Robust HNSW Vector Retrieval under Dynamic Product Updates via Local Rewiring

Anurag Sengupta[1]

[1]Independent Researcher, Fremont, California, United States of America

Abstract

Semantic search lies at the heart of e-commerce product catalog discovery, with vector databases and indexes serving as the backbone of low-latency retrieval. However, graph-based vector indexes such as Hierarchical Navigable Small World (HNSW) are architected under a static data assumption that conflicts with the dynamic nature of product catalogs, where insertions, deletions, and metadata updates occur continuously at scale. In this paper, we show that under heavy update workloads, existing graph-based index repair strategies silently accumulate unreachable nodes. As a consequence, affected products are silently excluded from search results, severely degrading the accuracy and reliability of semantic retrieval. We present LR-RN (Local Rewiring — Representative Neighbor), an alternative to existing re-indexing and repair logic that maintains graph connectivity under heavy update workloads. Experiments on standard Approximate Nearest Neighbor (ANN) benchmarks and an Amazon product catalog dataset demonstrate that LR-RN reduces unreachable node growth by two thirds on standard ANN benchmarks, and arrests growth entirely on production-grade product catalog data, while keeping recall degradation within 1-3% across cumulative update cycles. As LR-RN operates directly on standard HNSW graph structures, it requires no changes to the underlying index format, enabling drop-in deployment within existing production retrieval pipelines. Our implementation is publicly available at https://github.com/anurag93/VectorDB.

Keywords

HNSW, Approximate Nearest Neighbor, Dynamic Updates, Vector Databases, E-Commerce Search

1. Introduction

E-commerce has emerged as one of the most rapidly growing sectors of the global economy, fundamentally reshaping how consumers discover, evaluate, and purchase products. Users today interact with platforms not merely to transact, but to browse, compare, and research across vast and ever-expanding product catalogs. This scale of interaction has placed product search at the critical junction between user experience and revenue generation.

Mirroring advances in web search, e-commerce search has undergone a significant paradigm shift from lexical matching over product metadata to semantic understanding of user intent [1]. Rather than relying on exact keyword overlap between query terms and product descriptions, modern retrieval systems encode both queries and products into dense vector representations that capture semantic proximity. This transition has had measurable business impact. A deployment at JD.com demonstrated a 1.29% improvement in overall conversion rate and a 10.03% improvement on long-tail queries [2], directly attributable to semantic retrieval over the product catalog.

The bedrock of this semantic search paradigm is the vector embedding, a dense numerical representation of a product that encodes its semantic attributes, contextual relationships, and behavioral signals [3, 4]. These embeddings convert the structured product catalog into a high-dimensional vector space where semantically similar products cluster together, enabling efficient Approximate Nearest Neighbor (ANN) retrieval. More recently, the adoption of large language models for embedding generation has further expanded the representational power of these systems [5], making semantic retrieval increasingly central to production search pipelines.

However, a fundamental tension arises between the requirements of high-performance ANN search and the operational realities of e-commerce. Vector database systems such as Milvus [6], FAISS [7],

ECOM'26: SIGIR Workshop on eCommerce, Jul 24, 2026, Melbourne, Australia

sengupta.anurag93@gmail.com (A. Sengupta)

Weaviate [8], and GaussDB-Vector [9] that index and serve these embeddings at scale, are architecturally optimized under an assumption that the underlying data remains largely static. Hierarchical Navigable Small World (HNSW) graphs [10] which support the indexing of data within these vector databases, are constructed to maximize search accuracy and throughput for a fixed corpus. However, product catalogs are anything but static. Product metadata in large-scale e-commerce platforms changes continuously and at high frequency [11, 12]. Prices are updated, inventory fluctuates, new SKUs are introduced, and discontinued items must be removed. Across a platform of millions of listings, these changes constitute a high-velocity stream of insertions, updates, and deletions that must be reflected in the embedding index to preserve retrieval quality.

When embeddings in an HNSW index are deleted or replaced, the graph's navigability can degrade, a phenomenon we refer to as the *unreachable nodes problem* [13, 14]. Deleted nodes leave behind structural gaps in the graph, severing paths that previously connected regions of the embedding space. Nodes that lose all incoming edges become unreachable during greedy graph traversal, causing them to be systematically excluded from search results regardless of their semantic relevance, thereby affecting the reliability of the underlying system supporting search. This degradation is silent and progressive, accumulating more unreachable product nodes with each catalog update cycle, without any explicit system failure to signal the problem.

In this paper, we study the unreachable nodes problem specifically in the context of e-commerce product catalog data, where the characteristics of catalog churn, including the prevalence of long-tail products with sparse interaction signals [12] and the high frequency of structured metadata updates [11], create conditions under which graph connectivity degrades. We study the conditions under which unreachable nodes arise and characterize their impact on search recall across simulated catalog update workloads. To address this, we introduce Local Rewiring with Representative Neighbor (LR-RN), which selects a single random neighbor as a representative of the updated node and performs a bounded local rewiring among its one-hop neighbors, as a mitigation strategy suited to the update patterns of production e-commerce systems.

We benchmark LR-RN on standard ANN dataset SIFT-1M, Deep-100K and curated Amazon product catalogs to emulate a production product catalog dataset. Our evaluation explores update latency, search latency, and Recall@10, comparing it against existing techniques including Native-RU update methodology [15]. More importantly, we calibrate the performance of LR-RN with respect to unreachable nodes phenomenon, observing the structural impact of large-scale updates.

To this end, we propose LR-RN, a lightweight neighborhood repair strategy for HNSW-based vector databases that preserves graph connectivity under high-frequency product catalog updates.

2. Background

2.1. Approximate Nearest Neighbor Search and Hierarchical Navigable Small World

Approximate Nearest Neighbor (ANN) search has become a fundamental building block for large-scale retrieval systems, particularly in applications involving high-dimensional vector representations such as recommendation systems, semantic search, and retrieval-augmented generation (RAG) [16, 17, 18]. Instead of performing exhaustive comparisons across the entire dataset, ANN methods aim to efficiently retrieve a subset of vectors that are approximately closest to a given query, significantly reducing query latency while maintaining high recall [19].

