CANDOR-Bench: Benchmarking In-Memory Continuous ANNS under Dynamic Open-World Streams [Experiments & Analysis]

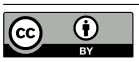
MINGQI WANG*, Huazhong University of Science and Technology, China
JUNYAO DONG*, Nanyang Technological University, Singapore
ZHUOYAN WU, National University of Singapore, Singapore
JUN LIU†, Huazhong University of Science and Technology, China
RUICHENG ZHANG, Huazhong University of Science and Technology, China
JIANJUN ZHAO, Huazhong University of Science and Technology, China
RUIPENG WAN, Huazhong University of Science and Technology, China
XINYAN LEI, Huazhong University of Science and Technology, China
SHUHAO ZHANG‡, Huazhong University of Science and Technology, China
BOLONG ZHENG, Huazhong University of Science and Technology, China
HAIKUN LIU, Huazhong University of Science and Technology, China
XIAOFEI LIAO, Huazhong University of Science and Technology, China
HAI JIN, Huazhong University of Science and Technology, China

Continuous *Approximate Nearest Neighbor Search* (ANNS) over real-time vector data streams is an increasingly critical yet underexplored problem. In open-world settings—where data distributions shift, noise accumulates, and concurrent access is common—existing ANNS algorithms, originally designed for static or simplified streaming scenarios, struggle to balance ingestion latency, retrieval quality, and update efficiency. While benchmarks such as `ANN-Benchmarks` and `Big-ANN-Benchmarks` have standardized evaluation in static or large-scale settings, they fail to capture the nuanced, high-churn dynamics of real-world streams. We introduce `CANDOR-Bench` (**C**ontinuous **A**pproximate **N**earest neighbor search under **D**ynamic **O**pen-wo**R**ld Streams), a benchmarking framework built on Big-ANN-Benchmark to evaluate in-memory ANNS under dynamic,

*The first two authors contributed equally to this research.

†Mingqi Wang, Jun Liu, Ruicheng Zhang, Jianjun Zhao, Ruipeng Wan, Xinyan Lei, Shuhao Zhang, Bolong Zheng, Haikun Liu, Xiaofei Liao, and Hai Jin are with the National Engineering Research Center for Big Data Technology and System, Service Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China.
‡Shuhao Zhang is the corresponding author and can be contacted at shuhao_zhang@hust.edu.cn.

Authors' Contact Information: Mingqi Wang, Huazhong University of Science and Technology, China, mingqiwang@hust.edu.cn; Junyao Dong, Nanyang Technological University, Singapore, junyao002@e.ntu.edu.sg; Zhuoyan Wu, National University of Singapore, Singapore, e1132310@u.nus.edu; Jun Liu, Huazhong University of Science and Technology, China, liujun@hust.edu.cn; Ruicheng Zhang, Huazhong University of Science and Technology, China, ruichengzhang@hust.edu.cn; Jianjun Zhao, Huazhong University of Science and Technology, China, curry_zhao@hust.edu.cn; Ruipeng Wan, Huazhong University of Science and Technology, China, u202312454@hust.edu.cn; Xinyan Lei, Huazhong University of Science and Technology, China, leixy@hust.edu.cn; Shuhao Zhang, Huazhong University of Science and Technology, China, shuhao_zhang@hust.edu.cn; Bolong Zheng, Huazhong University of Science and Technology, China, bolongzheng@hust.edu.cn; Haikun Liu, Huazhong University of Science and Technology, China, hkliu@hust.edu.cn; Xiaofei Liao, Huazhong University of Science and Technology, China, xfliao@hust.edu.cn; Hai Jin, Huazhong University of Science and Technology, China, hjin@hust.edu.cn.



ACM 2836-6573/2026/2-ART16
https://doi.org/10.1145/3786630

open-world conditions. CANDOR-Bench supports high-frequency ingestion (up to hundreds of thousands of vectors per second), adaptive drift modeling (including modality shifts), stochastic noise injection, and concurrent query-update execution—all without requiring modifications to algorithm code. Across 12 datasets and 19 representative ANNS algorithms, our evaluation reveals that no single ANNS algorithm consistently delivers high recall, throughput, and update efficiency across dynamic open-world scenarios, which challenges assumptions drawn from static benchmarks. This variability reflects deeper trade-offs inherent to streaming settings. For example, smaller update batches improve data freshness but can introduce higher insertion overhead and reduce accuracy. We further observe that throughput in concurrent settings is often constrained by insertion overhead rather than query latency, which highlights a mismatch between streaming workloads and designs originally tuned for offline construction.

CCS Concepts: • **Information systems → Stream management**; *Retrieval models and ranking*.

Additional Key Words and Phrases: Approximate Nearest Neighbor Search, Vector Database, Data Streams, Benchmarking

ACM Reference Format:
Mingqi Wang, Junyao Dong, Zhuoyan Wu, Jun Liu, Ruicheng Zhang, Jianjun Zhao, Ruipeng Wan, Xinyan Lei, Shuhao Zhang, Bolong Zheng, Haikun Liu, Xiaofei Liao, and Hai Jin. 2026. CANDOR-Bench: Benchmarking In-Memory Continuous ANNS under Dynamic Open-World Streams [Experiments & Analysis]. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 16 (February 2026), 27 pages. https://doi.org/10.1145/3786630

1 Introduction

Continuous *Approximate Nearest Neighbor Search* (ANNS) over vector data streams presents profound challenges, especially in open-world environments [15, 27, 49, 56]. Unlike static datasets, real-world streams are characterized by unpredictable arrival rates, evolving semantics, and persistent read–write contention on shared index structures. We identify four key challenges: (1) *High-velocity data dynamics*, where uncontrollable ingestion rates degrade retrieval accuracy and freshness due to indexing delays and outdated results [22]; (2) *Data drifts*, including both evolving distributions [3] and cross-modal transitions [15], which undermine index reliability over time; (3) *Noise and stochastic loss*, arising from packet drops, sensor errors, or unreliable sources [9, 24], which reduce resilience to real-world anomalies; and (4) *Concurrent queries and updates*, which introduce read-write conflicts and synchronization overhead [26], severely affecting throughput in streaming environments.

These challenges are not merely theoretical—they are pervasive in dynamic open-world vector streams, where data arrives unpredictably, evolves rapidly, and demands real-time responsiveness. For example, in real-time personalization at Airbnb [15], major events such as sports tournaments can trigger bursts in user activity, resulting in high-velocity ingestion and sudden shifts in user preferences. These patterns illustrate the dual pressures of freshness and adaptability under evolving data distributions. Similarly, Marqo [33] handles multimodal queries that combine textual and visual inputs. When modality distributions drift over time, retrieval performance degrades unless indexing mechanisms can adapt. In the Manu vector database [17], concurrent workloads such as virus scanning and video recommendations require systems to ingest and query simultaneously at scale—exacerbating synchronization and throughput trade-offs.

