain rule,

$$\begin{aligned} \Pr[E] &= \Pr[E_{\mathrm{cand}}] \Pr[E_{\mathrm{loc}} \mid E_{\mathrm{cand}}] \Pr[E_{\mathrm{reach}} \mid E_{\mathrm{cand}} \cap E_{\mathrm{loc}}] \\ &= \rho_{\mathrm{cand}} \rho_{\mathrm{loc}} \rho_{\mathrm{reach}}. \end{aligned} \tag{11}$$

Therefore,

$$\Pr[S(q, G') = N_k(q)] \geq \rho_{\mathrm{cand}} \rho_{\mathrm{loc}} \rho_{\mathrm{reach}}. \tag{12}$$

No independence assumption is required because the latter two probabilities are conditional. □

Scope. The result is conditional rather than an unconditional recall guarantee. Both ρ_{loc} and ρ_{reach} depend on p, T', graph structure, and the candidate distribution, and are evaluated empirically using Recall@k, pruning ratio, path length, visited nodes, and latency.

C.5 Complexity Analysis

Proposition 4 (Index Construction Complexity)**.** *Let $n = |V|$, d be the vector dimensionality, k the neighborhood size used in Top_k, and C_Q the expected number of candidates examined by the LSH-seeded retrieval used during construction. Then*

$$T_{\mathrm{build}}^{\mathrm{DAPG}} = \mathcal{O}(ndC_Q) + \mathcal{O}(nk \log k) + \mathcal{O}(n\bar{d}_{\mathrm{DAPG}}). \tag{13}$$

Proof. The LSH-seeded query contributes $\mathcal{O}(ndC_Q)$ over all n points. Computing the local percentile by sorting at most k candidate distances costs $\mathcal{O}(nk \log k)$ overall. Materializing the retained adjacency lists costs $\mathcal{O}(|E''|)$. □

Table 2: Comparison of build and query complexity across representative ANN methods.

Method	Build	Query
HNSW (Malkov & Yashunin, 2020)	$\tilde{\mathcal{O}}(n \log n)$	$\mathcal{O}(\bar{d}\beta)$
NSG (Fu et al., 2019)	$\tilde{\mathcal{O}}(kn^{1+1/d} + f(n))$	$\tilde{\mathcal{O}}(cn^{1/d})$
HCNNG (Vargas Muñoz et al., 2019)	$\tilde{\mathcal{O}}(mnd)$	$\mathcal{O}(\beta \bar{d}_{\text{sub}})$
FastNSG (Yang et al., 2025)	$\tilde{\mathcal{O}}(kn^{1+1/d} + f(n))$	$\tilde{\mathcal{O}}(cn^{1/d})$
FastHNSW (Yang et al., 2025)	$\tilde{\mathcal{O}}(n \log n)$	$\mathcal{O}(\bar{d}\beta)$
DB-LSH (Tian et al., 2024)	$\tilde{\mathcal{O}}(n^{1+\rho})$	$\tilde{\mathcal{O}}(n^{\rho})$
LSH-APG (Zhao et al., 2023)	$\mathcal{O}(ndC_Q)$	$\mathcal{O}(dC_Q)$
DIGRA (Jiang et al., 2025)	$\mathcal{O}(n \log n\, c_{\text{upd}})$	$\mathcal{O}(\log n + \bar{d}_{\text{N}}\beta)$
DAPG	$\mathcal{O}(\mathbf{ndC_Q} + \mathbf{nk}\log\mathbf{k} + \mathbf{n\bar{d}}_{\text{DAPG}})$	$\mathcal{O}(\bar{\mathbf{d}}_{\text{DAPG}}\beta(\ell))$

$\tilde{\mathcal{O}}$ hides polylogarithmic factors. For DAPG, $\bar{d}_{\text{DAPG}} \lesssim \min\{p\bar{d}_{\text{pre}}, T'\}$.

Space complexity. The graph requires $\mathcal{O}(|E''|) = \mathcal{O}(n\bar{d}_{\text{DAPG}})$ space. If T' is fixed, then $\bar{d}_{\text{DAPG}} \leq T'$ and the graph component uses $\mathcal{O}(n)$ space, excluding vector storage and the underlying LSH structures.

Table 2 compares the build and query complexity of DAPG with representative ANN methods.