Hierarchical Navigable Small World (HNSW) graphs [10] have emerged as one of the most widely adopted ANN indexing structures due to their favorable trade-off between search efficiency and accuracy. HNSW organizes data points into a multi-layer graph, where each node is assigned a maximum level sampled from an exponentially decaying distribution, resulting in fewer nodes at higher layers. This hierarchical structure enables efficient long-range navigation at upper layers and fine-grained local search at lower layers.

Each node maintains a bounded number of connections per layer, controlled by the parameter M, referred to as the branching factor, which defines the maximum number of neighbors per node at higher

layers. The base layer (layer 0) is typically denser, allowing up to $M_0 = 2M$ neighbors to improve local connectivity and search accuracy. A node is assigned to a maximum layer ℓ using this distribution, with probability proportional to $\frac{1}{\ln(M)}$, resulting in exponentially fewer nodes at higher layers. The choice of M directly impacts graph sparsity, memory consumption, and recall. Larger values improve connectivity and robustness at the cost of higher storage and construction overhead.

During insertion, a new vector is introduced into the graph via an entry point node and navigates from higher layers to lower layers to identify candidate neighbors. The size of the candidate pool explored during construction of the index is controlled by the parameter $ef_{\text{Construction}}$, which determines the breadth of exploration before selecting neighbors. A larger $ef_{\text{Construction}}$ leads to higher-quality graph construction and improved recall, albeit with increased insertion latency. From the candidate pool, a subset of neighbors is selected using pruning strategies that aim to preserve both proximity and structural diversity in the graph, ensuring efficient navigation. In this work, we adopt *α-RNG pruning* for neighbor selection, as detailed in Section 4.

At query time, the search process similarly maintains a dynamic candidate list whose size is governed by the parameter ef_{Search}. Increasing ef_{Search} allows the algorithm to explore a larger portion of the graph, improving recall at the expense of higher query latency.

Together, M, $ef_{\text{Construction}}$, and ef_{Search} define the core trade-offs between search accuracy, query latency, and memory footprint in HNSW-based systems. In particular, the combination of M and the exponential layer assignment distribution determines the structural topology of the graph, controlling the density of connections at each layer and the navigability of the index as a whole.

2.2. Challenges in Dynamic Updates

While HNSW is highly effective for static datasets, its design implicitly assumes that the underlying vector representations remain unchanged. However, real-world systems frequently encounter dynamic updates, including vector modifications, deletions, and insertions [13, 11].

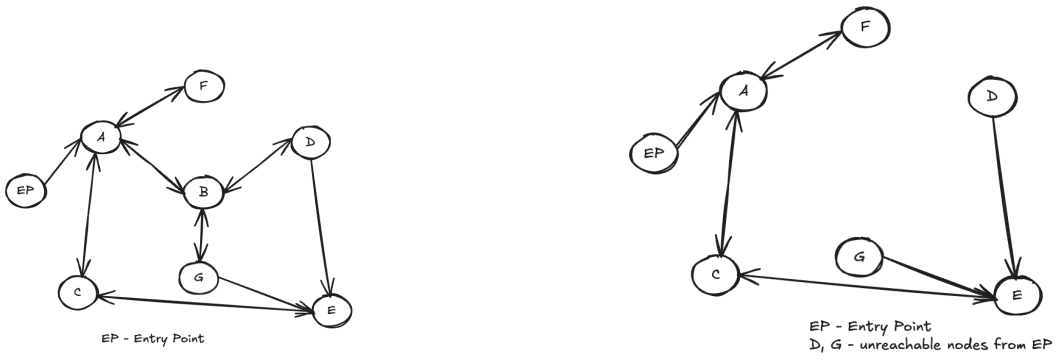
Handling updates in HNSW is non-trivial due to the tight coupling between node embeddings and their local graph structure. When a node is updated, its existing neighborhood may no longer reflect its nearest neighbors in the updated vector space. Naively modifying or removing such nodes can lead to inconsistencies in the graph. Two common approaches exist for handling updates in HNSW index [14]:

- **Delete + Insert (Tombstone approach):** The original node is marked as obsolete, and a new node is inserted with the updated vector [15]. While simple, this approach increases memory overhead and can degrade graph quality over time due to accumulation of stale nodes. Many production vector databases periodically rebuild the index to reclaim storage and evict stale nodes, increasing the operational cost of maintaining such indexes.
- **In-place Update with Neighborhood Repair:** The node is updated directly, and its neighborhood is repaired to reflect the new vector [13, 20, 21]. This approach is more memory-efficient but requires careful handling to maintain graph connectivity and search performance. In this work, we focus on neighborhood repair as the primary update strategy, analyzing its structural implications under high-frequency catalog update workloads.

2.3. Unreachable Nodes and Graph Fragmentation

A key failure mode in neighborhood updates is the emergence of unreachable nodes. When a node is deleted or updated without proper neighborhood repair, its removal may disconnect portions of the graph. Consider a node B that acts as a bridge between two regions of the graph, as shown in Figure 1a. If this node is removed or its connections are not adequately rewired, the graph may split into multiple disconnected components. As a result, nodes in one component become unreachable from the entry point EP located in another component. This phenomenon is illustrated in Figure 1b.

This fragmentation is particularly detrimental in HNSW, where search relies on greedy traversal from a fixed entry point. Once disconnected, nodes in isolated components cannot be discovered during



(a) HNSW Indexed Graph

(b) Node B deleted from the Index

Figure 1: Unreachable Nodes phenomenon in HNSW Indexed Graph

search, leading to a systematic drop in search recall and compromising the reliability of any retrieval system backed by that index.