While recent benchmarks like ANN-Benchmarks [5] and Big-ANN-Benchmarks [42] have advanced standardized evaluation of ANNS algorithms, they fall short in supporting the core requirements of dynamic open-world vector streams. ANN-Benchmarks focuses on static datasets, evaluating retrieval quality and efficiency over fixed indices, but lacks support for runtime updates or distributional changes. Big-ANN-Benchmarks extends ANN-Benchmarks to billion-scale evaluation and introduces several tracks—including streaming, *Out-of-Distribution* (OOD), sparse,

Table 1. Summary of ANNS Benchmarks and Support for Open-World Challenges

Feature	ANN-Benchmarks	Big-ANN-Benchmark[1]	CANDOR-Bench
Target Scenario	Static	Simplified Streaming	Open-World Streaming
Streaming Inputs	×	△[2]	✓
Data Drifts	×	×	✓
Noise and Loss	×	×	✓
Concurrency	×	×	✓
Hardware Profiler	×	×	✓

[1] We only consider its "streaming" track, which considers a simplified streaming workload. There are three other workload scenarios which are beyond the scope of this study.
[2] △ indicates partial support, limited to specific tracks. The "streaming" track of Big-ANN-Benchmarks lacks mechanisms corresponding to our open-world challenges: event-rate control, micro-batching, fault injection, and fine-grained concurrency.

and filtered queries—but its streaming track assumes pull-based updates and coarse-grained ingestion, missing the finer temporal dynamics, distribution shifts, and concurrency typical of modern real-time workloads.

As summarized in Table 1, these benchmarks lack mechanisms for event-rate control, micro-batching, fault injection, or fine-grained concurrency management—all of which are vital for evaluating the responsiveness, adaptability, and efficiency of ANNS in realistic deployments. Furthermore, existing efforts largely overlook hardware-level profiling, which is necessary to understand system bottlenecks of running ANNS algorithms.

To systematically evaluate ANNS algorithms under dynamic open-world vector streams, we introduce `CANDOR-Bench` (**C**ontinuous **A**pproximate **N**earest neighbor search under **D**ynamic **O**penwo**R**ld Streams)—an open-source benchmarking framework built on and extending the codebase of `Big-ANN-Benchmarks`. Its goal is to provide a practical, reproducible basis for selecting in-memory ANNS under dynamic open-world streams by jointly evaluating query accuracy, throughput, and ingestion scalability to surface real-deployment trade-offs and scenario-based recommendations. Unlike prior benchmarks [5, 6, 42, 50] that assume static indices or simplified workloads, `CANDOR-Bench` is designed to emulate real-world scenarios featuring continuous ingestion, evolving data distributions, and concurrent query workloads. To ensure breadth and relevance, `CANDOR-Bench` evaluates 19 widely-used ANNS algorithms across four algorithmic families (Section 4.1), and includes 12 datasets spanning different domains and streaming characteristics (Section 4.2). The system maintains plug-and-play compatibility with `Big-ANN-Benchmarks`-based implementations, supporting seamless integration without modifying algorithm code. All datasets, algorithm integrations, and benchmarking tools are open-sourced at https://github.com/CGCL-codes/CANDOR-Bench, enabling reproducible and extensible evaluation.

Unless explicitly mentioned, all experiments in `CANDOR-Bench` run on a single-node in-memory and use single-threaded execution by default, to isolate intrinsic algorithmic behaviors from parallel coordination and I/O interference, and to ensure fair comparisons because many ANNS implementations lack stable multi-threaded support for index construction. Since parallel execution can be viewed as an interleaving of serial operations [21], observations from the single-threaded baseline remain a useful reference when interpreting results under multi-threaded execution. This setup follows common practice in prior ANNS research [10, 13, 18, 30]. While `CANDOR-Bench` also supports SSD-backed evaluation when the index exceeds memory capacity [2, 43, 47], our primary focus remains on this regime for clarity. We exclude GPU-accelerated and distributed

settings [16, 18, 36, 59, 63], as hardware-specific optimizations and coordination overheads can obscure core algorithmic trade-offs; we leave systematic evaluation under open-world conditions as future work.

2 Open-World Challenges

We define a *vector data stream* as a potentially unbounded sequence of data windows $\{W_k\}_{k\geq 1}$, where each window W_k contains a batch of new vector entries arriving at time step k. The window size is configurable and determines the granularity of streaming updates. At each step, the system must ingest the current window W_k into the ANNS index, while preserving accurate retrieval across the cumulative dataset $\bigcup_{i=1}^{k} W_i$. The objective in time-critical applications is to maintain low-latency, high-recall query capability across all ingested data, even as the stream evolves.

Such requirements arise in real-world systems like real-time recommendation [15] and virus scanning [17], where new data arrives continuously and must be promptly indexed. For example, in a streaming setup with configurable time windows, arrivals within each window are aggregated and ingested atomically at the window boundary, while queries may run at any time over all committed data. Maintaining both ingestion efficiency and retrieval quality becomes especially challenging in *open-world environments*, which are characterized by non-stationary input distributions, unpredictable arrival rates, incomplete or noisy data, and concurrent query–update workloads. To this end, we identify four core challenges in open-world vector stream processing: (1) *High-velocity data dynamics*, (2) *Data drifts and modality shifts*, (3) *Noise and stochastic loss*, and (4) *Concurrent operations*.

2.1 High-Velocity Data Dynamics

Real-time vector data streams in open-world environments introduce substantial pressure on ANNS systems, as they must rapidly incorporate new data while simultaneously handling queries and deletions. Applications such as recommendation and content moderation [22, 45] generate data at rates that challenge index maintenance and update latency.

The core challenge lies in balancing data freshness, update efficiency, and retrieval quality. Frequent updates may incur high insertion overhead or trigger index instability, while infrequent updates risk returning stale results. Deletions—though widely used in practice—can further amplify performance volatility if not handled efficiently.

Existing benchmarks [5, 42] typically assume static indices or batch-style updates and thus fail to capture the operational constraints imposed by high-velocity, fine-grained data streams. Addressing this challenge is essential for deploying ANNS in time-sensitive open-world scenarios.

2.2 Data Drifts

Data drift is a major challenge for ANNS in open-world vector streams, where underlying data distributions evolve continuously over time. As observed in real-world applications [3, 15], these changes can occur gradually or abruptly, and often affect both the content and structure of the indexed vectors. Following Souza et al. [44], data drifts can be broadly categorized into:

- **Prior probability drift**, where the distribution of labels $P(Y)$ changes while $P(Y|X)$ remains the same.
- **Covariate drift**, where the distribution of features $P(X)$ changes but the label conditional distribution $P(Y|X)$ is unchanged.
- **Concept drift**, where the relationship $P(Y|X)$ itself changes over time.

These types of drift may cause historical data to mislead current predictions, degrading retrieval accuracy. Open-world streaming adds further complexity due to the integration of multi-modal

content—such as text, images, and audio—each exhibiting different distributions, dimensions, and arrival patterns. This amplifies the impact of drift on both indexing and retrieval quality. Moreover, in graph-based ANNS systems, such drifts may also manifest as *graph structure drift* [58], where frequent deletions or updates reshape the index topology over time. This dynamic restructuring may lead to increased search path lengths or disconnected components, reducing query accuracy and efficiency. Robust ANNS algorithms must therefore adapt to both data-level and structure-level drift to remain effective in long-running, high-churn deployments.

2.3 Noise and Stochastic Loss

Noise and stochastic loss are inherent in many real-world data streams, such as sensor readings, wireless transmissions, or user-generated content, where transient failures or uncertain measurements are common [9, 24]. These factors inject high-variance or incomplete data into the stream, degrading both the semantic quality of the vectors and the structural consistency of indexing.