C.6 DAPG Query Procedure

Algorithm 5 summarizes the LSH-seeded best-first search used by DAPG. The traversal configurations evaluated below operate on the same pruned graph and differ only in their entry-point selection strategy.

Algorithm 5: DAPG Best-First Query

Input: Query q, pruned graph G', LSH tables, k, ef
Output: Approximate top-k neighbors R'

$S_q \leftarrow \text{LSHSeeds}(q)$; $\mathcal{Q}, \mathcal{R}, \mathcal{V} \leftarrow \emptyset$;
foreach $s \in S_q$ **do**
  $\mathcal{Q}.\text{push}(s, \delta(q,s))$; $\mathcal{R}.\text{push}(s, \delta(q,s))$;
while $\mathcal{Q} \neq \emptyset$ **do**
  $(u, d_u) \leftarrow \mathcal{Q}.\text{popmin}()$;
  if $u \in \mathcal{V}$ **then**
    continue
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{u\}$;
  if $|\mathcal{R}| \geq k$ ***and*** $d_u \geq \max_{r \in \mathcal{R}} \delta(q,r)$ **then**
    break
  foreach $v \in N''(u) \setminus \mathcal{V}$ **do**
    $\mathcal{Q}.\text{push}(v, \delta(q,v))$; $\mathcal{R}.\text{push}(v, \delta(q,v))$;
    while $|\mathcal{R}| > ef$ **do**
      $\mathcal{R}.\text{popmax}()$
return top-$k(\mathcal{R})$;

D Additional Experimental Details

D.1 Experimental Settings

Datasets. Table 3 summarizes the dataset statistics used in our evaluation. LID is estimated using the MLE/Hill estimator (Amsaleg et al., 2015), and all vectors are stored using `float32` encoding.

Hardware and implementation. Experiments were conducted on both a local workstation and Microsoft Azure virtual machines. The workstation featured a 13th Gen Intel® Core™ i9-13900HX processor (24 cores, 32 threads, 2.2,GHz) running `Ubuntu 20.04`. All vectors were stored in

Table 3: Dataset statistics.

Dataset	Cardinality	d	n_q	LID	Similarity	Raw GiB
Audio (RSIA-LIESMARS-WHU, 2025)	5×10^4	192	10^3	21.5	L2	0.036
MNIST (Lecun et al., 1998)	6×10^4	784	10^4	12.7	L2	0.175
Deep1M (Babenko & Lempitsky, 2016)	10^6	96	10^4	26.0	Cos.	0.358
SIFT1M (Jégou et al., 2011b)	10^6	128	10^4	12.9	L2	0.477
SIFT100M (Jégou et al., 2011b)	10^8	128	10^4	23.7	L2	47.684
Text2Image1M (Simhadri et al., 2022)	10^6	200	10^5	8.4†	Cos.	0.745
MS-MARCO1M (Bajaj et al., 2018)	10^6	384	2×10^2	27.8‡	Cos.	1.431

Raw GiB is computed as $n \times d \times 4/2^{30}$ for float32 vectors. LID is estimated using the MLE/Hill estimator (Amsaleg et al., 2015). † Estimated on the available 1M-vector subset. ‡ Estimated on three random 10K samples of normalized Sentence-BERT passage embeddings using $k = 50$.

`float32` precision, and each configuration was evaluated over 10^4 queries with results averaged across five runs. For scalability evaluation, we deployed DAPG on an Azure `Standard E64-32s v3` VM equipped with 32 vCPUs (Intel® Xeon® Platinum 8272CL), 432,GiB RAM, and 1,TB premium SSD, running `Ubuntu 22.04`. This instance is optimized for memory-intensive workloads and provides sufficient capacity for large-scale benchmarks such as `SIFT100M`. All implementations were written in C++ with OpenMP-based parallelization for index construction, while queries were executed serially.

Evaluation metrics. The pruning ratio is $\mathrm{PR} = (|\mathcal{C}| - |\mathcal{C}_{\mathrm{kept}}|)/|\mathcal{C}|$, where $\mathcal{C}$ is the candidate neighbor set and $\mathcal{C}_{\mathrm{kept}}$ is the subset retained after pruning. For traversal experiments, Path denotes the average number of visited nodes, MaxR denotes the maximum Recall@100 attained in a search-parameter sweep, and Ef(MaxR) is the corresponding ef search budget. For matched-recall measurements, $\mathrm{QPS} = 1000/\mathrm{Rt@0.95}$. Per-update latency is reported in milliseconds where applicable.

