arXiv:2411.00970v1 [cs.DB] 1 Nov 2024

Incremental IVF Index Maintenance for Streaming Vector Search

Jason Mohoney
University of Wisconsin-Madison

Anil Pacaci
Apple

Shihabur Rahman Chowdhury
Apple

Umar Farooq Minhas
Apple

Jeffrey Pound
Apple

Cedric Renggli
Apple

Nima Reyhani
Apple

Ihab F. Ilyas
Apple

Theodoros Rekatsinas
Apple

Shivaram Venkataraman
University of Wisconsin-Madison

ABSTRACT

The prevalence of vector similarity search in modern machine learning applications and the continuously changing nature of data processed by these applications necessitate efficient and effective index maintenance techniques for vector search indexes. Designed primarily for static workloads, existing vector search indexes degrade in search quality and performance as the underlying data is updated unless costly index reconstruction is performed. To address this, we introduce Ada-IVF, an incremental indexing methodology for Inverted File (IVF) indexes. Ada-IVF consists of 1) an adaptive maintenance policy that decides which index partitions are problematic for performance and should be repartitioned and 2) a local re-clustering mechanism that determines how to repartition them. Compared with state-of-the-art dynamic IVF index maintenance strategies, Ada-IVF achieves an average of 2× and up to 5× higher update throughput across a range of benchmark workloads.

1 INTRODUCTION

Modern machine learning applications increasingly rely on high-dimensional vector embeddings to transform complex data such as images, text, or entities in knowledge graphs to vector representations that retain semantically meaningful information [19, 20, 27, 35, 48]. Vector similarity search is a critical component in such applications as it enables more accurate and contextualized search [28, 33, 34, 56] and recommendations [46, 47, 51, 53, 62] over multi-modal data. Partitioned indexes for vector similarity search have recently gained widespread adoption in these modern machine learning applications due to their performance and scalability [29, 38, 49, 66, 69, 71]. A common type of partitioned index is the Inverted File (IVF) index, where a vector quantization algorithm (typically k-means) is used to partition a vector dataset, and the resulting clusters constitute the partitions of the index [40]. While existing IVF index implementations are designed for static workloads, i.e., the partitioning is performed based on the initial state of the underlying vector dataset, real-world deployments require high-throughput *search* and *updates* over *dynamic vector data* where the underlying vector dataset is continuously modified through insertions and deletions [15, 58, 69]. In addition to being robust against modifications to the vector dataset, the indexes must be robust against changing query patterns over time, as observed in [15]. In this work, we study the effect of updates on the IVF index's search and update throughput and propose an incremental maintenance methodology for IVF indexes.

IVF indexes out-of-the-box do not have the notion of inserting new vectors or deleting existing vectors once constructed. Indeed, the most common method used by practitioners today is to rebuild the index from scratch to reflect any updates that have accumulated over time. However, depending on the scale of the vector dataset and the volume and frequency of updates, a full index rebuild can be prohibitively expensive. For example, it takes multiple days to rebuild an IVF index from scratch for billion-scale vector datasets [21, 69], making it necessary to revisit how updates can be reflected. Devising such an update mechanism consists of re-adjusting the partitioning of the high-dimensional space defined by the clusters and ensuring that the re-adjusted partitioning: (i) keeps the reconstruction error, *i.e.*, the average distance between a vector and its nearest cluster centroid, at minimum, otherwise, queries would require scanning more partitions to reach a target recall and degrades search throughput; (ii) does not create an imbalance in the size distribution of the partitions, which can result in variable search latency across queries [15, 39, 69].

In industrial vector search workloads, we observe that the partition access patterns of search and update operations over an IVF index are non-uniform and change over time. For instance, in a typical day of a KG entity search workload we studied [36, 49], we find that only 15% of partitions were accessed during search operations, and 80% of the updates affected partitions that were not accessed by any search operation. Such skewed access patterns induced by real-world vector search workloads present an opportunity for efficient and effective maintenance of IVF indexes over dynamic datasets. Specifically, index maintenance overhead can be minimized by devising a local, incremental indexing strategy and focusing the maintenance process on frequently accessed partitions during search. To the best of our knowledge, no existing approaches in the literature utilize partition access patterns for IVF index maintenance.

To this end, we propose Ada-IVF, an incremental maintenance mechanism for IVF indexes. Our approach is based on (i) the observation that reconstruction error and partition imbalance serve as indicators of an IVF index's search quality and performance (Section 2.2), and (ii) workload patterns can be utilized for local, incremental maintenance of its underlying partitions (Section 2.3). Ada-IVF consists of a workload-adaptive *policy* that identifies which partitions should be reindexed based on real-time statistics in order to minimize reconstruction error and partition imbalance and a *mechanism* that performs local reindexing over the target subset of partitions. Our policy uses *global* and *partition-local* indicator functions that quantify changes in reconstruction error and partition imbalance. In addition, Ada-IVF tracks read frequencies for individual partitions to quantify a *temperature* of each partition, which is then used by the local indicator function. Using temperature allows Ada-IVF to prioritize the maintenance of commonly accessed partitions and to avoid unnecessary work on rarely accessed partitions. Through its incremental maintenance policy and local reindexing mechanism, Ada-IVF effectively mitigates IVF index performance degradation due to updates.

Our experimental analysis confirms the efficacy of our approach. We conduct a comparison of Ada-IVF against the state-of-the-art LIRE [69] and other baseline IVF maintenance methodologies on synthetic data and public benchmarks. We show that Ada-IVF obtains similar or better search throughput when compared to baseline methods with an average of 2× improvement in throughput across all workloads and a maximum of 5× update throughput improvement on synthetic data. Furthermore, we verify Ada-IVF's robustness across various workload patterns using traces of industrial workloads and public and synthetic benchmarks, thereby establishing its applicability in diverse real-world scenarios.

In summary, our contributions include:

- Global and local indicator functions to quantify the impact of updates on global index and individual partition levels.
- A workload-adaptive policy that uses real-time statistics and these indicator functions to monitor reconstruction error and partition imbalance and decide when to perform reindexing.
- A local reindexing mechanism that uses k-means to split and reindex candidates and their neighboring partitions.
- A comprehensive empirical evaluation demonstrating the impact of updates on IVF indexes under a diverse array of real and synthetic workloads.

In Section 2, we discuss the preliminaries behind our study, detailing existing index solutions and systems as well as observations on real workloads. Section 3 provides an overview of the design of Ada-IVF. Section 4 details our contributions to IVF index quality maintenance. Section 5 presents our experimental results on internal industrial workloads and public and synthetic benchmarks, with a suite of microbenchmarks to validate our contributions. Section 6 concludes with a discussion of our findings and related work.

2 PRELIMINARIES & MOTIVATION

We now define the streaming vector search problem. We also provide a brief overview of IVF indexes, their use of vector search, and existing methods for supporting updates. Finally, we discuss our observations on the characteristics of streaming vector search workloads and their impacts on the IVF index performance, which motivate the incremental indexing strategy introduced in this paper.

2.1 Vector Search & Data Updates

Streaming vector search [9, 58] is the process of finding the top-k nearest neighbors of a d-dimensional query vector q in an evolving set of d-dimensional vectors $\mathbf{X}$. We consider the following operations over the set of vectors $\mathbf{X}$:

- **Insertion**: Adding a new vector x to the set $\mathbf{X}$, making $\mathbf{X} = \mathbf{X} \cup \{x\}$.
- **Deletion**: Removing an existing vector $x \in \mathbf{X}$ from the set, resulting in $= \mathbf{X} \setminus \{x\}$.
- **Search**: Find the top-k nearest neighbors to q in $\mathbf{X}$ according to a similarity metric.

Obtaining the exact top-k nearest neighbors of a query vector q in $\mathbf{X}$ is prohibitively expensive if $|\mathbf{X}|$ is large. In practice, approximate nearest neighbor (ANN) search methods that leverage indexes are used, providing a trade-off between accuracy and performance. Search accuracy is most commonly measured using *recall*, defined as $r = |\mathbf{G} \cap \mathbf{R}|/k$, where $\mathbf{R}$ is the set of ids returned from approximate search and $\mathbf{G}$ is the set of ground truth ids obtained by a linear scan. Performance, on the other hand, is most generally measured as the number of queries processed per second (QPS). Indexes for ANN search need to be updated otherwise performance can deteriorate with respect to a fixed recall target.

A streaming vector search workload is an ordered set of search and update operations over the set of vectors $\mathbf{X}$. As defined above, search operations correspond to finding k-nearest neighbors of a query vector over the current state of $\mathbf{X}$, and update operations modify set $\mathbf{X}$. These operations can be batched, searching, or updating multiple vectors at a time.

2.2 IVF Indexes

An IVF index is a partitioned data structure for high-dimensional vector search that consists of n_c partitions (clusters) and their representative vectors (centroids). An IVF index over a vector dataset $\mathbf{X}$ is constructed offline using a vector quantizer, typically a variant of k-means [21, 39]; consequently, the index construction process requires a global view of the underlying vector dataset $\mathbf{X}$. Clustering-based partitioning of the dataset enables similar vectors (based on the embedding distance) to be assigned to the same partition. At search time, such partitioning is utilized to reduce the number of distance computations performed; the distance from the query vector q to each cluster centroid is computed to identify n_p nearest partitions, and each of the p partitions is scanned to obtain the approximate k-nearest neighbors. By controlling the subset partitions scanned for each query, n_p provides a way to navigate the trade-off between quality and performance.