2.4. Update Pipeline: Deletion, Repair, and Reinsertion

Update operations in HNSW are typically decomposed into three stages:

- **Deletion:** The node to be updated is removed or marked as deleted from the graph.
- **Neighborhood Repair:** The neighbors of the deleted node are reconnected to restore local graph structure. Several approaches have been proposed for this step, ranging from global multi-hop candidate expansion to local rewiring strategies.
- **Reinsertion:** The updated vector is inserted back into the graph using the standard HNSW insertion procedure.

The effectiveness of this pipeline depends heavily on the quality of the neighborhood repair step. In this work, we systematically analyze the impact of different neighborhood repair strategies under cumulative update workloads, focusing on the trade-offs between update latency, graph stability, and search performance. We compare existing approaches such as Multi-Neighbor Replace Update (MN-RU) [13] and the native HNSW repair mechanism [15], and introduce LR-RN, detailed in Section 4.

3. Related Work

Updates in graph-based ANN indexes have received growing attention. FreshDiskANN [20] introduced one of the first graph-based streaming indexes supporting real-time inserts and deletes, using a two-stage merge protocol. Wolverine [21] proposed monotonic search path repair by adding in-edges for out-neighbors of deleted nodes. SPFresh [22] took a cluster-based approach with lightweight incremental rebalancing for disk-based indexes. MN-RU [13] specifically worked with existing HNSW index approach and addressed the unreachable nodes phenomenon by restricting neighborhood repair to mutual neighbors, reducing candidate expansion relative to Native-RU. Motivated by MN-RU's observation that local repair is sufficient to preserve graph connectivity, LR-RN takes this further by restricting neighborhood reconstruction to a single representative node within the one-hop neighborhood, eliminating multi-hop expansion entirely and achieving lower complexity while producing fewer unreachable nodes under sustained catalog update workloads. As FreshDiskANN, Wolverine, and SPFresh operate on different underlying index architectures, direct empirical comparison requires non-trivial adaptation and is deferred to future work.

4. Methodology

Recent work has proposed neighborhood repair strategies such as 2-hop candidate expansion to preserve graph connectivity under dynamic updates. In this section, we formalize these approaches alongside our proposed method, LR-RN, and analyze their computational trade-offs.

4.1. Preliminaries

Let $G = (V, E)$ denote the HNSW graph, where each node $v \in V$ corresponds to a vector embedding. Each node maintains connections across multiple hierarchical layers $\ell \in [0, L]$, where layer 0 is the base layer.

- $\mathcal{N}_\ell(u)$: set of neighbors of node u at layer ℓ
- M: branching factor
- $L \sim \mathcal{O}(\log N)$: number of layers
- $ef_{\text{Construction}}$: candidate pool size during insertion
- ef_{Search}: candidate pool size during query time

Given an update to node u, the goal of neighborhood repair is to reconstruct local connectivity while preserving search efficiency and overall graph navigability.

4.2. Neighborhood Repair Strategies

4.2.1. Native Replace Update (RU)

The native HNSW update strategy [15, 10] constructs a candidate pool for repairing the neighborhood of an updated node by considering both one-hop neighbors $\mathcal{N}_\ell(u)$ and two-hop neighbors $\bigcup_{v \in \mathcal{N}_\ell(u)} \mathcal{N}_\ell(v)$. As illustrated in Figure 2, if node B is the node to be updated, the candidate pool consists of nodes $\{A, C, G, E, D, F\}$. While this approach aims to improve graph connectivity, it introduces significant computational overhead. The worst-case time complexity of the repair operation per layer is $\mathcal{O}(M^3)$, giving a total complexity of $\mathcal{O}(L \cdot M^3)$ across all layers.

This makes Native-RU expensive under heavy update workloads, as the candidate pool scales rapidly with the branching factor M.

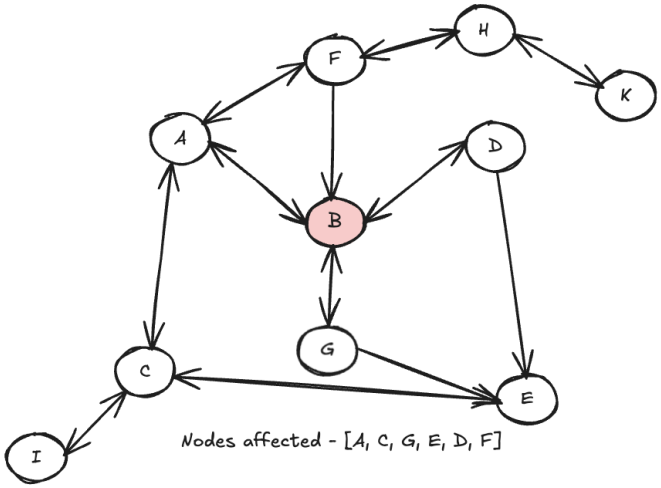


Figure 2: Native RU Graph: Updated Nodes

4.2.2. Multi-Neighbor Replace Update (MN-RU)

MN-RU [13] reduces the candidate space by restricting updates to a subset of neighbors that maintain direct connectivity with the updated node. Formally, let $N_1 = \mathcal{N}_\ell(u)$ and $N_2 \subseteq N_1$ be nodes that have a back-edge to u. For each $v \in N_2$, the candidate set is constructed as:

$$\mathcal{C}_v = \mathcal{N}_\ell(v) \cup N_1 \tag{1}$$

An example of this candidate pool and its neighborhood selection is shown in Figure 3. This reduces unnecessary expansion while preserving local structure. The resulting complexity per layer is $\mathcal{O}(M^2 \cdot d)$, giving a total complexity of $\mathcal{O}(L \cdot M^2 \cdot d)$ across all layers, where d denotes the average degree of nodes in N_2. MN-RU thus offers improved efficiency while maintaining reasonable graph quality.