In contrast to data drift—where data distributions evolve but often retain semantic coherence—noisy inputs may follow entirely unstructured or adversarial distributions, making them harder to isolate or adapt to. Similarly, stochastic packet loss may silently remove a portion of the data, breaking the continuity of data arrival and fragmenting the index structure.

Such disturbances are particularly harmful to ANNS algorithms that rely on well-formed, globally connected structures, such as proximity graphs or quantization clusters. Missing or malformed vectors can distort pruning heuristics or lead to disconnected regions, severely reducing retrieval performance. Open-world systems must therefore contend not only with evolving data, but also with unpredictable corruption or incompleteness—making robustness to noise and data loss a fundamental design requirement.

2.4 Concurrent Queries

Concurrent query processing is essential in real-time vector search systems, where read and write operations occur simultaneously as new data arrives. Unlike static settings, where indices are pre-built and read-only, open-world streaming requires ANNS algorithms to support frequent updates while maintaining responsive query performance.

Most existing ANNS algorithms lack built-in support for concurrent read-write execution and instead rely on system-level locking mechanisms—such as those used in Milvus [48] and Qdrant [39]—to maintain consistency. While effective in ensuring correctness, such locking often leads to performance bottlenecks: during updates, queries are stalled, reducing overall throughput under high-concurrency conditions [14].

A key challenge lies in balancing consistency and performance. Fine-grained updates from streaming workloads amplify synchronization overhead, especially with small micro-batches. In such cases, even algorithms with fast query execution may experience reduced overall throughput, as insertions dominate the critical path. This shifts the bottleneck from query latency to update efficiency—a reversal from traditional static evaluation assumptions.

Recent efforts, such as ParlayANN [32], explore asynchronous, snapshot-based construction to decouple reads from writes. These designs avoid blocking reads during updates, offering new trade-offs between consistency, freshness, and throughput. As open-world workloads demand increasingly low-latency, high-frequency operation, the ability to scale concurrent query-update execution becomes a core evaluation dimension for ANNS algorithms.

3 Related Work

We review prior work across three areas: ANNS algorithm design, benchmarking frameworks, and system-level support for streaming vector search.

3.1 ANNS Algorithms

ANNS has been widely applied in recommendation [11, 65], information retrieval [31, 62], and vector database systems [48]. Traditional methods such as tree-based structures [8], locality-sensitive hashing [13], and product quantization [19] lay the foundation for scalable search but are generally optimized for static datasets and batch processing.

Graph-based algorithms have become state-of-the-art due to their high recall and low-latency performance. Methods such as *Hnsw* [30], *Nsw* [29], *FreshDiskANN* [43], and *Pyanns* [47] incrementally build navigable small-world graphs and support online updates. These algorithms exhibit strong performance in large-scale settings but often lack robustness under data drift, noisy updates, or concurrent access. Clustering-based methods (e.g., *Pq*, *Ivfpq* [19]) provide good compression and search scalability, yet their tight coupling to distributional assumptions limits their adaptability in dynamic environments.

Despite their strengths, most existing ANNS algorithms are designed for static or minimally dynamic workloads. Their performance under streaming-specific constraints—such as evolving data distributions, ingestion-query overlap, and update efficiency—remains underexplored.

3.2 ANNS Benchmarking Frameworks

Benchmarking has played a central role in standardizing ANNS evaluation. `ANN-Benchmarks` [5] evaluates accuracy and throughput on static datasets using prebuilt indices, making it unsuitable for dynamic updates. `Big-ANN-Benchmarks` [42] extends `ANN-Benchmarks` by supporting billion-scale datasets and adding streaming, OOD, sparse, and filtered query tracks. However, its streaming mode is limited to pull-based, batched ingestion and does not simulate continuous arrival, concurrency, or drift scenarios.

As shown in Table 1, neither benchmark provides mechanisms for modeling high-velocity input, noise injection, concurrent query-update interactions, or system-level profiling—factors that are crucial for evaluating ANNS under open-world streaming conditions. Our work complements these efforts by offering a benchmark tailored to dynamic, high-churn environments.

3.3 System Support for Streaming ANNS

Modern vector databases like Milvus [48], Qdrant, and Faiss integrate ANNS indexing with basic ingestion pipelines, but typically assume background batch updates and do not fully address open-world streaming constraints. Manu [17] proposes a cloud-native architecture for scalable vector search, with emphasis on concurrency and write latency, but abstracts away algorithm-specific behavior.

VectraFlow [27] takes a step further by building a dedicated vector stream processing engine that integrates ingestion, indexing, and query processing. It supports dynamic scheduling and drift-aware operators, aiming for real-time adaptability. However, its focus lies in architectural integration and workload orchestration. In contrast, `CANDOR-Bench` is designed to isolate algorithmic behavior under controlled streaming conditions and can serve as a complementary benchmarking layer for evaluating vector engines like VectraFlow.

4 Methodology

In this section, we present our methodology for evaluating ANNS algorithms under streaming conditions, covering algorithm selection across major indexing paradigms, a diverse set of real-world and synthetic datasets spanning multiple domains and dimensionalities, and CANDOR-Bench—our benchmarking framework that supports both serialized and concurrent execution for fine-grained performance analysis.

Table 2. Summary of Representative ANNS Algorithms

Category	Algorithm Name	Description
Tree-based	*SPTAG* [10]	Space-partitioning tree structure for efficient data segmentation.
LSH-based	*LSH* [13]	Data-independent hashing to reduce dimensionality and approximate nearest neighbors.
	LSHAPG [64]	LSH-driven optimization using LSB-Tree to differentiate graph regions.
	PLSH [46]	Parallel LSH optimized for high-throughput similarity search on data streams.
Clustering-based	*PQ* [19]	Product quantization for efficient clustering into compact subspaces.
	IVFPQ [18]	Inverted index with product quantization for hierarchical clustering.
	ONLINEPQ [54]	Incremental updates of centroids in product quantization for streaming data.
	PUCK [7]	Non-orthogonal inverted indexes with multiple quantization optimized for large-scale datasets.
	SCANN [4]	Small-bit quantization to improve register utilization.
Graph-based	*NSW* [29]	Navigable Small World graph for fast nearest neighbor search.
	HNSW [30]	Hierarchical Navigable Small World for scalable search
	FRESHDISKANN [43]	Streaming graph construction for large-scale proximity-based search with refined robust edge pruning
	MNRU [53]	Enhances *HNSW* with efficient updates to prevent unreachable points in dynamic environments.
	CUFE [2]	Enhances *FRESHDISKANN* with batched neighbor expansion.
	PYANNS [47]	Enhances *FRESHDISKANN* with fixed-size huge pages for optimized memory access.
	IP-DISKANN [55]	Enables efficient in-place deletions for *FRESHDISKANN*, improving update performance without reconstructions.
	GTI [28]	Hybrid tree-graph indexing for efficient, dynamic high-dimensional search, with optimized updates and construction.
	PARLAY-HNSW [32]	Parallel, deterministic *HNSW* for improved scalability and performance.
	PARLAY-VAMANA [32]	Parallel, deterministic *FRESHDISKANN* implementation using *VAMANA* for graph construction, with performance improvement.

4.1 Algorithm Selection

Tree- and LSH-based Methods. *SPTAG* [10] is a representative tree-based algorithm that combines space partitioning with kNN refinement using a lightweight graph structure. For LSH-based methods, we evaluate *LSH* [13], which hashes data points via random projections; *LSHAPG* [64], which augments classic LSH with proximity graphs to improve recall under high-speed queries; and Parallel LSH (*PLSH*)[1] [46].