Static search. For the static search experiments, DAPG is configured with $K = 18$, $L = 2$, $T = 24$, and $T' = 48$. DAPG supports both adaptive and controlled global sparsity. The adaptive formulation sets $T' = \alpha\bar{d}_{\mathrm{local}}$, whereas controlled static comparisons use $T' = 48$ to ensure a common degree budget across datasets and baselines. Density-aware adaptation is provided by the node-specific threshold τ_i, while T' controls the maximum degree and overall graph sparsity.

Additional parameters include $W = 1.0$, $p_C = 0.95$, $p_Q = 0.9$, and $ef_C = 80$. Based on the LSH-APG parameter study, we set $K = 18$ instead of the default $K = 16$ for fair comparison under optimized conditions. For other baselines, we use the following configurations: HNSW ($M = 48$, $ef = 80$), NSG ($L = 40$, $R = 50$, $C = 500$), HCNNG (maximum cluster size 500, 10 clustering iterations), and DB-LSH ($c = 1.5$, $K = 12$, $L = 5$). We adopt official implementations of FastNSG, FastHNSW, and RNNDescent with their default settings. For FastHNSW, nsg_R is aligned with HNSW's M, and the candidate pool size is set to $\max\{K, L, search_L, search_K\}$.

Dynamic evaluation. For NSG and GATE, we use rebuild-per-window with FAISS-HNSW ($k = 200$, $M = 32$, $efC = 200$, $efSearch = 200$) and NSG ($L = 200$, $R = 64$, $C = 500$). GATE uses two 8-iteration k-means stages. We sweep $L \in \{100, 120, 150, 200, 300, 400\}$ for matched-recall comparisons. For traversal variants, PSEUDO uses eight entry points, AWARE reranks 512 candidates and selects four entry points for $k > 50$, ANCHOR uses precomputed anchors, and LEARNED uses softmax anchors with 128 warm-up expansions. For the random delete-and-new-insert workload, the active subset $\mathcal{A}_t$ contains $900K$ vectors from $\mathcal{X}$, where $|\mathcal{X}| = 10^6$. At each round, we delete $10K$ random vectors from $\mathcal{A}_t$ and insert $10K$ unseen vectors from $\mathcal{X} \setminus \mathcal{A}_t$. We run five rounds. DAPG uses $k = 100$, $ef = 4000$, $W = 600$, $L = 16$, $K = 24$, $T = 48$, $efC = 200$, $p_C = 0.95$, and $p_Q = 0.9$. We vary δ_{in} to evaluate sensitivity to update intensity. For DAPG+Hybrid, the auxiliary HNSW candidate layer uses $M = 32$ and returns 400 candidates for exact reranking. DAPG-LAZY and DAPG-BATCH use the same DAPG parameters, with $ef = 2100$, and are evaluated with a $900K$ active set, $10K$ updates per round, and five rounds. They differ only in their update-maintenance strategy. Wolverine++ (Liu et al., 2025) is evaluated using its FIFO update runner with $M = 32$, $ef = 300$, candidate limit 32, threshold 64, and the WolverineProMax deletion model.[1]

To evaluate dynamic text retrieval, we use MS-MARCO1M passage embeddings generated with Sentence-BERT MiniLM (Wang et al., 2020) and L2-normalized. We initialize the index with $900K$

[1] https://github.com/LDW2020/Wolverine

Table 4: τ_i sensitivity. Reported τ_i values denote mean node-specific thresholds for each evaluated setting.

DEEP1M					AUDIO				
Method	τ_i	**Recall**	**QT**	**PR**	**Method**	τ_i	**Recall**	**QT**	**PR**
DAPG	1.080	0.9750	**1.47**	**0.2954**	DAPG	1.779	**1.0000**	**239.90**	**0.5899**
	1.106	0.9660	2.72	0.2930		1.624	1.0000	254.52	0.5877
	1.148	0.9720	1.52	0.2938		1.187	0.9940	325.23	0.5644
	1.386	**0.9770**	2.18	0.2950		3.088	1.0000	244.38	0.5855
LSH-APG	–	0.9588	3.43	0.2672	LSH-APG	–	0.9946	528.13	0.4834
MNIST					**SIFT1M**				
Method	τ_i	**Recall**	**QT**	**PR**	**Method**	τ_i	**Recall**	**QT**	**PR**
DAPG	0.0958	**0.9988**	0.79	0.4971	DAPG	1.0425	0.9738	**0.92**	0.5448
	0.2734	0.9976	0.80	0.4991		1.7969	0.9732	0.96	0.5469
	0.6873	0.9984	**0.75**	0.5025		1.3976	0.9716	0.98	**0.5471**
	0.7768	0.9980	0.77	**0.5032**		2.6855	**0.9762**	**0.92**	0.5440
LSH-APG	–	0.9972	0.6823	0.4070	LSH-APG	–	0.9706	1.2684	0.4958