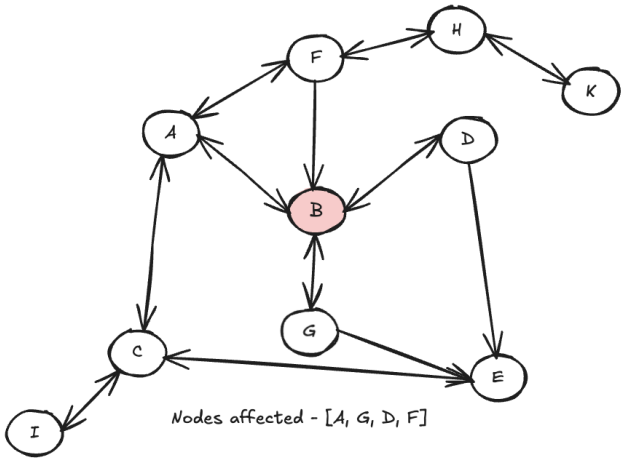


Figure 3: MN-RU Graph: Updated Nodes

4.2.3. Local Rewiring with Random Neighbor (LR-RN)

We propose LR-RN, a strictly local neighborhood repair strategy that avoids expensive multi-hop candidate expansion. LR-RN selects a node within the updated node's neighborhood as a representative and rewires all remaining neighborhood nodes through it, preserving local graph structure without expanding the candidate pool beyond the one-hop neighborhood. Rather than reconnecting all nodes in the neighborhood among each other, connectivity is restored by linking each neighborhood node to the selected representative. While this work employs uniform random selection for the representative node, the framework naturally extends to heuristic-driven selection strategies, such as choosing the neighbor with the highest degree or the closest embedding to the deleted node.

Key Principles:

- **Random Representative Selection:** A representative node $r \in \mathcal{N}_\ell(u)$ is selected uniformly at random.
- **Local Candidate Pool:** The repair is restricted to the one-hop neighborhood $\mathcal{C} = \mathcal{N}_\ell(u)$. No two-hop expansion is performed.
- **Local Rewiring:** For each node $v \in \mathcal{N}_\ell(u)$, edges are rewired using the representative r, ensuring connectivity within the local subgraph.

The representative node is selected from level-0 neighbors as the base layer contains the densest connectivity, making it the most structurally critical layer for preserving graph navigability.

Since LR-RN operates strictly within the one-hop neighborhood, the candidate size is $\mathcal{O}(M)$ and per-node pruning is $\mathcal{O}(M^2)$. Thus, the total complexity across layers is $\mathcal{O}\left(L \cdot M^2\right)$.

Representative Selection Sensitivity While this work employs uniform random selection for the representative node, we observe empirically that the structural impact of representative choice is bounded by the local graph topology. Since the pruning methodology enforces direction and magnitude diversity constraints on all rewired edges regardless of which node is selected as representative, the pruning step absorbs variation in representative quality. In practice, random selection produces stable and consistent unreachable node counts across repeated runs, suggesting that the repair outcome is not highly sensitive to the specific choice of representative within the one-hop neighborhood.

A detailed illustration of the LR-RN repair process is shown in Figure 4, and the algorithm pseudocode is documented in Algorithm 1.

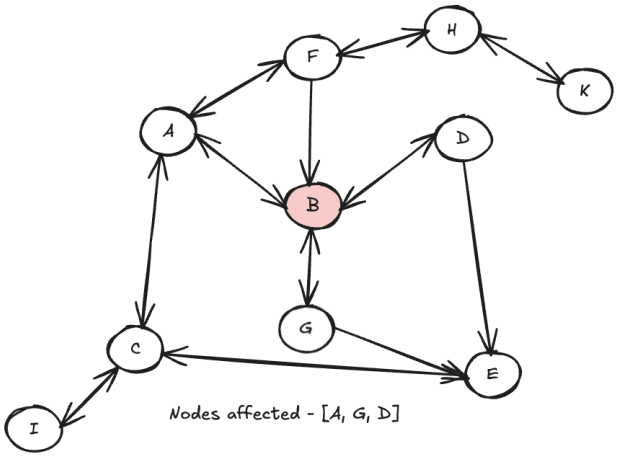


Figure 4: LR-RN Graph: Updated Nodes

Algorithm 1: LR-RN(u, G, M, α)

Input: Node u to delete, graph G, maximum degree M, pruning factor α
Output: Updated graph G with repaired local connectivity

Let $\ell_u \leftarrow$ level of node u;
Let $N_1 \leftarrow$ neighbors of u at level 0 (excluding deleted nodes);
Remove u from G;
if $N_1 = \emptyset$ **then**
    return G;
Randomly select representative node $r \in N_1$;
for $\ell \leftarrow \min(\ell_u, \mathit{level}(r))$ *down to* 0 **do**
    Let $S \leftarrow$ neighbors of u at level ℓ (excluding deleted nodes);
    foreach $v \in S$ **do**
        if $v = r$ **then**
            continue;
        if $r \notin \mathit{neighbors}(v, \ell)$ **then**
            $C \leftarrow \text{neighbors}(v, \ell) \cup \{r\}$;
            $C \leftarrow C \setminus \{u, v\}$;
            $R \leftarrow \text{PruneAlphaRNG}(v, C, M, \alpha)$;
            Update neighbors$(v, \ell) \leftarrow R$;
return G;

Unlike RU and MN-RU, LR-RN prioritizes locality over exhaustive candidate exploration. This significantly reduces update cost while preserving structural integrity for efficient search. Empirically, LR-RN avoids excessive graph rewiring, reduces unreachable node growth, and leads to improved index

stability under sustained update workloads.

4.3. Neighbor Selection and α-RNG Pruning

A critical component of the HNSW update pipeline is the selection and pruning of neighbors from a candidate pool. Traditional heuristics prioritize selecting the closest nodes while enforcing constraints that avoid redundant connections [10].

One such approach is α-relative neighborhood graph (α-RNG pruning), which introduces a geometric constraint to ensure diversity among selected neighbors [23]. A candidate node is accepted only if it is not "covered" by an already selected neighbor under the α-RNG condition. Intuitively, this prevents selecting multiple neighbors that lie in the same direction or cluster in the vector space.