Clustering-based Methods. This category covers both classical and streaming-optimized quantization schemes. *PQ* [19] and *IVFPQ* [18] represent static approaches that decompose vector space and use inverted indices. *ONLINEPQ* [54] adapts centroid updates for streaming adaptability. *PUCK* [7] introduces non-orthogonal multi-quantization and optimized encoding for large-scale retrieval. *SCANN* [4] leverages quantization-aware CPU register optimizations. For completeness, we also mention *SPFRESH*, which supports efficient in-place disk-based updates—although our evaluation only considers in-memory settings.

[1]To our knowledge, *PLSH* has no public implementation. We implemented it following the paper description, including the two-level partitioning and the delta-table merge design.

Table 3. Summary of Representative Datasets

Category	Name	Description	Dimension	DataSize	QuerySize
Real-world	*SIFT* [23]	Image	128	$1M$	$10K$
	OpenImagesStreaming [3]	Image	512	$1M$	$10K$
	Sun [23]	Image	512	$79K$	200
	SIFT100M [23]	Image	128	$100M$	$10K$
	Msong [23]	Audio	420	$990K$	200
	COCO [25]	Multi-Modal	768	$100K$	500
	Glove [23]	Text	100	$1.192M$	200
	MSTuring [47]	Text	100	$30M$	$10K$
Synthetic	*Gaussian* [38]	i.i.d values	Adjustable	500K	1000
	Blob [35]	Gaussian Blobs	768	$500K$	1000
	WTE [1, 3]	Text	768	$100K$	100
	FreewayML[40]	Constructed	128	$100K$	$1k$

Graph-based Methods. We include eight algorithms in this category. *Nsw* [29] and *Hnsw* [30] serve as foundational approaches. *FreshDiskANN* [43] improves robustness via pruning, and *Pyanns* [47] enhances memory locality with large fixed-size pages. *MNRU* [53] focuses on improving update efficiency by avoiding disconnected points. *Cufe* [2] adds neighbor expansion to strengthen local connectivity. *GTI* introduces a hybrid tree-graph structure to accelerate index updates and construction. For algorithms like *FreshDiskANN* and its variants, we only benchmark their in-memory modes with SSD-related features disabled. We omit *Nsg* and *ELPIS* from our main evaluation because both are offline, batch-constructed graph indexes without an incremental insertion mechanism. In streaming settings they require periodic full rebuilds, and rebuild time dominates end-to-end performance; adding a second "rebuild-on-update" baseline would be redundant and would not change our conclusions.

Special Consideration: Parallel Graph Builders. Parlay-HNSW and *Parlay-Vamana* [32] are parallel, deterministic variants of *Hnsw* and *FreshDiskANN*, respectively. They are used primarily to study concurrency control and offline scalability in large-scale deployments. Since they are not optimized for dynamic updates, we exclude them from streaming benchmarks.

Algorithmic Optimizations. We also explore several orthogonal optimizations: data-level methods like *Locally-Adaptive Vector Quantization* (LVQ) [3], randomized approximations via AdSampling [12], and machine learning-based indexing using Spectral Hashing [52]. These serve to examine whether hybrid techniques can improve resilience to drift, noise, or concurrency.

4.2 Dataset Workloads

To evaluate ANNS performance across diverse real-world and synthetic workloads, we curate a suite of 12 datasets spanning multiple domains, as summarized in Table 3.

4.2.1 Real-world Datasets. These datasets are drawn from widely used benchmarks and public sources [5, 23, 42], and are consistent with prior ANNS studies. For instance, *Glove*, *SIFT*, *SIFT100M*, *Msong*, *Sun*, and *MSTuring* have been extensively used in previous evaluations [23, 42], ensuring compatibility and reproducibility.

COCO [25]: A dataset used to simulate multimodal transitions for evaluating algorithmic robustness under modality shifts. By progressively varying the text-to-image ratio, we emulate scenarios where systems must adapt to changing content types—e.g., from textual product descriptions to image-heavy queries. This helps reveal how indexing and retrieval degrade under heterogeneous data mixtures.

OpenImagesStreaming [3, 20]: A concept drift dataset built from cropped images across evolving semantic classes. It captures distributional changes across time, simulating gradual semantic transitions in visual data streams.

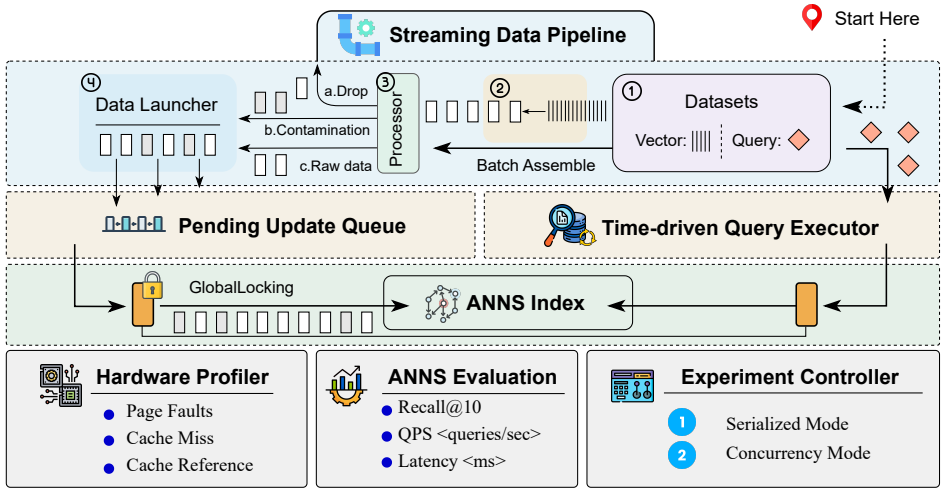


Fig. 1. Open-world ANNS benchmark flowchart

4.2.2 Synthetic Datasets. To explore scalability and robustness under high dimensionality, we also include two synthetic datasets: *Gaussian* and *Blob*. These are generated using `torch.rand` and `sklearn.datasets.make_blobs`, respectively, following prior practices in [3, 23]. They enable controlled evaluation of ANNS algorithms under varying feature dimensionality and cluster distributions. Beyond these fundamental synthetic datasets, we also incorporate more specialized synthetic and custom-constructed datasets to simulate specific real-world challenges related to data distribution shifts:

WTE: A custom dataset simulating distribution shifts in user intent, constructed from the War Thunder wiki [1]. It contains word embeddings for different categories of land vehicles and aircraft, and incorporates hot spot emergence by modulating category activation and controlled contamination.

FreewayML [40]: Constructed datasets for studying various scenarios. Our collection includes *FreewayML-G*, serving as a global baseline dataset, simulating a static, no-drift scenario. Furthermore, the *FreewayML-A1,A2* suite models gradual distribution shifts, including *A1* (linear shift in the 0th dimension mean of the query set) and *A2* (slightly increased variance in the query set). For simulating abrupt distribution shifts, the *FreewayML-B1,B2,B3,B4* suite is used, comprising *B1* (fixed-direction shifts in learn set segments), *B2* (random-direction shifts), *B3* (random shifts with large segments), and *B4* (query set aligned with initial learn set distribution), with query sets for *B1-B3* aligning with the final learn set distribution. Finally, the *FreewayML-C1,C2* suite models recurrent distribution shifts, including *C1* (learn set transitions between state A and B with a small offset, and the query set alternates) and *C2* (recurrent transitions with a large offset).