QT: query time (ms), PR: pruning rate.

passages and use the remaining $100K$ as a 10-round update stream, replacing $10K$ passages per round. DAPG and Wolverine++ are evaluated using the same active window and query set.

D.2 Pruning Ablation and Threshold Sensitivity

Table 4 shows that DAPG maintains high recall across the evaluated τ_i settings. Its key design distinction from LSH-APG is the combination of node-specific distance-aware pruning through τ_i with global sparsity control through T', replacing a fixed degree constraint with locally adaptive edge selection. Deletion is not evaluated in the public LSH-APG implementation[†].

Table 5 shows that the local and global pruning components are complementary. Full DAPG maintains high recall with a sparse graph and low query latency. The Global-only variant achieves similar sparsity and latency but slightly lower maximum recall, while the Local-only variant produces a much denser graph and substantially higher query, indexing, and update costs. At nearly equal average degree (36.55 vs. 36.56), Full DAPG achieves slightly higher maximum recall than Global-only (0.9772 vs. 0.9768), suggesting a modest benefit from node-specific percentile pruning. The Local-only variant reaches a much higher average degree of 149.61, showing that global capping is essential for controlling graph sparsity.

Table 5: Pruning ablation on SIFT1M. Rt@0.95 denotes query latency at Recall@100 $= 0.95$.

Pruning	**Local**	**Global**	**AvgDeg**	**MaxRecall**	**Rt@0.95**	**IndexTime (s)**	**Insert/Delete (ms)**
Full DAPG	Yes	Yes	36.55	0.9772	0.885	78.15	0.644 / 0.150
Global-only	Minimized	Yes	36.56	0.9768	0.880	77.95	0.654 / 0.095
Local-only	Yes	Relaxed	149.61	0.9769	1.736	199.47	1.974 / 1.241

D.3 Search Sensitivity and Query-Efficiency Analysis

Table 6 shows that DAPG improves query efficiency while maintaining high recall. On DEEP1M, the best evaluated DAPG setting increases recall from 0.9590 to 0.9632 while reducing query latency by 32.9%. On SIFT1M, the best evaluated setting achieves recall of 0.9870 with query latency of 0.83 ms, corresponding to a 65.6% latency reduction relative to LSH-APG. At the matched setting $k = 50$, DAPG achieves recall of 0.9738 with query latency of 0.921 ms, reducing latency by 62.0%. Across these datasets, DAPG prunes up to 55% of candidate neighbors while maintaining high recall. Figure 7 shows that recall remains high across the evaluated k and ef settings, while query time and search cost vary more substantially. Pruning ratios remain relatively stable within each dataset. Figure 8 shows that DAPG maintains competitive recall with low query latency relative

[†] LSH-APG

Table 6: DAPG vs. LSH-APG (fastG).

Dataset	Method	Recall	Query Time (ms)	Index Size (MB)	Build Time (s)	Pruning Rate	k
DEEP1M	fastG	0.9590	3.43	250	230.1	0.267	50
	DAPG	**0.9632±0.0057**	**2.30±0.20**	449	**98.3–121.5**	**0.30–0.32**	10–100
	Δ	+0.44%	+32.9%				
MNIST	fastG	0.9972	0.682	10	6.4	0.410	50
	DAPG	**0.9984±0.0010**	**0.560±0.29**	27.77‡	**3.2–4.5**	**0.50–0.53**	10–100
	Δ	+0.12%	+17.9%				
SIFT1M	fastG	0.9580	2.42	468	105	0.210	50
	DAPG	**0.9870±0.0025**	**0.83±0.00**	**455**	**70–103**	**0.54–0.55**	10–100
	Δ	+3.03%	+65.6%				
Matched comparison at $k = 50$							
DEEP1M	DAPG	0.9615	2.30	Recall Δ: +0.26%, Query Time Δ: +32.9%			
MNIST	DAPG	0.9984	0.560	Recall Δ: +0.12%, Query Time Δ: +17.9%			
SIFT1M	DAPG	0.9738	0.921	Recall Δ: +1.65%, Query Time Δ: +62.0%			

Δ denotes relative recall improvement or query-time reduction over fastG. ‡The larger MNIST index is due to denser graph construction. Values reported with ± denote 12-run mean ± standard deviation.