By enforcing this constraint, α-RNG pruning improves the angular diversity of neighbors, leading to better graph navigability and robustness during search. This property becomes particularly important in dynamic settings, where maintaining connectivity and coverage of the vector space is critical. The pseudocode for the algorithm is shown in Algorithm 2.

Algorithm 2: PruneAlphaRNG(u, C, M, α)

Input: Source node u, candidate set C, maximum degree M, pruning factor α
Output: Pruned neighbor set R

```
Compute d(u, c) for each c ∈ C;
Sort C in ascending order of d(u, c);
R ← ∅;
foreach c ∈ C do
    keep ← true;
    foreach r ∈ R do
        if d(u, c) > α · d(c, r) then
            keep ← false;
            break;
    if keep then
        R ← R ∪ {c};
    if |R| = M then
        break;
return R;
```

For simplicity, we treat α as an optional configuration parameter. Table 1 summarizes the behavioral implications of different α settings. In this paper, we conduct our experiment with α set to 1.1, consistent with the balanced range identified in Table 1 and validated empirically via a small parameter sweep across $\alpha \in \{1.0, 1.1, 1.2, 1.3\}$ prior to the main evaluation. Our experiments show that $\alpha = 1.1$ produces the fewest unreachable nodes at index construction time across all datasets, with approximately 100 unreachable nodes observed per dataset before any updates are applied. At $\alpha < 1.1$, the number of unreachable nodes grows to the range of 10,000, while at $\alpha > 1.1$, unreachable node counts remain comparable to those at $\alpha = 1.1$ but index construction and search time increase significantly. We therefore retain $\alpha = 1.1$ as the primary experimental setting, as it represents the optimal trade-off between graph connectivity and computational overhead.

4.4. Reinsertion of the Updated Node

Following neighborhood repair, the updated vector is reinserted into the graph. Instead of performing full deletion and reinsertion, we reuse the existing node identifier and update its embedding in-place. The insertion follows the standard HNSW procedure:

- Greedy traversal from the entry point

Table 1
Implications of α parameter settings in HNSW graph construction

Setting	Implication
$\alpha < 1$	Over-pruned, lighter graph; very low latency but reduced recall
$\alpha = 1$	Strict setting; risks recall degradation and reduced graph navigability
$\alpha \in [1.1, 1.2]$	Balanced graph; optimal trade-off between performance and stability
$\alpha > 1.3$	Marginal improvement in graph diversity; larger navigability overhead

- Candidate selection using $ef_{\text{Construction}}$
- Neighbor pruning using heuristic or α-RNG

This ensures compatibility with the original graph construction process while enabling efficient updates.

5. Experiments and Results

5.1. Premise and Datasets

We benchmark our proposed method and the comparative methods on key performance metrics and graph stability. Our evaluation considers two categories of datasets: standard ANN benchmark datasets widely used in the literature, and a production-grade product catalog embedding dataset to emulate real-world e-commerce workloads. All experiments are conducted on a Google Cloud VM with 4 vCPUs, 32 GB memory, 64-bit CPU, in a Linux operating system. While shared virtualized infrastructure introduces potential cache interference between co-located workloads, production vector database deployments commonly operate on similar virtualized environments, making VM-based evaluation representative of real-world conditions. All runs are conducted over a single-threaded process and evaluation under concurrent multi-threaded update and search workloads, more representative of production conditions, is left for future work. The following metrics are evaluated in our analysis:

Update Time Time to perform neighborhood repair and update a node's vector in the index.

Search Time Time to retrieve top-k approximate nearest neighbors from the graph following updates.

Recall@10 Fraction of true top-10 nearest neighbors returned by ANN search, measuring index accuracy after cumulative updates.

Unreachable Nodes Number of nodes unreachable from the entry point following a cumulative set of updates.

Experiments are evaluated at cumulative update milestones of 10, 100, 1,000, and 10,000 nodes. For the 100K-scale datasets, 10,000 nodes represents approximately 10% of the dataset population. For the million-scale SIFT-1M dataset, updates are evaluated up to 100,000 nodes, also representing 10% of the dataset. Update time and search time are reported as the mean time per operation averaged across all updates at each milestone. For recall evaluation, we use the standard query set provided with SIFT-1M, consisting of 10,000 queries. For Deep-100K and Amazon-100K, we use a sample of 1,000 queries from the dataset. Ground truth nearest neighbors for all datasets are computed via exact brute-force search over the full index prior to any updates.

The ef values were tuned per dataset to achieve a stable baseline recall prior to updates, reflecting the varying geometric structure of each dataset.

Table 2
Dataset Summary

Dataset	Dimensions	Branching Factor (M)	ef (search, construct)
SIFT-1M	128	16	(400, 400)
Deep-100K	256	16	(600, 600)
Amazon-100K	384	16	(200, 200)

5.1.1. SIFT-1M

SIFT-1M [24] is a standard ANN benchmark consisting of 1 million 128-dimensional Scale-Invariant Feature Transform (SIFT) descriptors, chosen for its clustered geometric structure which provides a representative evaluation setting for graph-based ANN indexes.

5.1.2. Deep-100K

Deep-100K is a subset of the Deep1B dataset [25] consisting of 100,000 256-dimensional neural descriptors, chosen for its more uniform vector space distribution, providing a contrasting evaluation setting to SIFT-1M.

5.1.3. Amazon-100K

We construct a production-grade product catalog dataset of 100,000 products sourced from the Amazon Reviews 2023 metadata corpus [26]. Product embeddings are generated using the Sentence BERT model `all-MiniLM-L6-v2` [27], encoding product titles and descriptions into 384-dimensional vectors. We sample categories across Electronics (43.9 million ratings), Amazon Fashion (2.5 million ratings), and Home Appliances (2.1 million ratings), selected to represent diverse update workload characteristics spanning high-interaction products, fashion items with seasonal metadata churn, and sparse long-tail home products respectively. The category distribution of the dataset is shown in Figure 5 and summary statistics are provided in Table 3.