4.3 Benchmark Testbed

CANDOR-Bench is a modular framework to evaluate in-memory ANNS under dynamic open-world streams. Figure 1 overviews the pipeline.

Streaming Data Pipeline: The data pipeline routes data to core components. The Batch Assembler first groups incoming raw vectors into batches of a configurable size, while the Data Launcher and the Time-driven Query Executor operate with configurable update arrival rate and

query cadence, respectively. These groups are optionally transformed via Raw, Contamination, or Drop. The transformed data are delivered by the Data Launcher to the Pending Update Queue, while query sets are directed to the Time-driven Query Executor. This decouples update and query workloads.

The Pending Update Queue: The Pending Update Queue buffers micro-batches (fixed-size update batches [60, 61]) in a bounded FIFO measured in batches (default capacity is one batch). Admission control is drop-tail: while any batch is pending, arriving micro-batches are rejected and counted as drops. Admitted batches are immutable and applied atomically in arrival order. The same semantics apply in Serialized and Concurrent modes; only the dequeue timing differs. This configuration models arrivals outpacing update capacity and exposes queuing-delay effects under sustained load.

Workload Scheduling and Execution: Update batches from the Pending Update Queue are applied directly to the ANNS index, while queries are generated and executed by the Time-driven Query Executor. The controller has two modes: **(1) Serialized Mode:** The controller alternates between an update-batch phase and a query phase, so only one operation type runs at a time. This enforces batch-level isolation: a query is executed either before an update batch starts or after it completes. Queries do not preempt updates, and updates pause during query phases. **(2) Concurrent Mode:** Updates and queries run simultaneously with multithreading and a configurable read/write ratio. In this mode, we optionally disable system-level readers–writer locks between reads and writes. Without system-level locks, we do not guarantee consistency unless the underlying algorithm natively provides its own safe concurrent read/write support. For algorithms that do not provide such support, we enforce a system-level readers–writer lock at the granularity of update batches: the write lock is held while applying one update batch, and queries execute under read locks between batches. Queries are issued on schedule without waiting for pending update batches; when the write lock is held, queries wait for the current batch to finish.

Fine-Grained Performance Evaluation: A Performance Evaluator runs alongside the query engine without affecting scheduling. It measures (1) **algorithm-level** metrics—query throughput (QPS) and recall over time—reported by the query executor, and (2) **hardware-level** metrics—cache/TLB statistics and page-fault events—collected via a C++ profiler using PAPI [37]. The profiler interfaces with the indexing and analysis code through LibTorch [38] and NumPy [35], and uses SIMD (AVX-512) for sampling and aggregation.

The evaluator runs in a background thread with non-blocking instrumentation and buffered logging. It does not throttle or reschedule queries or updates, and the overhead is kept uniform across algorithms by using the same instrumentation points and sampling rates in every run, preserving relative comparisons.

5 Experiments

In this section, we present a comprehensive evaluation of ANNS algorithms under dynamic open-world streaming conditions. We first introduce our experimental setup, including datasets, evaluation metrics, and baseline configurations (Section 5.1), followed by an overview of general trends observed across workloads (Section 5.3). We then systematically assess how various open-world challenges affect algorithmic behavior, including high-velocity data dynamics (Section 5.4.1), distribution and concept drifts (Section 5.4.2), noise and stochastic loss (Section 5.4.3), and concurrent query-update workloads (Section 5.4.4). Beyond these core challenges, we further explore the effectiveness of lightweight optimization techniques (Section 5.5), highlighting limitations of non-memory-resident indexing. Our primary analysis focuses on single-core, in-memory execution to isolate algorithmic trade-offs without interference from system-level bottlenecks.

5.1 Experimental Setup

Execution Workflow. Each experiment begins with an offline warm-up phase, during which ANNS algorithms preprocess and index the first 50,000 vectors. After this initialization, the system enters the **streaming phase**: remaining vectors are injected into the index at a default rate of 10,000 rows per second, grouped into configurable micro-batches. These batches are submitted as update requests to the indexing engine, simulating high-throughput online ingestion. To prevent system overload, an admission control module discards updates if previous batches are still pending, mimicking real-world resource constraints. Meanwhile, continuous queries are executed once per second, using the full query set from the dataset, and prioritize retrieval over pending index updates. All queries target top-10 nearest neighbors unless otherwise specified. For concurrency experiments, we scale the total event rate proportionally with the number of threads. Each thread independently maintains a 10,000 rows-per-second rate, enabling realistic evaluation of concurrent read-write performance.

Evaluation Metrics. CANDOR-Bench leverages continuous-time metrics to reflect algorithm behavior under dynamic streaming conditions, with these metrics integrating the measurement of retrieval accuracy and system responsiveness. Unlike static evaluations that measure performance post-indexing, our metrics account for index incompleteness and real-time ingestion delays:

- **Continuous Query Recall** ($\bar{R}(q)$): Average fraction of true nearest neighbors retrieved across time. Given recall at query time t as $R(q,t)$, we compute:

$$\bar{R}(q) = \frac{1}{T}\int_0^T R(q,t)\,dt$$

- **Continuous Query Throughput** ($\bar{\tau}_q$): Average queries per second (QPS) sustained throughout the streaming process. Let $\tau_q(t)$ denote instantaneous throughput at time t:

$$\bar{\tau}_q = \frac{1}{T}\int_0^T \tau_q(t)\,dt$$

These metrics provide a unified view of effectiveness and efficiency in evolving open-world scenarios.

Hardware Configuration. All experiments are performed on a single-node server with a 76-core Intel Xeon Platinum 8368Q CPU (152 threads), 512 GiB of DDR4 RAM, and Ubuntu 22.04.5 LTS. All in-memory benchmarks are restricted to single-threaded execution unless otherwise stated.

5.2 Key Findings

We summarize our key findings based on a number of observations in the experiment as follows:

K1 **The recall–throughput frontier fails; no method dominates universally.** Backlog-induced missing entries shrink the searchable corpus, inflating apparent QPS while recall collapses **(O1)**; rankings flip across datasets/modalities, with outcomes hinging on anisotropy, dimensionality, and mix **(O3-O4)**.

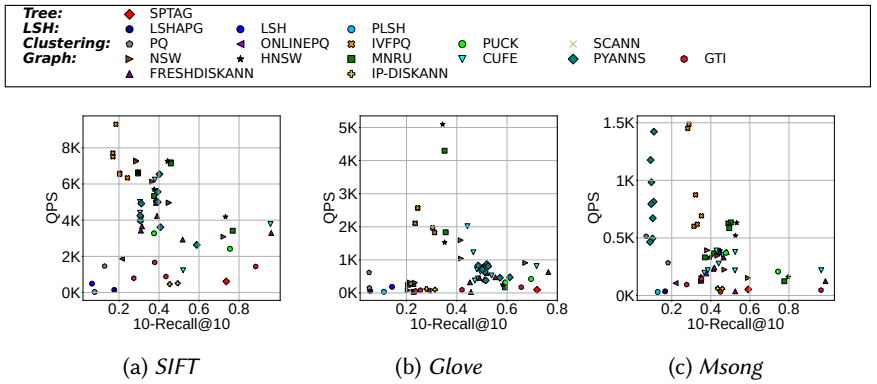
K2 **Freshness is set by rate, micro-batch, and deletions. Recall** collapses once ingestion outpaces update capacity, and higher QPS can be an artifact of a smaller searchable corpus **(O7)**. Smaller micro-batches cut staleness but add fixed per-batch overhead that erodes amortized throughput/CPU efficiency **(O8)**. Bulk deletions reorganize structure and perturb neighborhoods, degrading recall **(O9)**.