The primary indicators of IVF index search performance are i) *partition imbalance* and ii) *reconstruction error*. Partition imbalance refers to the uneven size distribution of partitions, and it adversely affects the search throughput as queries will take longer to scan overpopulated partitions. A common strategy to mitigate partition imbalance during index construction is to use balanced k-means, which can produce roughly equal-sized partitions so that latency is consistent across queries for a given n_p [69]. The reconstruction error is the average distance from each vector to its nearest centroid, and the objective function of the quantization algorithm minimizes the error. Numerous works have been proposed to minimize the reconstruction error of static vector search indexes [13, 25, 67]. Indexes with a larger reconstruction error have less compact clusters and require queries to scan more partitions to find their nearest

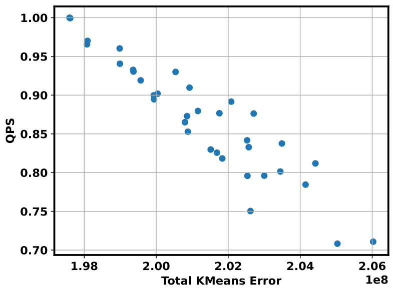


Figure 1: Static IVF indexes trained with balanced k-means on SIFT1M. Each point is a different initialization or number of iterations for k-means. As error increases, QPS degrades for recall@0.9

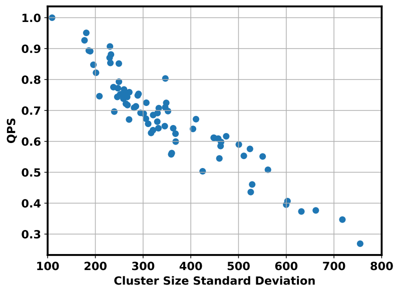


Figure 2: Evaluation of a *Frozen* IVF index on a SIFT1M dynamic workload with insert/delete ratio = 1. As partition imbalance increases, read throughput degrades for recall@0.9

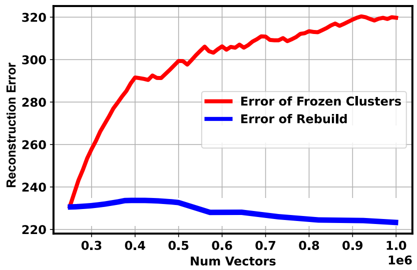


Figure 3: Evaluation of an IVF index reconstruction error as vectors from SIFT1M are inserted into the index. The error grows away from the error observed from fully rebuilding the index.

neighbors, degrading search performance. Figure 1 measures the search throughput vs. reconstruction error (k-means error) on the SIFT1M dataset for IVF indexes trained using various configurations of balanced k-means with a fixed number of centroids (n_c=1000). We observe degradation in performance as the error increases. Thus, partition imbalance and reconstruction error must be minimized to optimize search performance in IVF indexes.

Naively modifying partitions of the IVF index via partition append and deletes can support update operations in IVF indexes. However, as the clustering of vectors deviates from its initial state, such modifications can lead to an increase in partition imbalance and reconstruction error, which results in degradation in search quality and performance (See Section 2.3). To ensure robust search performance in the face of updates, it is necessary to address the increase in partition imbalance and reconstruction error. Below are previously proposed maintenance strategies for supporting updates in IVF indexes.

- **Rebuild** [66]: Periodically rebuilds the index from scratch.
- **Frozen** [1]: Partitions are modified without changing the centroids or number of partitions.
- **Update Centroids** [11]: Similar to *Frozen*, but centroids are updated upon partition modification to reflect the true mean of the partition.
- **DeDrift** [15]: Similar to *Update Centroids*, but for each update, the $k_1 << n_c$ largest and smallest partitions are collected and reclustered using k-means, keeping the total number of partitions fixed.
- **LIRE** [69]: Uses splitting/merging of partitions that violate pre-defined size thresholds. The contents of violating partitions with r_c neighboring partitions are reassigned to minimize reconstruction error. Neighboring partitions are determined by centroid distance to the violating partition.

Each IVF index maintenance approach presents a trade-off between the maintenance overhead (efficiency) and the ability to address partition imbalance and reconstruction error (effectiveness). While *Rebuild* is highly effective, i.e., it can minimize both imbalance and error by building the IVF index from scratch over the latest state of the dataset. Yet, it is inefficient and has the highest maintenance overhead. Although it can be acceptable for applications with static or slowly changing vector datasets, *Rebuild* is prohibitively expensive for large-scale vector datasets used in real-world applications, taking days for billion-scale datasets [69].

Frozen, at the other end of the efficiency vs effectiveness tradeoff, does not incur any maintenance overhead. However, it is only effective when the distribution of vectors and queries is uniform. As described in the next section in detail, *Frozen* does not address the increasing partition imbalance or reconstruction error for real-world workloads with skewed, non-uniform vector and query distributions. *Update Centroids* incurs the cost of modifying centroids with each update, which mitigates increasing reconstruction error but does not address imbalance. *DeDrift* not only updates centroids but also pays the cost of reclustering a subset of the partitions for each update operation, effectively addressing both imbalance and reconstruction error. *LIRE*, the state-of-the-art incremental IVF indexing solution, focuses on addressing imbalance, and its overhead mainly stems from the reassignment phase, which varies with the number of violating partitions. Overall, these existing update methods for IVF indexes do not consider the trade-off between maintenance overhead and search performance (i.e., partition imbalance and reconstruction error) in a holistic manner.

2.3 Impact of Updates on IVF Indexes

Here, we motivate Ada-IVF and its incremental maintenance methodology by discussing the characteristics of vector search workloads and the impact of updates on existing IVF index maintenance strategies. First, we show how partition imbalance and reconstruction error are impacted by simple baseline strategies such as *Frozen*, which does not change the clustering. Second, we measure partition access patterns of an industrial search workload to show the skew in read and write access patterns of real-world workloads. Finally, we show how the state-of-the-art incremental IVF indexing approach, *LIRE*, incurs unnecessary maintenance overhead by over-triggering reindexing for rarely accessed partitions.

Figure 2 demonstrates the relationship between the search performance and partition imbalance by showing the query throughput

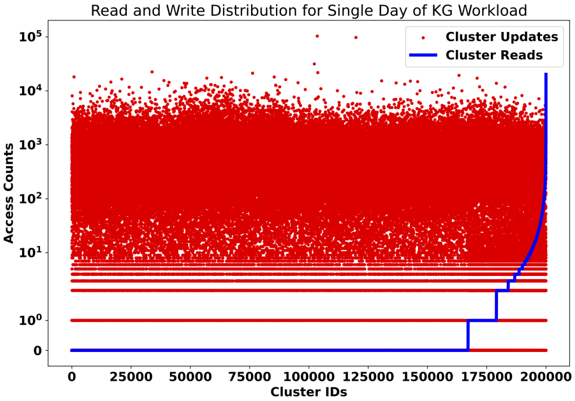


Figure 4: Read and write access patterns of partitions in an internal entity search workload. Partition IDs are ordered by their read count. The read distribution is skewed and uncorrelated with the write distribution, showing that many partitions are modified but not read from.

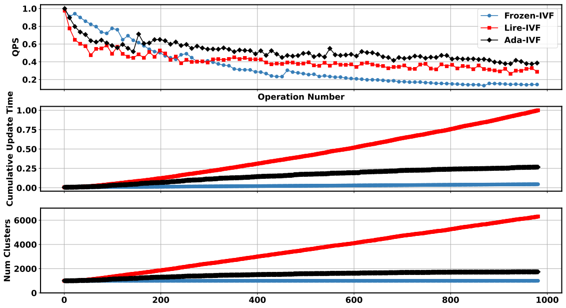


Figure 5: Evaluation of LIRE and Ada-IVF for a dynamic MSTuring10m workload where queries are localized to a few partitions. LIRE and our locality-aware approach Ada-IVF achieve similar QPS, but LIRE creates 3× as many partitions and requires 4× the update time due to its maintenance of partitions that are not accessed by the queries.

and the standard deviation of partition sizes over different snapshots of an IVF index updated via a sequence of insertions and deletions from the SIFT1M dataset. As vectors are added to and deleted from the index, partitions grow imbalanced, as measured by the increasing standard deviation of partition sizes. As the imbalance grows, there is a clear correlation to degrading search performance as more vectors are scanned to achieve the recall target. We observe in Figure 3 that the reconstruction error increases with updates on a similar insert-only workload. The error for *Frozen* deviates from the error observed for *Rebuild* as the number of vectors added to the index increases. Increasing reconstruction error results in degrading search performance (Figure 1). Our measurements demonstrate that IVF indexes exhibit increasing imbalance and reconstruction error due to updates, necessitating a maintenance strategy.