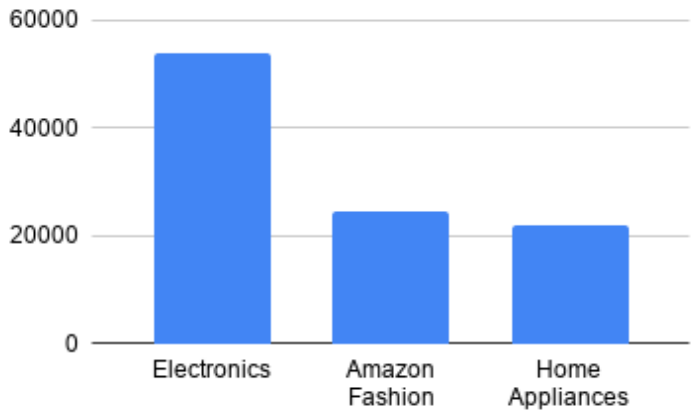


Figure 5: Amazon Product Catalog Category Distribution

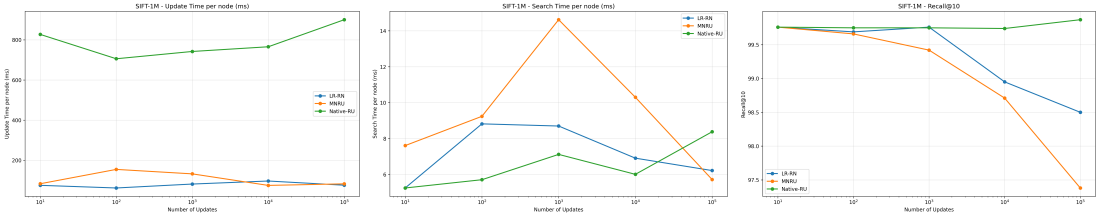
While Amazon-100K and Deep-100K operate at 100,000 vectors, SIFT-1M provides a million-scale evaluation setting. Extending the analysis to billion-scale product catalogs representative of large e-commerce platforms remains an avenue for future work.

For all experiments, we simulate a dynamic update workload by designating 10% of the dataset population for cumulative updates. Performance is measured progressively as nodes are updated, tracking metrics across the full update trajectory up to the 10% threshold. The index configurations used across all three datasets are summarised in Table 2.

Table 3
Amazon-100K Dataset Statistics

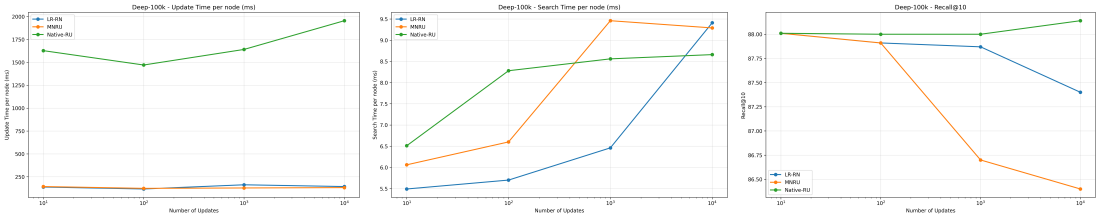
Property	Value
Total Products	100,000
Embedding Dimensions	384
Embedding Model	all-MiniLM-L6-v2
Number of Categories	3
Largest Category	Electronics
Avg. Description Length	~120 tokens

5.2. Performance Evaluation



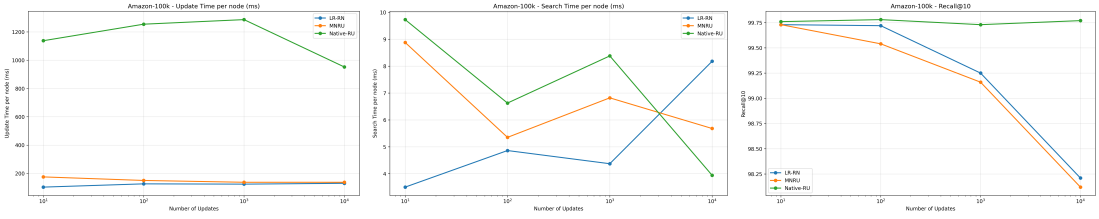
(a) SIFT-1M: Update Time (b) SIFT-1M: Search Time (c) SIFT-1M: Recall@10

Figure 6: SIFT-1M Performance Metrics



(a) Deep-100K: Update Time (b) Deep-100K: Search Time (c) Deep-100K: Recall@10

Figure 7: Deep-100K Performance Metrics



(a) Amazon-100K: Update Time (b) Amazon-100K: Search Time (c) Amazon-100K: Recall@10

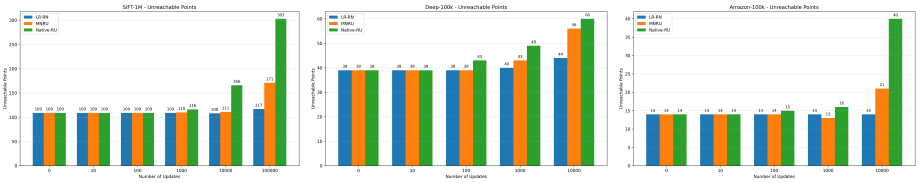
Figure 8: Amazon-100K Performance Metrics

Update Time. As opposed to the 2-hop neighborhood repair strategy employed by Native-RU, both MN-RU and LR-RN substantially outperform Native-RU on update time. This is consistent with our theoretical analysis of time complexity. Concretely, Native-RU requires roughly 10 times more time per node update compared to both MN-RU and LR-RN, as shown in Figures 6a, 7a and 8a. LR-RN achieves the lowest per-node update time across all three methods, owing to its repair strategy that restricts neighborhood reconstruction to only directly affected nodes.