K3 **Under hotspots and modality shifts, locality-preserving methods stay comparatively stable; abrupt/high-frequency shifts degrade all.** Under emerging hotspots, locality-sensitive methods remain comparatively stable as hot spots coalesce **(O11)**. When one

modality becomes dominant, hashing/graph-based methods can adapt and improve recall **(O13)**. Under concept drift, recall steadily degrades across methods; *SPTAG* may exhibit delayed recall increases as late-phase updates take effect **(O12)**. All methods struggle under abrupt and high-frequency distribution shifts **(O14)**.

K4 **System stressors dominate end-to-end behavior.** Noisy inputs and stochastic drops corrupt neighborhoods and sharply depress recall **(O15)**. Concurrent read–write execution is effectively write-bound: as the write ratio rises, read QPS falls, and small micro-batches pay heavy synchronization overhead unless batches are large enough to amortize it **(O16)**.

K5 **Tuning and optimizations are narrow-band and ingestion-coupled.** Parameter choices shift the frontier with consistent trade-offs—wider *PQ* codes or denser graphs lift recall at the cost of throughput, while tightening *PUCK*'s radius yields recall gains with minimal throughput loss **(O17)**. Optimizations that add update work can raise query QPS but reduce recall in streaming, so effectiveness must be ingestion-aware rather than offline-only.



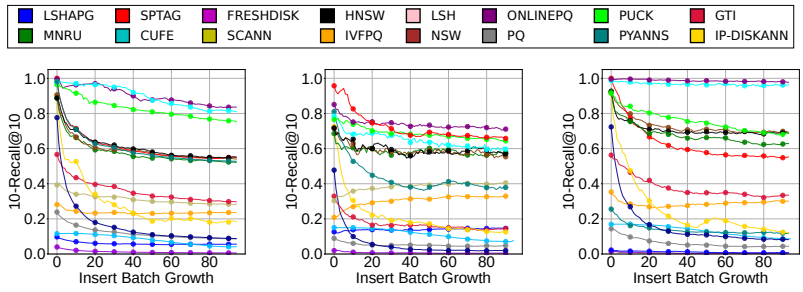
(a) *SIFT* (b) *Glove* (c) *Msong*

Fig. 2. Average Continuous Search QPS vs. Average Continuous Recall across Diverse Parameters

5.3 Evaluating Existing Algorithms on Open-World Vector Data Streams

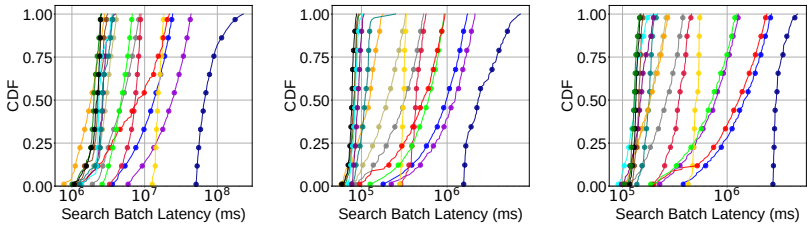
To understand baseline performance under open-world conditions, we evaluate the ANNS algorithms in Table 2 across three representative datasets: *SIFT* (images), *Glove* (text), and *Msong* (audio). These modalities differ in embedding dimensionality, vector distributions, and update behaviors, yielding diverse algorithmic responses. Figure 3–5 present the aggregated recall, latency, and throughput characteristics across these workloads.

O1: Update inefficiency causes accuracy collapse. In contrast to prior static or simplified streaming studies [23, 42], where most ANNS algorithms exhibit a clear concave trade-off curve between recall and throughput, we observe no such regularity in the open-world setting. As shown in Figure 2, algorithms such as *HNSW*, *NSW*, and *MNRU* —previously recognized for high recall—fail to reach 0.6 recall on million-scale data streams like *SIFT*, *Glove*, and *Msong*. More surprisingly, several algorithms (e.g., *IVFPQ*, *PYANNS*, *CUFE*) display near-vertical distributions in the recall-throughput space, indicating that increasing search effort (lower QPS) does not yield recall gains. This breakdown reflects a deeper issue: many algorithms fail to process incoming updates in time, leading to substantial data loss and stale indices. As a result, even with aggressive search configurations, recall does not improve due to missing relevant vectors entirely



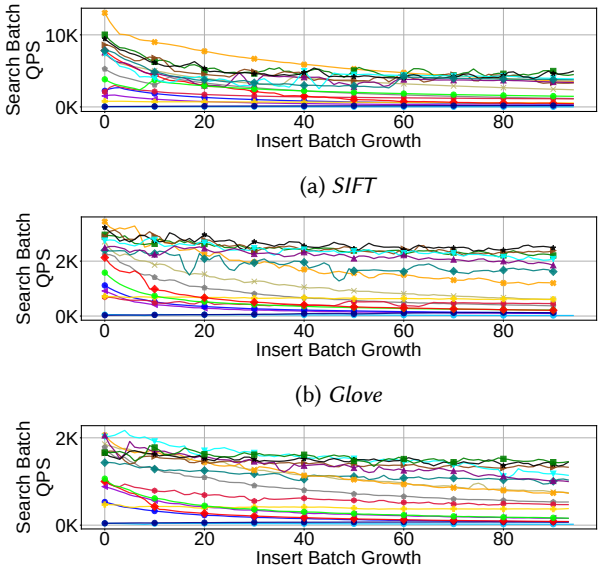
(a) *SIFT* (b) *Glove* (c) *Msong*

Fig. 3. Continuous Recall vs. Insert Batch Growth



(a) *SIFT* (b) *Glove* (c) *Msong*

Fig. 4. Batch Search Latency CDF



(a) *SIFT*

(b) *Glove*

(c) *Msong*

Fig. 5. Continuous Search QPS vs. Insert Batch Growth

O2: Adaptivity sustains recall; static partitions falter. Figure 2 reveals that algorithms such as *LSHAPG*, *IVFPQ*, *PQ*, and *LSH* consistently fail to surpass 0.6 recall across most datasets. This shortfall largely stems from their reliance on static data partitioning—whether via clustering or hashing—based on the initial index construction. As new data arrives, these static structures lack