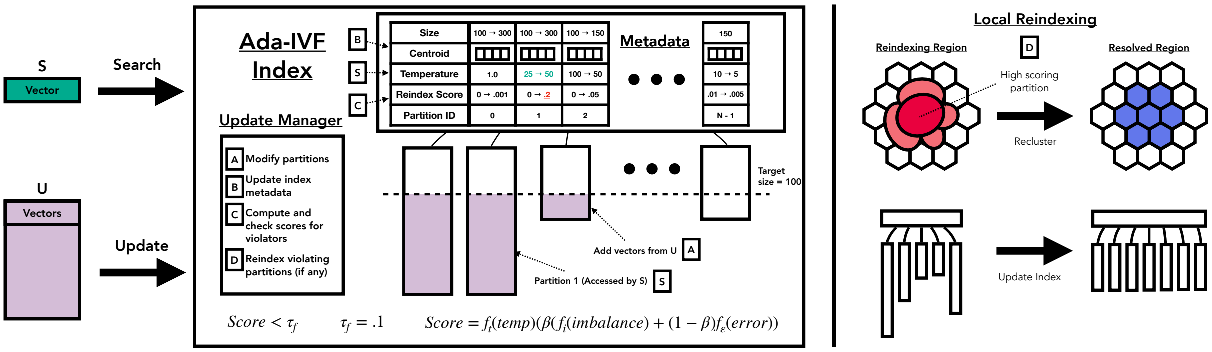
Next, we examine an industrial search workload over knowledge graph embeddings [36] and find that partitions are accessed non-uniformly by search and update operations, exhibiting skewed partition access locality. In Figure 4, we plot the total number of reads and writes for each partition over an IVF index for a single day of the workload. We see that 15% of partitions are accessed for search operations in a given day, with only 3% accessed by more than ten queries. We also see for this workload that the search and update operations access different sets of partitions, with 80% of updates affecting partitions that have not been accessed by search operations and 96% affecting partitions that are accessed by fewer than ten search queries. As we describe in the following sections in detail, these locality patterns provide an optimization opportunity for efficient and effective incremental update of IVF indexes compared to existing IVF update methodologies.

Finally, we observe that existing IVF index maintenance methods do not account for partition read locality. This results in DeDrift and LIRE over-triggering index maintenance for partitions that are rarely accessed. In Figure 5, we compare the update performance of LIRE against our locality-aware approach using a workload where queries are localized to a small number of partitions and updates modify all partitions uniformly. While both approaches can achieve similar search QPS, LIRE spends 4× more time processing updates. This is because LIRE eagerly reindexes partitions that exceed the pre-defined size threshold regardless of whether recent queries access them, while our proposed approach lazily reindexes partitions as queries access them.

3 ADA-IVF OVERVIEW

We introduce Ada-IVF, an incremental maintenance solution for IVF indexes. Figure 6 depicts the high-level overview of our incremental reindexing solution. In brief, Ada-IVF maintains robust search performance in the face of updates by selectively reindexing a subset of partitions that negatively contribute to the partition imbalance and reconstruction error, proxies for index performance, as discussed in the previous section. Ada-IVF has the following steps:

- Step 1: Maintain index metadata. Ada-IVF maintains the size, centroid, and *read temperature* of partitions. Where the temperature denotes the frequency and recency of reads for a given partition. Search operations modify the temperature (**S**), and update operations modify the partition contents, size, and centroid (**U, A, B**).
- Step 2: Monitor index partitions for increases in imbalance and reconstruction error. Partitions are assigned a *reindexing score* (**C**), which is a function of imbalance and reconstruction error, a higher score indicating higher imbalance/error. Partitions exceeding a score threshold of τ_f are *violators* (e.g., Partition 1).
- Step 3: Reindex violating partitions using local reindexing. Violating partitions are split and merged with neighboring partitions (**D**) to minimize imbalance and reconstruction error.



Size	100 → 300	100 → 300	100 → 150	150
Centroid				
Temperature	1.0	25 → 50	100 → 50	10 → 5
Reindex Score	0 → .001	0 → .2	0 → .05	.01 → .005
Partition ID	0	1	2	N - 1

Figure 6: Ada-IVF Overview and Example. Search and update operations modify the partitions (A) and metadata (S, B). For example, S increases the temperature of partition 1 from 25 to 50, and U increases the size from 100 to 300. After each update, a reindexing score is computed for modified partitions (C), and if the score violates a threshold, the partition is selected for reindexing (D).

Tracking Index Metadata. Ada-IVF tracks each partition's size, initial and current centroid vector, read temperature, and reindexing score as index metadata. The partition size determines imbalance by checking the deviation from a target partition size. The change in the centroid from its initial state signals an increase in reconstruction error. The read *temperature* is a float that captures how often a partition is accessed by a search operation. The *reindexing score* is a float indicating whether a partition should be reindexed. The total memory overhead is the index metadata stored by Ada-IVF is $(2 \times n_c \times d + 3) \times 4$ bytes for an index with n_c partitions, and d is the vector dimension.

It is important that the index metadata is maintained in a lightweight process to avoid slowdowns in search and update operations. Consider the effect of a search operation S followed by an update U on the Ada-IVF index in Figure 6. The search operation S modifies the temperature for each partition, which is a small overhead compared to executing the search operation. The temperature is increased for partitions accessed by S and decreased for all non-accessed partitions. For example, partition 1 is accessed by S, and its temperature increases from 25 to 50 while the other partitions decrease correspondingly. The update operation U modifies the size, centroid, and reindexing score of a set of partitions (0, 1, 2 in the example). Updating the size is made inexpensive by maintaining a counter for each partition. The centroids of modified partitions are updated incrementally rather than computing the mean over all vectors in the partition. Then, the reindexing score is computed using the new size and centroid of the modified partitions. We observed that the metadata tracking process can take up to 20% of the maintenance time of Ada-IVF, where the majority of the time is due to updating centroids. The metadata update process is detailed in 4.1.

Update Manager. Ada-IVF's update manager (Algorithm 1) consists of a reindexing policy that identifies which and when partitions should be reindexed and a mechanism for reindexing them.

The reindexing policy is informed by two indicator functions: *local* and *global* indicator functions, where both functions are evaluated after each update. The *local* indicator function $f(c)$ computes a reindexing score indicating if a specific partition c is imbalanced or has drifted from its initial state and is described in Section 4.2. If the indicator function exceeds a given threshold, the update manager identifies the partition as a violator and a candidate for reindexing. For example, partition 1 has a score of .2, which exceeds the threshold of .1, and therefore the partition is selected for reindexing. The *global* indicator function $G(I)$ acts as a fail-safe procedure to ensure that our local reindexing actions don't degrade imbalance or reconstruction error globally. The entire index is rebuilt if the *global* indicator exceeds a threshold. The global indicator function is described in 4.3. Indicator function values are checked after a batch of updates are issued to the index.

The local reindexing mechanism resolves quality violations by splitting violating partitions and merging with neighboring partitions using a variant of the original clustering algorithm. The violating partition is split using balanced k-means to meet the target partition size, e.g., partition 1 in Figure 6 is split into three partitions of size 100. Then, the merging phase balances the selected partitions and minimizes the error introduced by the splitting phase. Here, the nearest r_c partitions to the violator are selected based on centroid distance. The split partitions and neighboring partitions are then iteratively refined using a clustering algorithm. r_c enables Ada-IVF to control the trade-off between reindexing cost and resulting index quality, a larger value of r_c selects a larger region and incurs a more expensive reindexing cost but improves the resulting index quality. The reindexing mechanism and the impact of r_c is further detailed in Section 4.2.2.

4 INDEX QUALITY MAINTENANCE

This section describes the details of Ada-IVF's incremental index maintenance. Beginning with the update rules for index metadata, covering local reindexing, and concluding with global reindexing.

4.1 Tracking Index Properties

To identify the subset of partitions that cause degradation in query throughput, it is necessary to track how the underlying clustering properties evolve over time. For this purpose, we maintain the set of vectors that are present in each partition ($\mathbf{X_i}$) and the corresponding mean of the vectors (μ_i) for each partition. These are used to

Algorithm 1 Reindexing Manager

Input IVF Index I, target partition size τ_s, local indicator threshold τ_f, reclustering radius r_c, global indicator threshold τ_G
Output Updated Index I'

```
 1: function CheckReindex($I$)
 2:     $I' = I$
 3:     $\mathbf{C}' = \{\}$
 4:     for $c \in$ I.Partitions do // check for local violations
 5:         if $f(c) > \tau_f$ then
 6:             $\mathbf{C}' = \mathbf{C}' \cup \{c\}$
 7:     if $|\mathbf{C}'| > 0$ then
 8:         $I'$ = LocalReindex($\mathbf{I}', \mathbf{C}', \tau_s, r_c$)
 9:     if $G(I') > \tau_G$ then // check for global violation
10:         $I'$ = BuildIVF($I'$)
11:     return $I'$
```

determine if the reconstruction error has increased due to modifications of the partition. We also track the partition read temperature (T_i), which captures the frequency and recency of reads (scans) performed over the vectors in the partition.

At the global index level, we track the standard deviation of partition sizes (σ) to help us detect when partitions are imbalanced and the global reconstruction error (ε) to determine if the clustering quality has degraded globally. We do not use temperature at the global level. Table 1 summarizes the set of properties tracked by Ada-IVF.

Symbol	Description
σ	Standard deviation of partition size
ε	Reconstruction error (MSE) of vectors and assigned centroid
$\mathbf{X_i}$	Set of vectors in partition i
T_i	Partition read temperature
$\boldsymbol{\mu}_i$	running centroid of the partition

Table 1: Tracked Clustering Properties

Updating each partition's local properties is shown in Algorithm 2 and Algorithm 3. Algorithm 2 shows how vectors in a partition (Lines 2-5) and its running centroid (Lines 6-8) are updated when a batch of vectors X_δ are added or removed from the partition.

Read temperature T_i is a floating point value defined between $[1.0, \inf)$ that indicates the frequency and recency of reads for a given partition. The intuition behind tracking the temperature for a partition is that, for workloads with skewed access patterns, as we observed from real-world applications (Section 2.3), degradations of partitions that are frequently accessed have a larger impact on the overall workload performance. Therefore, Ada-IVF prioritizes partitions with high temperature (frequently accessed partitions) during reindexing. Temperature is updated for all partitions for each search operation issued to the system using the Algorithm 3. If a given query accesses a partition, its temperature increases (Lines 5-7) according to a multiplicative heating factor η; if not, the temperature is decreased by cooling factor ν (Line 9).