Recall and Search Time. The primary objective of this evaluation is to demonstrate that cumulative updates do not cause catastrophic degradation of search quality across the index in terms of search time and recall. Our experiments demonstrate strong resilience across all three methods on both recall and search time over the full update trajectory. Recall remains bounded within 1–3% degradation over cumulative deletes across all datasets. Notably, search time does not worsen with cumulative updates for LR-RN and MN-RU. As shown in Figures 6b, 7b and 8b, it marginally improves or remains stable, likely due to sparser graph traversal paths in regions affected by deletions. Native-RU is an exception on Deep-100K, where search time shows a slight increase at higher update volumes, consistent with its heavier neighborhood repair causing greater structural disruption to traversal paths.

However, a critical nuance emerges from the recall results. Although Native-RU exhibits the least absolute recall degradation among the three methods, this observation is misleading. As we show in Section 5.3, Native-RU's apparent recall stability masks a silent and progressive accumulation of unreachable nodes. In the context of the Amazon-100K product catalog, this translates to products being structurally excluded from ANN traversal despite remaining in the index. This makes Native-RU's recall metric an unreliable indicator of true index health.

5.3. Unreachable Nodes Analysis



(a) SIFT-1M: Unreachable Nodes (b) Deep-100K: Unreachable Nodes (c) Amazon-100K: Unreachable Nodes

Figure 9: Unreachable Nodes Analysis

As established in our problem formulation, unreachable nodes represent a silent failure mode for dynamically updated HNSW indexes. A node becomes unreachable when all of its incoming edges are severed during neighborhood repair operations, making it permanently invisible to greedy graph traversal regardless of its semantic relevance to any query.

Figure 9 illustrates the growth of unreachable nodes as a function of cumulative updates across all three methods and all datasets. Native-RU exhibits a consistent and monotonically increasing accumulation of unreachable nodes across all settings. In image-based benchmark datasets, Native-RU produces approximately 3 times more unreachable nodes than LR-RN on the SIFT-1M dataset. The disparity is substantially more pronounced on the production-grade Amazon product catalog dataset, where Native-RU continues to accumulate unreachable nodes while LR-RN arrests their growth entirely, as shown in Figure 9c. MN-RU similarly produces unreachable nodes across all datasets, as shown in Figure 9, though at a lower rate than Native-RU.

This result has direct and serious implications for e-commerce retrieval. In a product catalog setting, this results in a significant and growing fraction of the catalog, including potentially high-relevance or newly introduced products, becoming permanently excluded from search results without any explicit system signal. We observe that unreachable nodes are disproportionately concentrated in the Electronics category, accounting for approximately 21 of the 26 additional unreachable nodes accumulated under Native-RU after cumulative updates on the Amazon-100K dataset. This concentration is consistent with Electronics being the largest and most densely connected category in the index, where aggressive 2-hop neighborhood repair is more likely to displace existing edges and sever connectivity. This is further illustrated in Table 4, which presents a representative sample of products excluded from the HNSW index following cumulative updates under Native-RU.LR-RN's ability to arrest unreachable node growth demonstrates that targeted local repair and rewiring is not only faster per update, but also

produces structurally healthier indexes under sustained catalog churn.

While this evaluation designates 10% of the dataset for cumulative updates, the monotonically increasing nature of Native-RU's unreachable node accumulation suggests the disparity would widen at higher update rates. Evaluating LR-RN under higher catalog churn rates representative of high-velocity e-commerce platforms remains an important direction for future work.

Table 4
Representative sample of products rendered unreachable in the Amazon-100K catalog index under Native-RU after cumulative updates.

Product Title	Category
ASUS ROG Swift 27" 1440P Gaming Monitor (PG279Q) — QHD, IPS, 165Hz, G-SYNC	Computers
LG 55IN 1920x1080 (FHD) Class LED Commercial Widescreen TV	All Electronics
Godly Living Silicone Wedding Ring — Multiple Colors, Sizes 5–12	Amazon Fashion
Whitmor Zippered Underbed Storage Bag, Jumbo	Amazon Home

6. Conclusion

In this paper, we presented a comprehensive analysis of the structural impact of continuous product catalog updates on HNSW-based vector indexes. We demonstrated that under sustained update workloads, the Native-RU strategy silently accumulates unreachable nodes, a critical failure point that compromises the reliability and catalog coverage of production semantic search systems without surfacing any explicit signal to the systems or users that depend on them.

Our proposed algorithm, LR-RN, outperforms Native-RU and MN-RU across all benchmark datasets on unreachable node growth. The core hypothesis behind LR-RN, that a strictly local one-hop repair is sufficient to preserve the global small-world navigability properties of the HNSW graph, is strongly supported by our experimental results. On standard ANN benchmarks, Native-RU accumulates approximately 3× more unreachable nodes than LR-RN. On the Amazon product catalog dataset, LR-RN effectively arrests unreachable node growth entirely, while Native-RU continues to accumulate them at an order of magnitude higher rate. LR-RN simultaneously maintains recall degradation within 1–3% across all datasets and update trajectories, preserves search time, and achieves per-node update times approximately 10× lower than Native-RU.

These results validate that locality-preserving neighborhood repair is not only sufficient but preferable over exhaustive multi-hop candidate expansion when working with dynamically updated HNSW indexes. Critically, our findings challenge the 2-hop neighbor selection strategy employed in hnswlib's default update implementation, showing it to be both computationally expensive and structurally harmful under high-frequency catalog updates.

Future Work. Looking ahead, we intend to study the behavior of LR-RN under simultaneous insert, search, and update operations within the index, bringing the analysis closer to real-world production conditions and providing stronger evidence for its suitability as a drop-in repair strategy in live e-commerce retrieval systems. While we compare against Native-RU and MN-RU, a direct empirical comparison with FreshDiskANN and Wolverine remains an important direction for future work, as these methods operate on different index architectures making direct comparison non-trivial. Additionally, extending evaluation to billion-scale product catalogs representative of large e-commerce platforms remains an important direction as noted in 5.1.