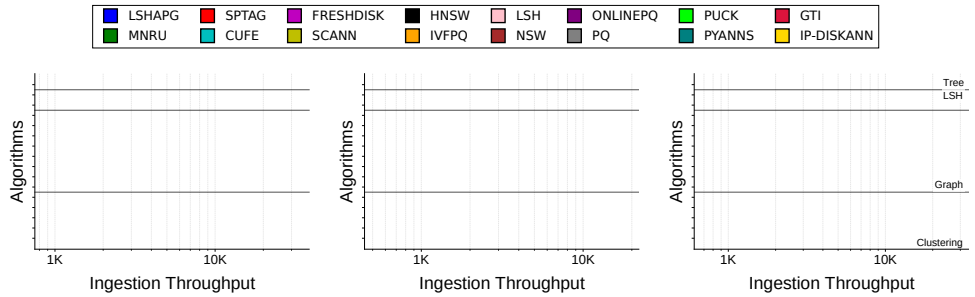
mechanisms for continuous adaptation, and retraining is often infeasible under real-time constraints. In contrast, algorithms that support online structural adjustment exhibit noticeable improvements. *onlinePQ* outperforms *PQ* due to its incremental centroid updates, while *Puck* achieves over 0.6 recall across datasets by leveraging adaptive clustering during the streaming process. These results underscore the importance of real-time adaptability: in dynamic open-world environments, ANNS algorithms must incorporate mechanisms for continuous structural evolution to maintain retrieval quality. Although*PLSH* allows overlapping index building and querying in a distributed dataflow design, the paper assigns objects and buckets to processing stages via fixed mapping functions and does not discuss online rebalancing. Our workload uses dense, real-valued embeddings; in our environment, we find that *PLSH* is not a good fit.

O3: Update-robust methods sustain recall; most decay under continuous ingestion. Figure 3 presents continuous recall as a function of incremental data insertion batches. We observe that algorithms from the *FreshDiskANN* family (except *Pyanns*), clustering-based methods, and *Lshapg* maintain relatively stable recall over time, exhibiting minimal fluctuations as more data enters the index. This stability indicates their inherent robustness to dynamic updates. In contrast, most other methods show a pronounced decline in recall with continued data ingestion, suggesting they are more sensitive to incremental data additions. Such algorithms, despite initially high recall, struggle to sustain performance under continuous data influx, highlighting a critical challenge in dynamic open-world streaming scenarios.

O4: Dataset anisotropy and dimensionality dictate latency tails and QPS stability. Figure 4 shows the *Cumulative Distribution Functions* (CDFs) of batch latency, where steeper curves indicate better robustness to latency variations. On the *SIFT* dataset, latency distributions remain tightly concentrated within one order of magnitude, suggesting stable batch processing. In contrast, the *Glove* and *Msong* datasets exhibit broader latency distributions, often spanning two or more orders of magnitude, signaling frequent high-latency outliers. These disparities arise primarily from intrinsic dataset characteristics. *Glove* embeddings are anisotropic and unevenly distributed in high-dimensional space, forming complex semantic manifolds with challenging outliers. Such a structure makes efficient partitioning and nearest neighbor retrieval inherently difficult, resulting in extended latency tails. Similarly, *Msong* 's higher dimensionality (420 dimensions compared to *SIFT* 's 128) amplifies the "curse of dimensionality," significantly impacting search efficiency and introducing greater latency variability.

Additionally, Figure 5 illustrates how continuous search throughput (QPS) generally decreases with growing data batches across algorithms. Notably, *Hnsw* and *Nsw* exhibit substantial QPS fluctuations on the *Glove* and *Msong* datasets. Their dynamic in-memory graph updates struggle with *Glove* 's intricate distribution and *Msong* 's dimensional complexity, producing unstable search paths and pronounced throughput instability. Conversely, the *FreshDiskANN* family achieves more stable QPS due to its robust indexing mechanism. These results demonstrate that dynamic and evolving data characteristics significantly influence algorithmic stability and performance. To consistently achieve high recall and throughput in open-world streaming scenarios, ANNS algorithms must adapt in real-time, efficiently incorporate new data, and maintain structural robustness under diverse dataset distributions.

O5: Ingestion upper bound as a leading indicator of continuous throughput. Figure 6 reports steady-state ingestion throughput under a continuous insert stream at saturation. Across *SIFT*, *Glove*, and *Msong*, families with predominantly local updates attain higher bounds than structures requiring global repair; within this pattern, *onlinePQ* leads among quantization methods, *Lsh* is



(a) *SIFT* (b) *Glove* (c) *Msong*

Fig. 6. Ingestion Throughput Upper Bound (ops/sec)

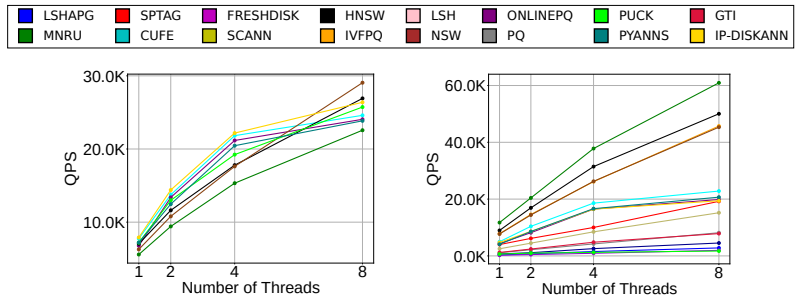
competitive mainly on high-dimensional *Msong*, graph indexes are slower overall, and the tree-based *SPTAG* is slowest. These results indicate that saturation throughput is governed by per-update cost rather than query efficiency.

O6: Thread scaling benefits reads far more than writes. As Figure 7 shows, ingestion increases with 1–8 threads but flattens around 4, with methods that rely on heavier global structures showing little gain. Search scales close to linearly for graph-based indexes and only modestly for globally partitioned ones. We do not report multi-threaded ingestion for *IVFPQ*, *SCANN*, *LSH*, *PQ*, *ONLINEPQ*, *SPTAG*, *LSHAPG*, or *GTI* because their reference implementations do not support parallel inserts. Reproducing multi-threaded ingestion for these methods would require substantial reengineering of internal data structures and update logic. To ensure fairness and consistency across all evaluated methods, we deliberately avoid introducing custom modifications.

5.4 Handling of Open-World Challenges

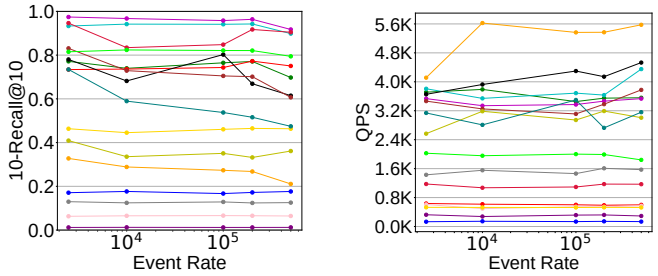
5.4.1 Challenges of Handling High-Velocity Data Dynamics. We evaluate ANNS algorithm performance under varying event rates and micro-batch sizes using the *SIFT* dataset, where parameters are optimized to maximize recall. This analysis first investigates insertion efficiency under high event rates, followed by the impact of micro-batch size adjustments and deletion operations—both critical in high-velocity open-world streams.