Algorithm 2 Partition Update Rule

Input partition vectors $\mathbf{X_i}$, partition size, mean μ_i, delta vectors X_δ, *isDelete*
Output updated partition and properties $\mathbf{X_{i+1}}$, μ_{i+1}

```
1: function UpdatePartitionProperties($\mathbf{X_i}, \mu_i, \mathbf{X}_\delta, isDelete$)
2:     if isDelete then
3:         $s_\delta = -|\mathbf{X}_\delta|$; $\mathbf{X_{i+1}} = \mathbf{X_i} - \mathbf{X}_\delta$
4:     else
5:         $s_\delta = |\mathbf{X}_\delta|$; $\mathbf{X_{i+1}} = \mathbf{X_i} \cup \mathbf{X}_\delta$
6:     $\mu_\delta = Mean(\mathbf{X}_\delta)$; $s_i = |\mathbf{X_i}|$          增量更新
7:     $s_{i+1} = s_i + s_\delta$
8:     $\mu_{i+1} = \mu_i + \frac{s_\delta}{s_{i+1}}(\mu_\delta - \mu_i)$
9:     return $\mathbf{X_{i+1}}, \mu_{i+1}$
```

Algorithm 3 Single Query Temperature Update

Input Query vector q, set of index partitions $\mathbf{C}$, nprobe n_p, heating parameter η, cooling parameter ν
Output Updated temperature for each partition $\mathbf{C}.T$

```
 1: function UpdateTemperature($q, \mathbf{C}, n_p, \eta, \nu$)
 2:     $\mathbf{C_{knn}}, \mathbf{C_d}$ = KNN($q$, $\mathbf{C}.\mu$, $k$=$n_p$)
 3:     $\mathbf{C_d} = \frac{\mathbf{C_d}[0]}{\mathbf{C_d}}$ // scale by distance to nearest centroid
 4:     for each $c \in \mathbf{C}$ do
 5:         if $c \in \mathbf{C_{knn}}$ then
 6:             $d_\mu = \mathbf{C_d}[c]$
 7:             $c.T = c.T(1 + d_\mu \eta)$
 8:         else   max
 9:             $min(c.T = c.T(1 - \nu), 1.0)$
10:     return $\mathbf{C}.T$
```

4.2 Local Reindexing

Next, we introduce Ada-IVF's local indicator function f, which is used to identify partitions that should be considered as candidates for reindexing and describe its local reindexing algorithm for addressing the increase in imbalance and reconstruction error without rebuilding the entire index from scratch.

4.2.1 Local Indicator Function f. The local indicator function (f) quantifies the deviation from a partition's original state w.r.t. imbalance and reconstruction error, determining when a local reindexing operation should be triggered. This function is evaluated for a single partition upon modification of the partition. The indicator function captures changes in the partition's size and mean over time, and the partition temperature is used as a scaling factor. If the value of f for a given partition exceeds a threshold, the partition is denoted as a *violator* and is selected for reclustering. Deviations in partition size introduce imbalance in the index, while change in the partition mean is a proxy for increasing reconstruction error.

Partition temperature is used to determine the severity of deviations; degradations in a partition with a high temperature have a significant impact on the overall query throughput for a workload because more queries access that partition. Therefore, we should more aggressively select highly accessed partitions for reindexing. Conversely, a partition with a low temperature (i.e., rarely or never

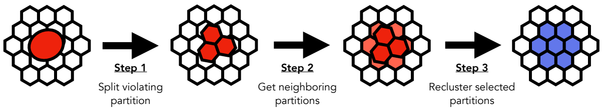

Step 1
Split violating partition
Step 2
Get neighboring partitions
Step 3
Recluster selected partitions

Figure 7: Local reindexing mechanism. Violating partitions are split to meet a target size and merged with neighboring partitions using balanced k-means to minimize imbalance and reconstruction error.

accessed) has a negligible effect on overall index performance. By using temperature in its indicator function, Ada-IVF can perform index maintenance *lazily*, i.e., when needed due to negative impact on index performance, instead of performing index maintenance *eagerly* at all times. As will be shown experimentally in Section 5, this significantly reduces index maintenance time with minimal impact on index performance.

Taking these factors into account, the local indicator function f for a given partition c is defined as:

$$f(c, c_0) = f_T(c.T)(\beta f_s(c.s, \tau_s) + (1 - \beta) f_d(c, c_0)) \tag{1}$$

The individual functions f_T, f_s, and f_d control the contribution of the partition temperature, deviation in size, and drift from the initial partition state c_0 to the current state c. $\beta \in [0, 1]$ controls the contribution of imbalance vs. drift. If the value of f exceeds the threshold τ_f, the corresponding partition is chosen for reindexing. We next discuss how we design f_T, f_s, and f_d.

Temperature Scaling Function f_T. The function f_T aims to capture how frequently we should re-index a partition based on the temperature T of the partition. Thus, if the temperature is high, it should have a larger score, and vice-versa for low temperature. Thus, we use a linear function parameterized by a scale factor α

$$f_T(T) = \alpha T \tag{2}$$

Local Size Imbalance Function f_s. The function f_s captures if a partition has grown too large or small. Thus, denoting τ_s as the target partition size, we define f_s as:

$$f_s(s, \tau_s) = \left\{ \begin{array}{ll} (s - \tau_s)/\tau_s, & \text{if } s \geq \tau_s \\ (\tau_s - s)/s, & \text{if } s < \tau_s \end{array} \right\} \tag{3}$$

Local Drift Function f_d. The function f_d captures if a partition c has drifted from its initial state c_0, which is a proxy for increasing reconstruction error. We quantify drift by measuring the relative change in partition mean μ from initial μ_0.

$$f_d = ||(\boldsymbol{\mu} - \boldsymbol{\mu_0})||/||\boldsymbol{\mu_0}|| \tag{4}$$

4.2.2 Local Reindexing Algorithm. Local reindexing consists of three phases: i) splitting and deleting of violating partitions, ii) finding the nearest partitions to the violating partitions, and iii) applying k-means to the vectors of the violating partitions and their neighbors (Algorithm 4). Partitions are selected as violators if the value of their indicator function f exceeds the threshold τ_f. If the size of the violating partition exceeds the target partition size τ_s, it is split using k-means (Lines 4-6). Otherwise, it is deleted (Line 9). The centroids of split and deleted partitions are used in KNN search to find their corresponding neighboring partitions (Line 10). The reindexing radius r_c controls how many neighboring partitions to consider for each violator, where a larger value will reduce reconstruction error while increasing the reindexing time. In our experiments, we used a fixed value of $r_c = 25$ to balance between reindexing time and minimizing error. A potential improvement of our method is to set r_c individually for each violator based on centroid distance, but we have not implemented this approach. Balanced k-means is applied to the region for ι iterations and is initialized using the previous centroid state M (Line 13). Finally, the newly created partitions are added to the index.

Algorithm 4 Local Reindexing

Input IVF Index I, Violating partitions C', target partition size τ_s, reclustering radius r_c, and local kmeans iterations ι

Output updated IVF Index I'

```
1:  function LocalReindex($I$, $C'$, $\tau_s$, $r_c$, $i$)
2:      $I' = I$; $\mathbf{X} = \{\}$; $\mathbf{M} = \{\}$
3:      for each $c \in C'$ do
4:          if $|c| > \tau_s$ then // Step 1: split violating partitions
5:              $\mathbf{S} = KMeans(c.X, k = \lceil |\mathbf{c.X}|| / \tau_s \rceil, \iota)$
6:              $\mathbf{M} = \mathbf{M} \cup \mathbf{S}.\mu$
7:          else
8:              $\mathbf{M} = \mathbf{M} \cup \{c.\mu\}$
9:          $\mathbf{X} = \mathbf{X} \cup c.\mathbf{X}$; $I' = I'$.remove($c$)
     // Step 2: get neighboring partitions
10:     $\mathbf{R} = KNN(\mathbf{M}, I'.\mu, k = r_c)$
11:     for each $c \in \mathbf{R}$ do
12:         $\mathbf{X} = \mathbf{X} \cup c.X$; $\mathbf{M} = \mathbf{M} \cup \{c.\mu\}$; $I' = I'$.remove($c$)
     // Step 3: recluster partitions with initialized centroids
13:     $\mathbf{C_n}$ = KMeans($\mathbf{X}$, initial_centroids=$\mathbf{M}$, $\iota$)
14:     for each $c \in \mathbf{C_n}$ do: $I'$.add($c$)
15:     return $I'$
```

4.3 Global Reindexing

While Ada-IVFs local reindexing mechanism can handle deviations in individual partitions when updates and their impact on partitions exhibit locality, it is possible to observe deviations in a large number of partitions or overall index quality that might render local, partition-specific maintenance ineffective. It is necessary for Ada-IVF to identify such an index state and perform a full index rebuild. Here, we describe the global indicator function G that we use to determine when to reconstruct the index from scratch instead of performing local reindexing. Similar to local reindexing, we use a *global* indicator function to quantify deviations in the overall clustering and construct a new index over the current state of $\mathbf{X}$.

The global indicator function G is a function of the current reconstruction error ε of the clustering, the partition size standard deviation σ, and the estimated reconstruction error ε', the estimation of the reconstruction error of k-means clustering if a full index rebuild were to occur over the current set of vectors.