Declaration on Generative AI

During the preparation of this work, the author used Claude (Anthropic) for grammar and spelling checking, language refinement, literature search, and code debugging assistance (similar to consulting Stack Overflow). After using these tools, the author reviewed and made appropriate edits to the content as needed and takes full responsibility for the publication's content. No Generative AI tools were used for generating research ideas, experimental design, data analysis, or substantive content.

References

[1] P. Nigam, Y. Song, V. Mohan, V. Lakshman, W. Ding, A. Shingavi, C. H. Teo, H. Gu, B. Yin, Semantic product search, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2019, pp. 2876–2885.

[2] H. Zhang, S. Wang, K. Zhang, Z. Tang, Y. Jiang, Y. Xiao, W. Yan, W.-Y. Yang, Towards personalized and semantic retrieval: An end-to-end solution for e-commerce search via embedding learning, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2020, pp. 2407–2416.

[3] S. Li, F. Lv, T. Jin, G. Lin, K. Yang, X. Zeng, X.-M. Wu, Q. Ma, Embedding-based product retrieval in taobao search, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2021, pp. 3181–3189.

[4] Y. Xie, T. Na, X. Xiao, S. Manchanda, Y. Rao, Z. Xu, G. Shu, E. Vasiete, T. Tenneti, H. Wang, An embedding-based grocery search model at instacart, arXiv preprint arXiv:2209.05555 (2022).

[5] T. Na, Z. Xu, H. Wang, Rethinking e-commerce search, arXiv preprint arXiv:2312.03217 (2023).

[6] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, et al., Milvus: A purpose-built vector data management system, in: Proceedings of the 2021 International Conference on Management of Data, ACM, 2021, pp. 2614–2627.

[7] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, H. Jégou, The FAISS library, arXiv preprint arXiv:2401.08281 (2024).

[8] B. Dilocker, et al., Weaviate: An open-source vector database, arXiv preprint arXiv:2301.02004 (2023).

[9] J. Sun, et al., GaussDB-Vector: A large-scale persistent real-time vector database, Proceedings of the VLDB Endowment 18 (2025). URL: https://www.vldb.org/pvldb/vol18/p4951-sun.pdf.

[10] Y. A. Malkov, 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 (2020) 824–836. doi:10.1109/TPAMI.2018.2889473.

[11] A. Magnani, F. Liu, S. Chaidaroon, S. Yadav, P. R. Suram, A. Puthenputhussery, S. Chen, M. Xie, A. Kashi, T. Lee, C. Liao, Semantic retrieval at walmart, in: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2022, pp. 3495–3503.

[12] A. Kekuda, Y. Zhang, A. Udayashankar, Embedding based retrieval for long tail search queries in ecommerce, in: Proceedings of the 18th ACM Conference on Recommender Systems, ACM, 2024, pp. 771–774.

[13] W. Xiao, Y. Hu, Y. Wang, W. Wang, Enhancing hnsw index for real-time updates: Addressing unreachable points and performance degradation, arXiv preprint arXiv:2407.07871 (2024).

[14] Y. Sato, Y. Matsui, How should we evaluate data deletion in graph-based ANN indexes?, in: Machine Learning for Systems Workshop at Neural Information Processing Systems (NeurIPS), 2025. URL: https://openreview.net/pdf?id=lnaC19Pd30.

[15] Y. A. Malkov, D. A. Yashunin, hnswlib — fast approximate nearest neighbor search library, https://github.com/nmslib/hnswlib, 2019. GitHub repository.

[16] P. Covington, J. Adams, E. Sargin, Deep neural networks for youtube recommendations, in: Proceedings of the 10th ACM Conference on Recommender Systems, ACM, 2016, pp. 191–198. doi:10.1145/2959100.2959190.

[17] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W.-t. Yih, Dense passage retrieval for open-domain question answering, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 6769–6781. doi:10.18653/v1/2020.emnlp-main.550.

[18] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Advances in Neural Information Processing Systems (NeurIPS), volume 33, 2020, pp. 9459–9474.

[19] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, X. Lin, Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement, IEEE Transactions on Knowledge and Data Engineering 32 (2020) 1475–1488. doi:10.1109/TKDE.2019.2909204.

[20] A. Singh, S. J. Subramanya, R. Krishnaswamy, H. V. Simhadri, FreshDiskANN: A fast and accurate graph-based ANN index for streaming similarity search, arXiv preprint arXiv:2105.09613 (2021).

[21] W. Zheng, et al., Wolverine: Highly efficient monotonic search path repair for dynamic approximate nearest neighbor search, Proceedings of the VLDB Endowment 18 (2025). doi:10.14778/3709649.3709912.

[22] Y. Xu, H. Liang, J. Li, S. Xu, Q. Chen, Q. Zhang, C. Li, Z. Yang, F. Yang, Y. Yang, P. Cheng, M. Yang, SPFresh: Incremental in-place update for billion-scale vector search, in: Proceedings of the 29th Symposium on Operating Systems Principles (SOSP), ACM, 2023, pp. 545–561.

[23] C. Fu, C. Xiang, C. Wang, D. Cai, Fast approximate nearest neighbor search with the navigating spreading-out graph, Proceedings of the VLDB Endowment 12 (2019) 461–474. doi:10.14778/3303753.3303754.

[24] D. G. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision 60 (2004) 91–110. doi:10.1023/B:VISI.0000029664.99615.94.

[25] A. Babenko, V. Lempitsky, Efficient indexing of billion-scale datasets of deep descriptors, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 2055–2063.

[26] Y. Hou, J. Li, Z. He, A. Yan, X. Chen, J. McAuley, Bridging language and items for retrieval and recommendation, arXiv preprint arXiv:2403.03952 (2024).

[27] N. Reimers, I. Gurevych, Sentence-BERT: Sentence embeddings using siamese BERT-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019, pp. 3982–3992. doi:10.18653/v1/D19-1410.