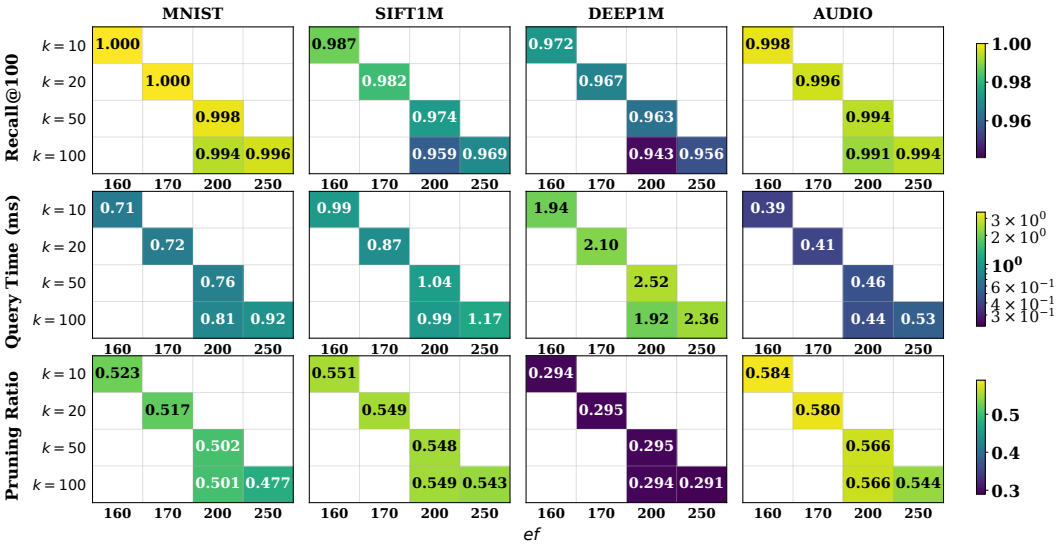


Figure 7: Sensitivity of DAPG to k and ef. DAPG maintains high Recall@100 across settings, while query time and pruning ratio reflect the efficiency and sparsity trade-off.

to ANN baselines, while settings S1 to S5 illustrate the trade-off among recall, indexing cost, and query efficiency relative to the LSH-APG baseline.

The evaluated configurations differ in entry-point selection and update maintenance, as described in Section 4.1 and Section 4.2. Table 7 shows that DAPG-PSEUDO and DAPG-AWARE preserve comparable maximum recall while reducing matched-recall latency relative to base DAPG and LSH-APG. Across W, the best configuration reduces latency by 42.8%-50.9%, whereas DAPG-ANCHOR and DAPG-LEARNED are generally slower. Table 8 shows that DAPG-AWARE achieves the highest search-only throughput at Recall@100 $= 0.95$, reaching approximately 680 QPS on D1M and 913 QPS on T2I-1M, while also achieving the shortest traversal paths.

D.4 Additional Dynamic Evaluation

Section 5.4 reports the main dynamic-maintenance comparison against Wolverine++, NSG, and GATE. Here, we provide additional results under FIFO sliding-window maintenance and random delete-and-new-insert updates on SIFT1M. For FIFO maintenance, we follow a sliding-window protocol (Ghanem et al., 2007; Baranchuk et al., 2023), building the index on the first n_c vectors and then repeatedly deleting the oldest n_d vectors and inserting n_d new vectors from the remaining dataset. Search performance is measured after each update round.