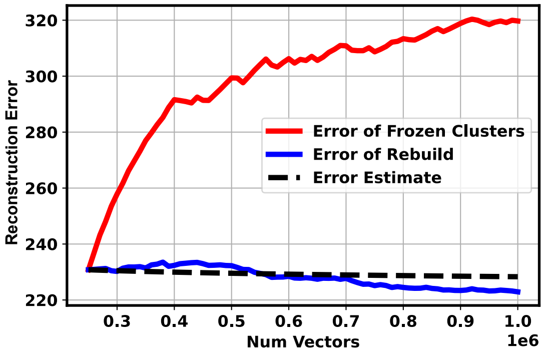


Figure 8: Error estimate ε' relative to true error for a growing index on Sift1M. Error of frozen clustering grows while the estimate is within 2.5% of that obtained by a full rebuild.

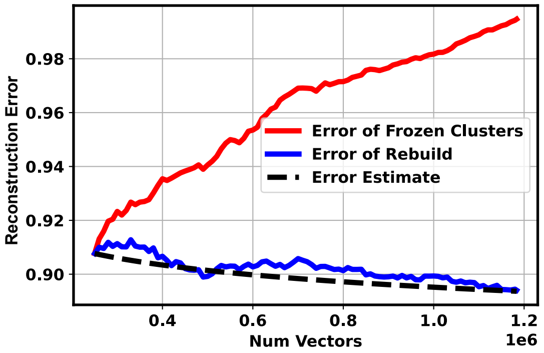


Figure 9: Error estimate ε' relative to true error for a growing index on Glove1M. Error of frozen clustering grows while the estimate is within 1% of that obtained by a full rebuild.

The function G is defined as:

$$G(\varepsilon, \sigma, \varepsilon') = \gamma G_s(\sigma) + (1-\gamma) G_d(\varepsilon, \varepsilon') \leq \tau_G \tag{5}$$

The individual functions G_s, G_ε capture the contribution of the size imbalance and reconstruction error, and γ controls their relative contribution; τ_G is a tunable threshold.

Global Imbalance Function G_s. We use the relative change in the standard deviation of partition sizes to estimate the increase in global imbalance. Intuitively, the standard deviation in partition size indicates how imbalanced the partitions are with respect to a balanced partitioning. The change in standard deviation captures if the imbalance has increased due to updates performed on the index.

$$G_s = |\sigma - \sigma_0| / \sigma_0 \tag{6}$$

Global Reconstruction Error Function G_ε. As a part of the global reindexing mechanism, we aim to trigger a rebuild if the global reconstruction error grows high. We propose doing this by estimating the projected improvement in reconstruction error that can be achieved if we perform a full index rebuild. Thus, we define G_o as the relative difference between the current observed error ε and an ideal error ε'. We define ideal error as the error that would be measured if a full index rebuild were performed, i.e., if k-means were to be applied over the current set of vectors. In other words, if the error of the index deviates far enough from the estimated ideal error, then a rebuild should be performed to recover performance. However, estimating the ideal reconstruction error is challenging and we next describe our approach to address this challenge.

$$G_o = |\varepsilon - \varepsilon'| / \varepsilon' \tag{7}$$

Estimating ε'. An IVF Index can be viewed as a vector quantization scheme, where groups of vectors (clusters) are represented by a representative encoding (centroid). It is proven that k-means clustering produces an empirically optimal quantizer [55] with error ε'. There are well-defined bounds [16] on the difference between the error of an empirically optimal quantizer ε' and an optimal quantizer ε^*, where the constant A is a data-dependent parameter.

$$\varepsilon' - \varepsilon^* \leq A \sqrt[d]{\frac{1}{n}} \tag{8}$$

Similarly, assuming that the initial state of the clustering is empirically optimal, we can derive a bound on its reconstruction error.

$$\varepsilon_0 - \varepsilon_0^* \leq A \sqrt[d]{\frac{1}{n_0}} \tag{9}$$

Combining 8 and 9 gives

$$\varepsilon' = \varepsilon_0 \sqrt[d]{\frac{n_0}{n}} + (\varepsilon^* - \varepsilon_0^* \sqrt[d]{\frac{n_0}{n}}) \tag{10}$$

We make the assumption that $(\varepsilon^* - \varepsilon_0^* \sqrt[d]{\frac{n_0}{n}})$ is negligible and achieve the final estimate.

$$\varepsilon' = \varepsilon_0 \sqrt[d]{\frac{n_0}{n}} \tag{11}$$

We empirically validate the estimate in Figures 8 and 9. Our experiments show that our estimate is within 2.5% and 1% of the true error of a rebuild for Sift1M and Glove1M vector datasets, respectively.

5 EXPERIMENTS

We evaluate Ada-IVF by using a variety of benchmark workloads against state-of-the-art techniques for IVF index maintenance. We begin with an end-to-end performance analysis on an internal workload trace and the BigANN-SS public benchmark (Section 5.2). We then present an extensive sensitivity analysis to demonstrate the robustness of Ada-IVF compared to baselines across various workload scenarios. To do this, we developed a tool that generates workloads from a given vector dataset with varied read and write properties (Section 5.3). Finally, we present results from a set of microbenchmarks that validate our contributions (Section 5.4).

Experimental Highlights

- On an internal recommendation workload, Ada-IVF reduces the update time to 62% of that of *LIRE* with a 9% improvement in QPS.
- For the public BIGANN-SS benchmark, Ada-IVF reduces the update time by 50% and matches the same QPS as *LIRE*.

Table 2: Workload Characteristics. r_{rw} = read/write ratio, r_{id} = insert/delete ratio

Workload	Update Size	r_{rw}	r_{id}	Write Locality	Read Locality
Internal	≈ 2%	10.0	≈ 5.0	high	high
BigANN-SS	0.001% - 10%	.01	≈ 2.0	high	high
Generator	variable	variable	variable	variable	variable

- Over a comprehensive set of synthetic workload configurations (used for sensitivity analysis), Ada-IVF consistently achieves the highest QPS over IVF baseline methodologies, and compared to *LIRE*, Ada-IVF achieves $1.5 - 5\times$ higher update throughput across all configurations.

5.1 Experimental Setup

5.1.1 Workloads. **Internal** is a real-world, industrial workload for an online recommendation application where both the updates to the index and the queries are processed in a streaming fashion. This read-heavy workload exhibits significant write locality as each batch of updates primarily targets a specific partition. We execute this workload using access properties derived from real-world data using MSTuring10m as the underlying vector dataset.

We also use **BigANN-SS**, a *batch* public streaming benchmark that is developed as part of the Neurips'23 BigANN competition. BigANN-SS uses a 30M subset of the MSTuring dataset, and its workload consists of batch inserts and delete,s where each batch is based on a cluster of the original dataset. By inserting and deleting based on partitions, BigANN-SS's workload exhibits update locality. The inserts and deletes vary in size between 10-250k. After each update, the same set of 10k queries are evaluated in bulk with a target recall of 0.9.

5.1.2 Workload Generator. We use a configurable workload generator that can simulate a variety of settings on any vector dataset. Using such a generator helps us evaluate the sensitivity and robustness of Ada-IVF and its components on public vector datasets. Given a vector dataset and workload parameters, the generator clusters the dataset and samples from the clusters to produce inserts, deletes, and queries. The six primary parameters of the generator are (i) the initial size s_0, (ii) the update size s_u, (iii) the insert/delete ratio r_{id}, (iv) the update cluster sample fraction CSF_u, (v) the read/write ratio r_{rw} and (vi) the query cluster sample fraction CSF_q, which is optional if queries are provided with the dataset. The update and query sample fractions allow for configuring the *locality* of updates and queries by controlling the fraction of vectors sampled from a given cluster to produce the update/query. With CSF=1.0, an entire cluster of vectors is sampled for an insert/delete; therefore, the update is highly localized. A small CSF (e.g., CSF=.001) corresponds to a less localized update, as vectors are sampled from many clusters across the vector space. We use the workload generator to conduct a sensitivity analysis using the MSTuring10M dataset. We use Sift1M and Glove1M for microbenchmarks. Unless otherwise specified, for all cases, we fix the following parameters: $s_0 = 0.1|X|$, $s_u = 10000$, $r_{id} = \text{inf}$ (insert only), $CSF_u = 1.0$, $r_{rw} = 0.1$, and use queries that come with the dataset so CSF_q does not apply.

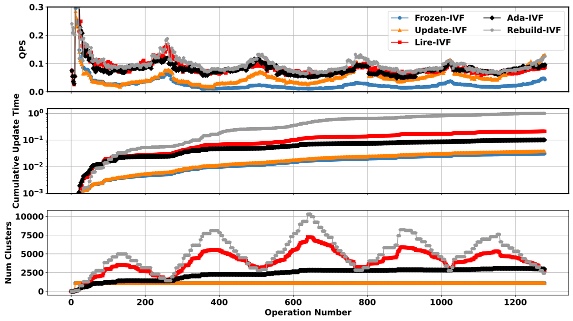


Figure 10: Workload trace for the BigANN-SS benchmark. Recall Target = .9. QPS and update times are normalized to the maximum observed value over the trace.

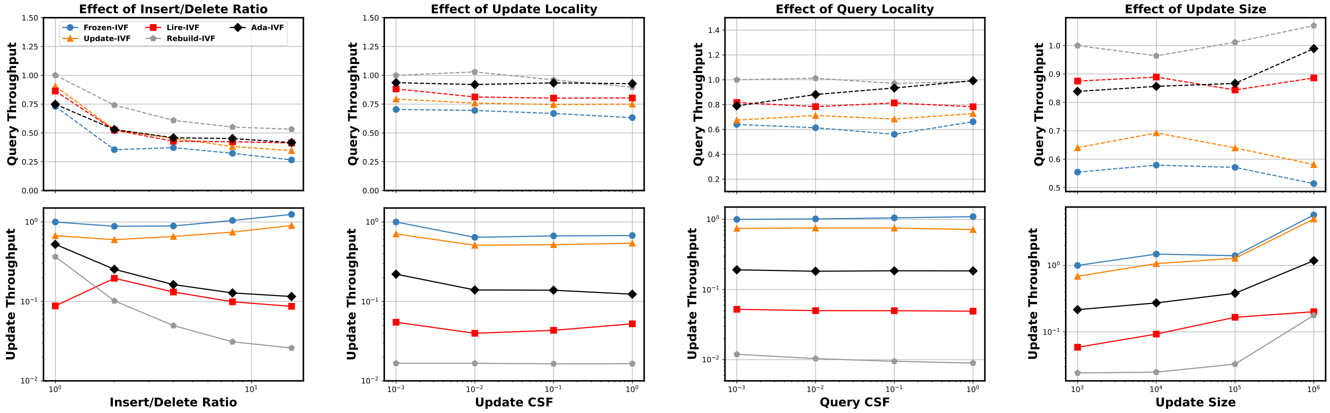
Table 3: Normalized search and update throughput for each workload. Throughput is normalized by the values obtained by *Rebuild*. Recall Target = .9.

Workload	Index	Normalized Search Throughput	Normalized Update Throughput
Internal	Frozen	.81	20.0
	Update	.84	12.5
	LIRE	.83	4.2
	Ada-IVF	**.89**	**6.7**
	Rebuild	1.0	1.0
BigANN-SS	Frozen	.27	33.3
	Update	.54	25.0
	LIRE	.89	4.7
	Ada-IVF	**.85**	**10.0**
	Rebuild	1.0	1.0

5.1.3 Baselines. We compare Ada-IVF's performance against the following standard and state-of-the-art maintenance strategies that are employed in existing systems for handling updates in IVF Indexes (Section 2). Ada-IVF's configurable indicator functions and flexible reindexing manager allow us to implement these algorithmic baselines in our system easily. This ensures that the system overheads are the same for all strategies. For ease of explanation, queries are serially executed in a synchronous fashion for all approaches and workloads.

- *Frozen*: Does no maintenance.
- *Update* [11, 15]: Updates centroids on partition modification to reflect the true mean of the partition.
- *Rebuild* [66]: Periodically rebuilds the index.
- *LIRE* [69]: Performs splitting and merging of partitions if a target size is violated. After splitting and merging, vector reassignment of neighboring partitions is performed. LIRE is implemented within Ada-IVF with the corresponding parameters $\tau_f = 0$, $\iota = 0$, and $\beta = 1.0$.

We set the target partition size for all baselines and workloads to 1000, which obtains good performance across workloads. We match the shared configuration parameters of Ada-IVF and *LIRE*, setting



(a) Vary Insert/Delete Ratio

(b) Vary Update Locality

(c) Vary Query Locality

(d) Vary Update Size

Figure 11: Sensitivity analysis conducted using the workload generator. Each experiment varies a single parameter and measures the effect on search and update throughput. Throughput is normalized to the maximum observed value for each experiment.

the maximum/minimum partition size to 2000/500 and using a reindexing radius of 25. For *Rebuild*, we trigger rebuilding after 2.5% of the total number of vectors in the workload have been modified. We individually tune the Ada-IVF specific parameters τ_f, α, and β for each workload. We measure normalized search throughput at a target recall of .9 and normalized update throughput. For clarity, we refer to search throughput as QPS. We run all experiments using an R5-24xlarge AWS instance with 96 vCPUs and 768GB of memory.

5.1.4 Implementation. We implement Ada-IVF using C++/Python using PyTorch 2.0 [54] as the underlying tensor library. OpenMP 5.1 [23] is used for parallelizing index operations. We use PAM [60] to map vector IDs to partition IDs for efficient deletes.

5.2 Industrial and Benchmark Workloads

Table 3 shows the search and update throughput on Internal and BigANN-SS of Ada-IVF and baselines relative to *Rebuild*, which is optimal for search throughput. Below, we detail each workload.

BigANN-SS. Figure 10 shows the workload trace for the BigANN-SS benchmark. In this workload, a series of inserts and deletes are performed, which grow and shrink the set of vectors in the index over time. First, looking at the search QPS (top figure), we observe that *Frozen* and *Update* achieve 3.5× and 1.9× less QPS than *Rebuild*, while *LIRE* and Ada-IVF can achieve similar QPS. We attribute this to the re-balancing mechanisms applied by both methods. There is still a gap between the QPS of Ada-IVF and *Rebuild*, suggesting room for improvement. The middle figure shows the cumulative update time over the workload normalized by *Rebuild*'s update time. *Frozen* and *Update* achieve the lowest update times as they have minimal overhead. *LIRE* and Ada-IVF obtain an update throughput of 4.7× and 10.0× relative to *Rebuild*, resulting in a lower total update time. Ada-IVF achieves a high update throughput due to the indicator function it employs, as it is not as aggressive in selecting partitions for reindexing. This can be observed in the bottom figure, which shows the number of partitions over time. The number of partitions varies significantly for *LIRE*, while Ada-IVF remains more stable throughout the workload. Overall, Ada-IVF achieves a 2× higher update throughput over *LIRE* while matching query throughput. Compared to *Rebuild*, it achieves 10× higher update throughput with 85% of the search throughput.

Internal Workload. Table 3 shows the search and update throughput for Internal. We see that *Frozen* and *Update* perform relatively well, showing an order of magnitude higher update time compared to *Rebuild* with only a .81× and .84× reduction in QPS for *Frozen* and *Update*, respectively. We attribute the good performance of the *Frozen* and *Update* to the large initial size of the workload. Unlike BigANN-SS, Internal starts with a significantly larger set of vectors, and *Frozen* and *Update* obtain a more representative initial clustering of the data points. *LIRE* achieves a similar QPS as *Frozen* with 4.7× higher update throughput than *Rebuild.* Ada-IVF achieves a higher QPS than baseline methods with .89× QPS with a further 1.6× improvement in update throughput over *LIRE.*

5.3 Sensitivity Analysis

To demonstrate the robustness of Ada-IVF to a variety of workload scenarios, we conduct a sensitivity analysis of Ada-IVF and baselines update methodologies using a workload generator. For each experiment, we vary a single generator parameter while keeping the rest fixed, as discussed in 5.1.2.

Insert/Delete Ratio. First, we vary the insert/delete ratio, which determines the index's rate of growth. We see that Ada-IVF matches the query performance of *LIRE* with a slight improvement in update time as the ratio increases. We also observe a general decrease in update and query throughput as the ratio increases. This is due to the growing dataset size for positive ratios.

Update Locality. Here, we vary the cluster sample fraction (CSF) used to generate inserts and deletes. This controls the locality of updates (5.1.2). Figure 11b shows the query and update throughput for Ada-IVF and baseline methods. Ada-IVF consistently matches the QPS of a *Rebuild* across all sample fractions, while other methods

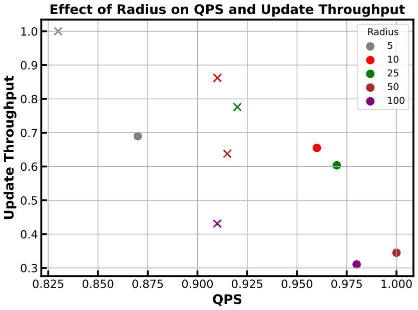


Figure 12: Parameter sweep varying the reindexing radius = {5, 10, 25, 50, 100}. X denotes *LIRE* configurations, and search and update throughput are normalized to the maximum value.

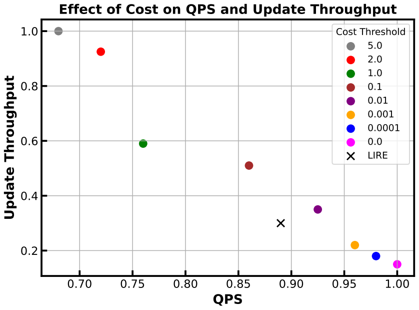


Figure 13: Parameter sweep varying the indicator threshold τ_f = {0, .0001 .001, .01, .1, 1.0, 2.0, 5.0}. X denotes *LIRE* configurations, and search and update throughput are normalized to the maximum value.

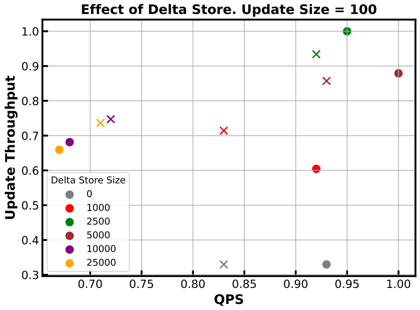


Figure 14: Parameter sweep varying the delta store size = {0, 1000, 2500, 5000, 10000, 25000}, X denotes *LIRE* configurations, and search and update throughput are normalized to the maximum value.

show degradation in QPS. *Frozen* and *Update* achieve the highest update throughput, with Ada-IVF achieving the next best update throughput, a 2 – 5× improvement over *LIRE*. We see that results are consistent across sample fractions.