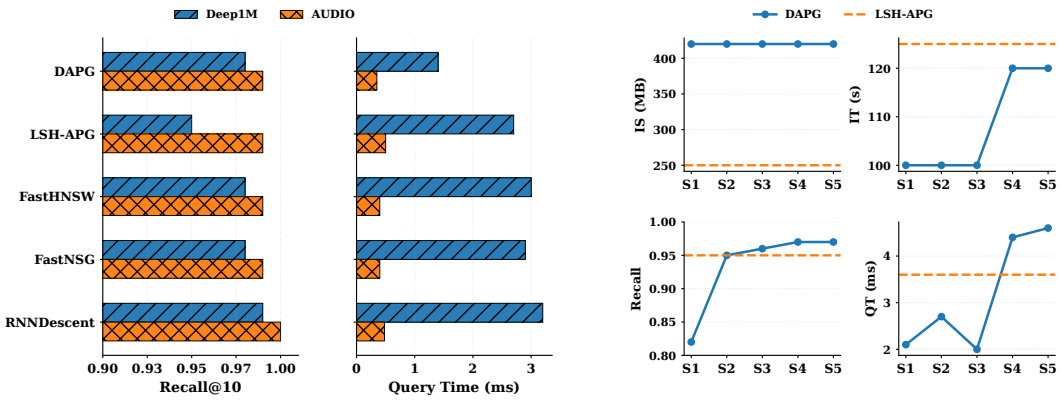


Figure 8: DAPG evaluation. Comparison with ANN baselines and LSH-APG.

Table 7: SIFT1M traversal sensitivity across LSH bucket widths W. Rt@0.95 and Rt@0.97 denote query latency at matched recall. Lower latency and update cost are better, while higher MaxR is better. The final column reports the best Rt@0.95 reduction relative to LSH-APG for each W.

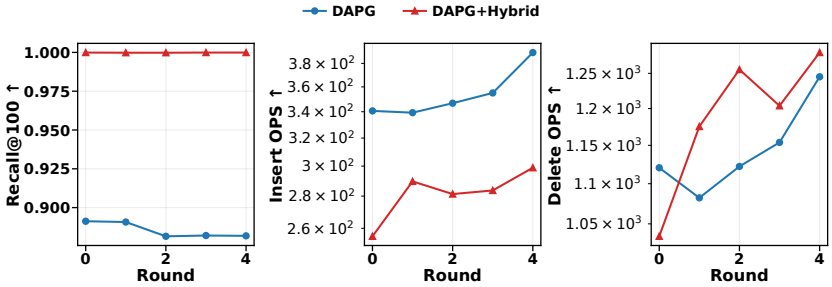
		Recall	Search Setting	Query Latency (ms)		Index and Update Cost			Best Reduction
W	Method	MaxR	Ef(MaxR)	Rt@0.95	Rt@0.97	Build (s)	Insert (ms)	Delete (ms)	vs. LSH-APG
0.1	DAPG	0.9779	1960	0.848	1.670	73.20	0.669	0.162	
	PSEUDO	**0.9784**	2000	**0.497**	0.931	73.20	0.583	0.031	**43.8%**
	AWARE	**0.9784**	2000	0.534	**0.913**	73.20	**0.582**	**0.026**	
	ANCHOR	**0.9784**	2000	2.921	3.217	73.20	0.628	0.067	
	LEARNED	**0.9784**	2000	3.097	3.417	73.20	0.605	0.042	
	LSH-APG	0.9776	2060	0.884	1.493	84.67	–	–	
0.3	DAPG	0.9772	1780	1.592	2.995	125.49	1.146	0.187	
	PSEUDO	**0.9779**	1800	1.036	1.799	125.49	1.217	0.082	
	AWARE	**0.9779**	1780	**0.848**	**1.527**	125.49	1.074	0.054	**50.9%**
	ANCHOR	**0.9779**	1820	5.758	6.452	125.49	**1.048**	**0.046**	
	LEARNED	**0.9779**	1820	6.254	6.974	125.49	1.304	0.065	
	LSH-APG	0.9778	1820	1.727	3.227	135.21	–	–	
0.5	DAPG	0.9776	1980	1.857	2.940	133.01	1.548	0.201	
	PSEUDO	**0.9779**	1920	0.949	1.989	133.01	1.136	0.067	
	AWARE	**0.9779**	1920	**0.924**	**1.864**	133.01	1.118	**0.051**	**42.8%**
	ANCHOR	**0.9779**	1920	5.730	6.637	133.01	1.210	0.056	
	LEARNED	**0.9779**	1920	6.117	6.972	133.01	**1.100**	0.066	
	LSH-APG	**0.9779**	2100	1.616	3.335	134.98	–	–	
1.0	DAPG	0.9782	2020	1.894	3.474	136.61	1.493	0.314	
	PSEUDO	**0.9788**	1840	**0.939**	1.813	136.61	1.445	0.081	**45.9%**
	AWARE	**0.9788**	1840	1.010	**1.747**	136.61	1.234	**0.063**	
	ANCHOR	**0.9788**	1860	5.956	6.565	136.61	1.178	0.065	
	LEARNED	**0.9788**	1860	6.320	6.675	136.61	**1.166**	0.071	
	LSH-APG	0.9782	2040	1.736	2.863	136.52	–	–	