Query Locality. Now, we vary the cluster sample fraction (CSF) used to generate queries. As query locality increases, Ada-IVF increases its QPS while maintaining a consistent update throughput (4× *LIRE*). This suggests that the temperature component causes aggressive reindexing of only the often-scanned partitions and thus increases query throughput. Other methods exhibit consistent behavior across all cases, which is expected given they do not consider locality. Overall, this validates our temperature model for utilizing read locality information.

Update Size. Figure 11d shows the effect of varying the update size on index performance. First, observing the impact on read throughput, we see Ada-IVF and *LIRE* obtain similar query performance across update sizes while *Frozen* and *Update* degrade in performance as the update size increases. For update throughput, all methods show an increase in throughput with increasing update size, demonstrating the effectiveness of batch on IVF index updates.

Takeaways. We see that across most workload scenarios, Ada-IVF best matches the query performance of *Rebuild* at a fraction of the update time of the next best baseline *LIRE*. Because of read temperature, Ada-IVF is especially powerful for workloads that exhibit significant read locality, where query performance increases as locality increases.

5.4 Microbenchmarks

We now evaluate individual components of our design in further detail using a synthetic SIFT1M workload with r_{id} = 1. We first evaluate the effect of indicator function parameters by performing a comprehensive parameter sweep of Ada-IVF parameters. We then perform a similar parameter scan to evaluate the effect of varying the local reindexing mechanism parameters. We then evaluate the effect of batching updates.

Effect of Reindexing Radius r_c. Figure 12 shows the effect of varying the reindexing radius. *LIRE* configurations (τ_f = 0, ι = 0, and β = 1.0) are marked with an X. We see that tuning the reindexing radius provides a clear trade-off between update throughput and query performance. We also observe that *LIRE* has higher update throughput for the same radius but with a lower QPS than Ada-IVF. *LIRE* also shows minimal QPS improvement beyond a radius of 10. Generally, a larger reindexing radius for Ada-IVF increases QPS and decreases update throughput. Index maintainers can tune this parameter to best meet their workload's needs.

Effect of Local Indicator Threshold τ_f. Figure 13 shows the effect of varying the threshold τ_f, which determines the value of f needed for a violator to be reindexed. *LIRE* configurations are marked with an X. Like the reindexing radius experiment, we see that tuning the local indicator threshold provides a clear trade-off between update and query throughput. A smaller threshold will trigger reindexing more often and better maintain index performance at a cost in update throughput. Generally, a smaller threshold increases QPS and decreases update throughput.

Delta Store. Now, we evaluate the effect on index performance when batching inserts with a *delta store*. The delta store is a fixed capacity list that stores recent updates. The contents of the delta store are scanned exhaustively by searches. When the delta store reaches capacity, its contents are flushed to the IVF index. Figure 14 shows the effect of varying the capacity of the delta store for a small update scenario (update size = 100). As before, we mark *LIRE* configurations with an X. We see that including a delta store can significantly improve update throughput and observe a slight improvement in QPS for small update sizes. However, if the delta store grows too large, query performance significantly degrades as queries must scan the delta store exhaustively. In conclusion, maintainers operating in small update environments can benefit significantly from the inclusion of a delta store in their index.

6 DISCUSSION

We outline extensions of our method to Product Quantization and hierarchical IVF indexes, and then we discuss related work.

6.1 Extensions

The IVF index maintenance techniques introduced in this work apply to single-level IVF indexes, but we believe they are applicable to any type of index based on vector quantization. Below, we discuss extensions of our index maintenance technique to two commonly used indexes: Product Quantization [40] and Hierarchical IVF indexes [59].

Product Quantization (PQ) encodes a d-dimensional vector into a d'-dimensional code vector using d' codebooks. Codebooks are obtained by KMeans to cluster d' subspaces of the vector distribution. Product quantization is generally used for lossy vector compression [40] and is a commonly used technique in ANN index design [66]. In a dynamic setting where new vectors are added, and existing vectors are deleted, the quality of the codebook trained on the initial vector dataset decreases. This eventually necessitates an expensive codebook rebuild and re-encoding of vectors to recover index quality. In contrast with IVF indexes, imbalance does not negatively affect query performance, as the only objective is to minimize the quantization error of the encoding. Hence, when applied in this setting, our indicator functions (Section 4) should only consider the error term ($\beta = 0$, $\gamma = 0$). However, a significant challenge with applying our method to PQ quantization is that, typically, there are few clusters in a given codebook. For example, it's common to use 8-bit codes corresponding to a total of 255 centroids in each codebook [40], meaning that a modest reindexing radius of 25 will recluster 10% of the data if a single cluster violates the indicator function. Therefore, local reindexing is more applicable to cases where larger codes (e.g., 16-bit codes, 65k centroids) are used. There exists some work that studies the impact of updates in the context of product quantization [45, 68]. Future study is required to assess the performance implications of using Ada-IVF's maintenance technique over product quantization codebooks.

Hierarchical IVF indexes [59] efficiently scale to large data sizes by maintaining a hierarchical KMeans clustering. In a basic two-level structure, vectors are grouped into fine-grained clusters whose centroids are further aggregated into coarse clusters. Changes to underlying vectors necessitate reindexing these fine clusters, impacting the coarse-level structure by centroid updates. We can extend our single-level methodology in Ada-IVF by using an independent update manager for each hierarchy level. This ensures that any changes in the fine-grained centroids have a limited effect on the quality of the coarse-grained clustering. Utilizing this approach, we can extend our method to maintain hierarchical IVF indexes.

We reserve the study of streaming workloads and approaches for handling updates on other types of indexes for future work.

6.2 Related Work

Vector Indexes. Indexes for vector search predominately fall under two categories: partitioned and graph indexes. IVF indexes are the most prominent type of partitioned index with a multitude of variants [12, 14, 21, 29, 41, 66]. IVFADC [41] pairs IVF indexes with Product Quantization (PQ). SCANN uses a hierarchical IVF index and PQ and includes optimizations for Maximum Inner Product Search (MIPS). [29, 59]. SPANN [21] replicates vectors across partitions and includes a graph index over the centroids to accelerate centroid scans. Hash-based [26, 37, 61], learned [10, 30, 31, 43], and tree-based [22, 42, 44, 52, 65] indexes are alternative types of partitioned indexes, but do not yet outperform leading IVF and graph approaches on public benchmarks [57?]. Graph indexes [24, 50, 64, 70] are the leading alternative solution to IVF indexes; however, they face challenges supporting updates at scale due to their large memory overhead and random access patterns [69]. SPFresh (the system that implements LIRE) shows superior performance to the leading updatable graph index DiskANN [50, 58].

Vector Data Management Systems. There has been an explosion of interest in vector search due to the success of embedding-based machine learning methods. To meet demand, new vector data management systems, known as vector databases, have sprouted [2–8, 63] and existing data management systems have added vector search capability [1]. Vector databases are typically general to the underlying index and support both IVF and graph index approaches. We believe our maintenance approach can be directly adopted by vector databases to improve their performance in update-heavy environments.

Updatable Spatial Indexes. The R-tree [32] is a classic example of a dynamic spatial index. R-trees maintain index performance by splitting and deleting leaf nodes once they become full or too small. The mechanism for splitting nodes aims to minimize overlap between the resulting nodes such that queries visit as few nodes as possible. In IVF indexes, overlap between clusters is difficult to measure, so we instead use the reconstruction error. Numerous r-tree variants have been proposed to improve the splitting procedure by minimizing overlap, such as [17] and [18].

7 CONCLUSION

We introduced Ada-IVF, an incremental methodology for maintaining IVF index performance for dynamic workloads. Ada-IVF uses local and global indicator functions that determine which clusters need to be reindexed using local reindexing. We evaluated our approach across a variety of benchmarks and found that compared with the state-of-the-art dynamic IVF index maintenance strategy, Ada-IVF achieves an average of 2× and up to 5× higher update throughput while matching query throughput across a range of benchmark workloads.

REFERENCES

[1] [n.d.]. pgvector/pgvector: Open-source vector similarity search for Postgres. https://github.com/pgvector/pgvector

[2] [n.d.]. Qdrant - Vector Database. https://qdrant.tech/

[3] [n.d.]. the AI-native open-source embedding database. https://www.trychroma.com

[4] [n.d.]. Vald. https://vald.vdaas.org/

[5] [n.d.]. The vector database to build knowledgeable AI | Pinecone. https://www.pinecone.io/

[6] [n.d.]. Vespa - data + AI, online. https://vespa.ai/

[7] [n.d.]. Welcome to Apache Lucene. https://lucene.apache.org/index.html

[8] [n.d.]. Why TileDB as a Vector Database. https://tiledb.com/blog/why-tiledb-as-a-vector-database

[9] Cecilia Aguerrebere, Mark Hildebrand, Ishwar Singh Bhati, Theodore Willke, and Mariano Tepper. 2024. Locally-Adaptive Quantization for Streaming Vector Search. http://arxiv.org/abs/2402.02044 arXiv:2402.02044 [cs].

[10] Abdullah Al-Mamun, Hao Wu, Qiyang He, Jianguo Wang, and Walid G. Aref. 2024. A Survey of Learned Indexes for the Multi-dimensional Space. http://arxiv.org/abs/2403.06456 arXiv:2403.06456 [cs].

[11] Relja Arandjelovic and Andrew Zisserman. 2013. All about VLAD. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 1578–1585.