Figure 9 compares DAPG and DAPG+Hybrid under FIFO and random-update workloads. DAPG maintains high post-update Recall@100, while DAPG+Hybrid approaches perfect recall at the cost of additional insertion overhead. We further evaluate update-intensity sensitivity with $\delta_{\text{in}} \in \{2\text{K}, 5\text{K}, 10\text{K}, 20\text{K}\}$ and a fixed $900K$ active set. Figure 10 shows stable performance across the evaluated update batch sizes. Figure 11 compares DAPG and DAPG+Hybrid over 50 update rounds. DAPG maintains Recall@100 above 0.97 for most rounds and above 0.95 after prolonged updates, while DAPG+Hybrid remains near 1.0 recall at the cost of lower search and insertion throughput. Deletion throughput remains comparable between the two methods.

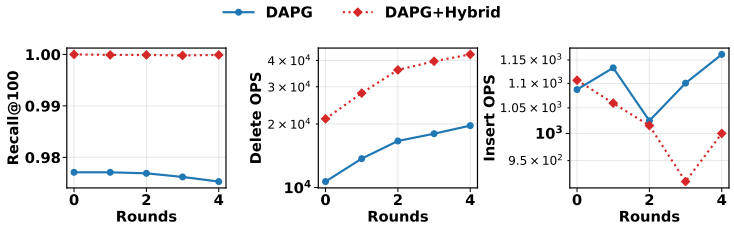
We further compare DAPG with FreshDiskANN, a dynamic graph-based ANN method for streaming updates. Figure 12 shows post-update recall and deletion throughput on SIFT1M across update rounds. DAPG and DAPG+Hybrid maintain stable recall while supporting localized deletion maintenance. We evaluate DAPG on SIFT1M over 50 random delete-and-new-insert rounds with a fixed $900K$ active set and $\delta_{\text{in}} = 2000$. Figure 13 shows that, as cumulative updates increase, DAPG be-

Table 8: Search efficiency at $k = 100$. Path and SearchRt are measured at Recall@100 $= 0.95$.

Data	Method	Path↓	SearchRt↓	QPS↑	MaxR↑	Ef(MaxR)
D1M	**DAPG-AWARE**	**5142**	**1.471**	**679.981**	**0.996**	2080
	DAPG-PSEUDO	5415	2.795	357.742	**0.996**	2080
	DAPG	5695	2.790	358.382	0.995	2100
	LSH-APG	5955	3.198	312.725	0.995	2100
T2I-1M	**DAPG-AWARE**	**1652**	**1.095**	**912.859**	0.9986	2100
	DAPG-PSEUDO	1856	1.230	812.361	0.9986	2100
	DAPG	1853	1.156	864.977	0.9985	2100
	LSH-APG	1838	1.158	863.021	0.9987	2080



(a) FIFO sliding-window updates.



(b) Random delete-and-new-insert updates.

Figure 9: Update evaluation of DAPG and DAPG+Hybrid on SIFT1M.

comes progressively sparser while maintaining high Recall@100. Zero-degree active nodes remain below 0.4%, indicating robust graph structure under the evaluated update workloads.

E Reproducibility Statement

We provide the implementation, experiment scripts, and processed results in an anonymous repository at https://anonymous.4open.science/r/DAPG. The repository includes DAPG index construction, search, localized insert/delete maintenance, and dynamic workload drivers corresponding to the methods described in Section 4.2 and the evaluation in Section 5. We also provide raw CSV outputs, plotting code, and per-round dynamic results, including recall, search throughput, update throughput, and graph statistics. Detailed experimental and dynamic-workload settings are provided in Appendix D.1 and Appendix D.4. These artifacts support reproduction of the reported results and maintenance-speedup claims.