[12] Artem Babenko and Victor Lempitsky. 2015. The Inverted Multi-Index. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 6 (June 2015), 1247–1260. https://doi.org/10.1109/TPAMI.2014.2361319

[13] Artem Babenko and Victor Lempitsky. 2015. Tree quantization for large-scale similarity search and classification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Boston, MA, USA, 4240–4248. https://doi.org/10.1109/CVPR.2015.7299052

[14] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Vol. 11216. Springer International Publishing, Cham, 209–224. https://doi.org/10.1007/978-3-030-01258-8_13 Series Title: Lecture Notes in Computer Science.

[15] Dmitry Baranchuk, Matthijs Douze, Yash Upadhyay, and I. Zeki Yalniz. 2023. DeDrift: Robust Similarity Search under Content Drift. http://arxiv.org/abs/2308.02752 arXiv:2308.02752 [cs].

[16] P.L. Bartlett, T. Linder, and G. Lugosi. 1998. The minimax distortion redundancy in empirical quantizer design. *IEEE Transactions on Information Theory* 44, 5 (1998), 1802–1813. https://doi.org/10.1109/18.705560

[17] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. ACM, Atlantic City New Jersey USA, 322–331. https://doi.org/10.1145/93597.98741

[18] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. 1996. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB '96)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 28–39.

[19] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[20] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher RÃ©, and Kevin Murphy. 2022. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. *Journal of Machine Learning Research* 23, 89 (2022), 1–64. http://jmlr.org/papers/v23/20-852.html

[21] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. [n.d.]. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search. ([n. d.]).

[22] Yewang Chen, Lida Zhou, Yi Tang, Jai Puneet Singh, Nizar Bouguila, Cheng Wang, Huazhen Wang, and Jixiang Du. 2019. Fast neighbor search by using revised k-d tree. *Information Sciences* 472 (2019), 145–162. https://doi.org/10.1016/j.ins.2018.09.012

[23] L. Dagum and R. Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering* 5, 1 (1998), 46–55. https://doi.org/10.1109/99.660313

[24] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2018. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. http://arxiv.org/abs/1707.00143 arXiv:1707.00143 [cs].

[25] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (April 2014), 744–755. https://doi.org/10.1109/TPAMI.2013.240 Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[26] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. [n.d.]. Similarity Search in High Dimensions via Hashing. ([n. d.]).

[27] Albert Gordo, Jon Almazán, Jerome Revaud, and Diane Larlus. 2016. Deep image retrieval: Learning global representations for image search. In *European conference on computer vision*. Springer, 241–257.

[28] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 311–320.

[29] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 3887–3896. https://proceedings.mlr.press/v119/guo20h.html ISSN: 2640-3498.

[30] Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J. Smola. 2022. BLISS: A Billion scale Index using Iterative Re-partitioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Washington DC USA, 486–495. https://doi.org/10.1145/3534678.3539414

[31] Nilesh Gupta and Patrick H Chen. [n.d.]. ELIAS: End-to-End Learning to Index and Search in Large Output Spaces. ([n. d.]).

[32] Antonin Guttman. 1984. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data - SIGMOD '84*. ACM Press, Boston, Massachusetts, 47. https://doi.org/10.1145/602259.602266

[33] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C Turnbull, Brendan M Collins, et al. 2019. Applying deep learning to airbnb search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1927–1935.

[34] Helia Hashemi, Aasish Pappu, Mi Tian, Praveen Chandar, Mounia Lalmas, and Benjamin Carterette. 2021. Neural instant search for music and podcast. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2984–2992.

[35] MD Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. 2019. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CsUR)* 51, 6 (2019), 1–36.

[36] Ihab F. Ilyas, Theodoros Rekatsinas, Vishnu Konda, Jeffrey Pound, Xiaoguang Qi, and Mohamed Soliman. 2022. Saga: A Platform for Continuous Construction and Serving of Knowledge at Scale. In *Proceedings of the 2022 International Conference on Management of Data*. 2259–2272. https://doi.org/10.1145/3514221.3526049

[37] Prateek Jain, Brian Kulis, Inderjit Dhillon, and Kristen Grauman. 2008. Online Metric Learning and Fast Similarity Search. In *Advances in Neural Information Processing Systems*, Vol. 21. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2008/hash/aa68c75c4a77c87f97fb686b2f068676-Abstract.html

[38] Wenqi Jiang, Shigang Li, Yu Zhu, Johannes De Fine Licht, Zhenhao He, Runbin Shi, Cedric Renggli, Shuai Zhang, Theodoros Rekatsinas, Torsten Hoefler, and Gustavo Alonso. 2023. Co-design Hardware and Algorithm for Vector Search. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, Denver CO USA, 1–15. https://doi.org/10.1145/3581784.3607045

[39] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2010. Improving Bag-of-Features for Large Scale Image Search. *International Journal of Computer Vision* 87, 3 (May 2010), 316–336. https://doi.org/10.1007/s11263-009-0285-2

[40] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (Jan. 2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57 Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[41] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. arXiv:1102.3828 [cs.IR]

[42] Haitao Li, Qingyao Ai, Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Zheng Liu, and Zhao Cao. 2023. Constructing Tree-based Index for Efficient and Effective Dense Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Taipei Taiwan, 131–140. https://doi.org/10.1145/3539618.3591651

[43] Wuchao Li, Chao Feng, Defu Lian, Yuxin Xie, Haifeng Liu, Yong Ge, and Enhong Chen. 2023. Learning Balanced Tree Indexes for Large-Scale Vector Retrieval. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Long Beach CA USA, 1353–1362. https://doi.org/10.1145/3580305.3599406

[44] King-Ip Lin and Congjun Yang. 2001. *The ANN-tree: an index for efficient approximate nearest neighbor search.* https://doi.org/10.1109/DASFAA.2001.916376 Pages: 181.

[45] Chong Liu, Defu Lian, Min Nie, and Xia Hu. 2020. Online optimized product quantization. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 362–371.

[46] David C Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related pins at pinterest: The evolution of a real-world recommender system. In *Proceedings of the 26th international conference on world wide web companion*. 583–592.

[47] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, and Youlong Cheng. 2022. Monolith: Real Time Recommendation System With Collisionless Embedding Table. In *5th Workshop on Online Recommender Systems and User Modeling (ORSUM2022), in conjunction with the 16th ACM Conference on Recommender Systems*.

[48] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. 2021. Deep learning–based text classification: a comprehensive review. *ACM Computing Surveys (CSUR)* 54, 3 (2021), 1–40.

[49] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F. Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-Throughput Vector Similarity Search in Knowledge Graphs. *Proceedings of the ACM on Management of Data* 1, 2 (June 2023), 1–25. https://doi.org/10.1145/3589777

[50] Jiongkang Ni, Xiaoliang Xu, Yuxiang Wang, Can Li, Jiajie Yao, Shihai Xiao, and Xuecang Zhang. 2023. DiskANN++: Efficient Page-based Search over Isomorphic Mapped Graph Index using Query-sensitivity Entry Vertex. http://arxiv.org/abs/2310.00402 arXiv:2310.00402 [cs].

[51] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1933–1942.

[52] Stephen M Omohundro. [n.d.]. Five Balltree Construction Algorithms. ([n. d.]).

[53] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2311–2320.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *CoRR* abs/1912.01703 (2019). arXiv:1912.01703 http://arxiv.org/abs/1912.01703

[55] David Pollard. 1981. Strong consistency of k-means clustering. *The annals of statistics* (1981), 135–140.

[56] An Qin, Mengbai Xiao, Yongwei Wu, Xinjie Huang, and Xiaodong Zhang. 2021. Mixer: efficiently understanding and retrieving visual content at web-scale. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2906–2917.

[57] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2022. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. http://arxiv.org/abs/2205.03763 arXiv:2205.03763 [cs].

[58] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search. *arXiv preprint arXiv:2105.09613* (2021).

[59] Philip Sun, Ruiqi Guo, and Sanjiv Kumar. 2023. Automating Nearest Neighbor Search Configuration with Constrained Optimization. http://arxiv.org/abs/2301.01702 arXiv:2301.01702 [cs].

[60] Yihan Sun, Daniel Ferizovic, and Guy E. Belloch. 2018. PAM: parallel augmented maps. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '18)*. ACM. https://doi.org/10.1145/3178487.3178509

[61] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. 2013. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proceedings of the VLDB Endowment* 6, 14 (Sept. 2013), 1930–1941. https://doi.org/10.14778/2556549.2556574

[62] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 839–848.

[63] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[64] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *Proceedings of the ACM on Management of Data* 2, 1 (March 2024), 1–27. https://doi.org/10.1145/3639269 arXiv:2401.02116 [cs].

[65] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. [n.d.]. Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. ([n. d.]).

[66] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3152–3165.

[67] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N Holtmann-Rice, David Simcha, and Felix Yu. 2017. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/hash/b6617980ce90f637e68c3ebe8b9be745-Abstract.html

[68] Donna Xu, Ivor W Tsang, and Ying Zhang. 2018. Online product quantization. *IEEE Transactions on Knowledge and Data Engineering* 30, 11 (2018), 2185–2198.

[69] 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. 2023. SPFresh: Incremental In-Place Update for Billion-Scale Vector Search. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 545–561. https://doi.org/10.1145/3600006.3613166

[70] Zhaozhuo Xu, Weijie Zhao, Shulong Tan, Zhixin Zhou, and Ping Li. 2022. Proximity Graph Maintenance for Fast Online Nearest Neighbor Search. *arXiv preprint arXiv:2206.10839* (2022).

[71] Zili Zhang, Chao Jin, Linpeng Tang, Xuanzhe Liu, and Xin Jin. [n.d.]. Fast, Approximate Vector Queries on Very Large Unstructured Datasets. ([n. d.]).