F AI Use Statement

An LLM was used only for language editing to improve clarity and conciseness. All research ideas, methodology, experiments, and analysis were developed by the authors.

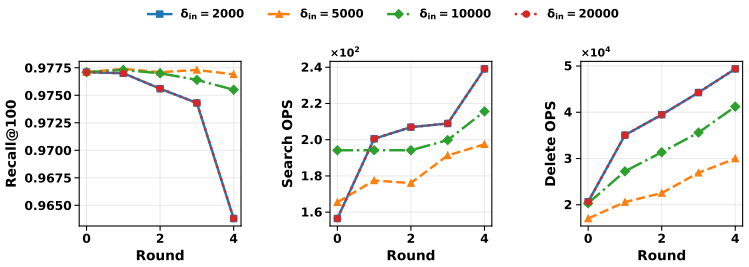


Figure 10: DAPG sensitivity to update batch size on SIFT1M.

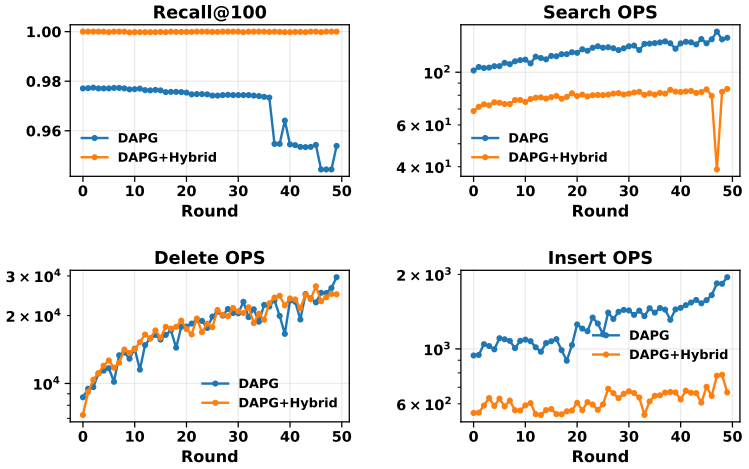


Figure 11: DAPG and DAPG+Hybrid over 50 update rounds on SIFT1M.

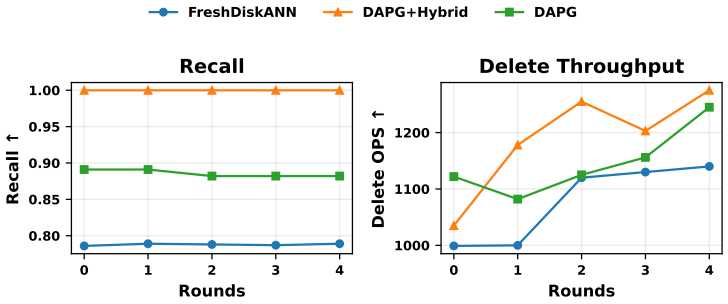


Figure 12: Post-update recall and deletion throughput on SIFT1M.

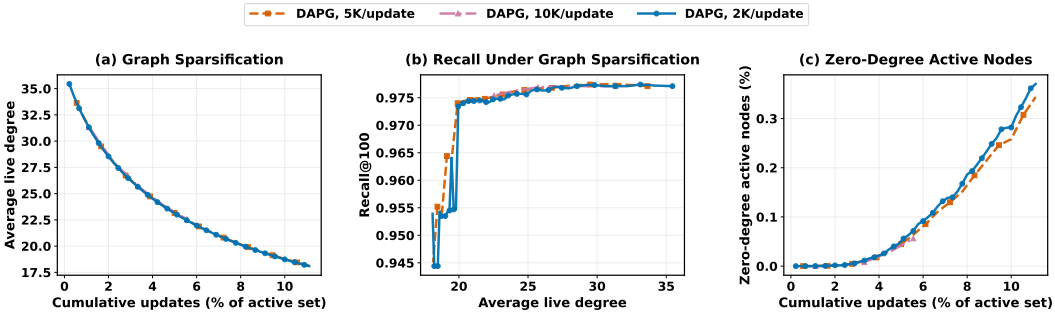


Figure 13: Structural robustness of DAPG on SIFT1M. Results are consistent across 2K, 5K, and 10K update batches, with high recall and fewer than 0.4% zero-degree active nodes.