O7: When ingestion outpaces updates, recall collapses; higher QPS reflects a smaller searchable corpus, not true efficiency. Figure 8a illustrates algorithm recall as a function of data ingestion rate, ranging from 2,500 to 500,000 vectors per second. Most ANNS algorithms experience notable recall deterioration when event rates surpass their update processing capacity. Algorithms such as *NSW*, *HNSW*, *MNRU*, and *PYANNS* exhibit significant recall drops beginning at approximately 2,500 rows/sec. In contrast, algorithms including *FRESHDISKANN*, *CUFE*, and *PUCK* sustain higher event rates, managing up to around 200,000 rows/sec before encountering pronounced recall degradation. This divergence arises primarily from differing efficiencies in real-time data ingestion, directly affecting data completeness and resulting in substantial data loss at high velocities. Notably, the observed increases in query throughput (QPS) for algorithms such as *IVFPQ*, *HNSW*, *MNRU*, and *NSW* are artifacts of insufficient data ingestion, where reduced indexing overhead leads to a deceptively smaller search space. Real-world streaming systems often operate at even higher data velocities [22], underscoring the urgent need for algorithms optimized to handle extreme insertion rates without compromising retrieval accuracy.



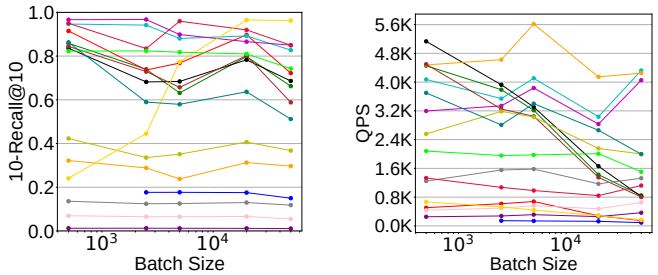
(a) Ingestion (b) Search

Fig. 7. Average Continuous QPS vs. Number of Threads



(a) Continuous Recall (b) Continuous QPS

Fig. 8. Tuning Event Rate from 2500 to 500000 ($rows/s$)

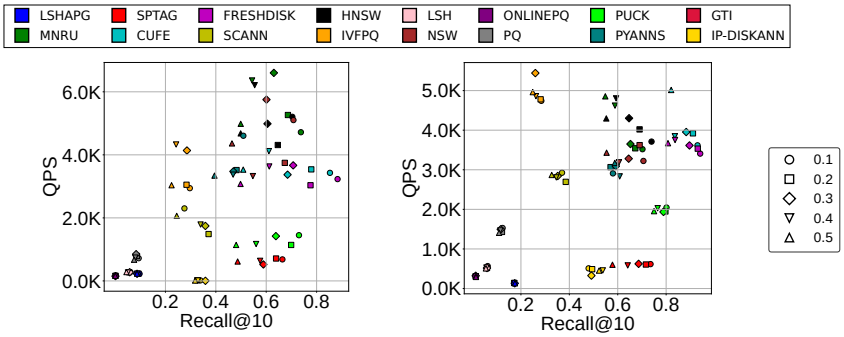


(a) Continuous Recall (b) Continuous QPS

Fig. 9. Tuning Micro-batch Size from 500 to 50000

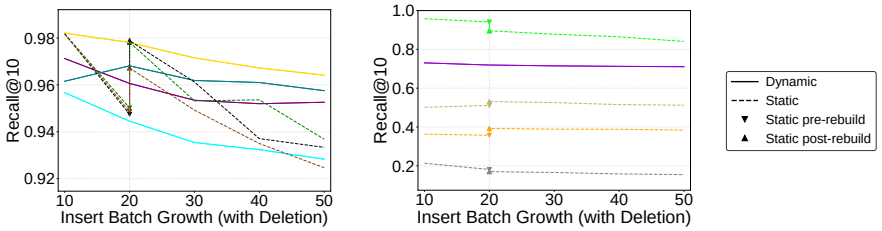
O8: Batch size controls recall; small batches raise freshness with higher overhead, large batches lower overhead but increase staleness. Figure 9 reports recall as the micro-batch size varies from 500 to 50 000 vectors. Smaller batches yield fresher results because the index is updated more often, but they incur higher per-batch overhead and amplify data loss when ingestion cannot keep pace. Larger batches amortize the update cost, yet the index becomes stale between updates. Several distinct behavioural patterns emerge: **(1) Inverse U-shaped recall:** For *SPTAG* and *MNRU*, recall rises as the batch size grows to ~20 000 (overhead becomes less dominant and data loss falls), then drops sharply as larger batches delay updates and return increasingly outdated results. **(2) U-shaped recall:** *NSW*, *HNSW*, and *PYANNS* first lose accuracy with moderate batch sizes—indicating unrealised freshness potential—then recover when batches are large enough that update overhead is negligible relative to query time. **(3) Monotonic decline:** *FRESHDISKANN* and

Cufe show steadily decreasing recall as batch size increases, demonstrating heightened sensitivity to index staleness when updates are infrequent.



(a) Bulk Deletion Proportions (b) Batch Deletion Proportions

Fig. 10. Handling Bulk and Batch Deletions. Tuning deletion proportion from 0.1 to 0.5.



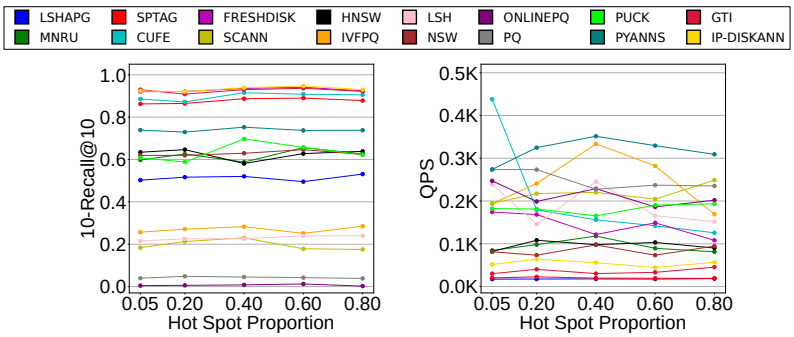
(a) Graph-based (b) Clustering-based

Fig. 11. Static vs. Dynamic under Streaming Inserts with Deletions

O9: Bulk deletions cause steep recall loss for real-deletion indexes; tombstones degrade more gently as the search space grows; batching deletions mitigate the impact. Deletion operations in ANNS have been widely adopted in industry [42], but their impact has only been studied on a limited number of algorithms [43, 57, 64] without formal study in open-world streaming settings. Among the 19 algorithms we evaluate, only *Sptag*, *FreshDiskANN*, *Cufe*, *Pyanns*, *Puck*, *Lshapg*, and *MNRU* offer specific deletion solutions. For algorithms without dedicated deletion mechanisms, we employ a *tombstone and expansive search* strategy, which ignores deleted points and continues the search without incrementing any counters.

We evaluate two deletion workloads: First, bulk deletion, where a deletion request is submitted at the end of the streaming process to remove a proportion of data at the head of the dataset, and recall is only computed by continuous queries after the bulk deletion request is submitted. The results are shown in Figure 10a. Algorithms that perform real deletions (*FreshDiskANN*, *Puck*, *Sptag*) suffer the steepest recall decline as the deletion fraction grows. Consolidated deletions force costly graph rewiring and memory reclamation [43], making timely index recovery infeasible. Tombstone-based methods (*Ivfpq*, *Hnsw*, *Nsw*) degrade more gently because they avoid immediate structural changes, albeit at the expense of a growing search space. *MNRU* mitigates this by maintaining a backup graph for previously unlinked nodes, trading extra update overhead for higher post-deletion recall. Second, batch deletion, where a deletion request is submitted after each insertion request to remove a proportion of data at the head of the corresponding batch.

The results are shown in Figure 10b. When deletions are amortised across micro-batches, *Cufe* and *FreshDiskANN* absorb the cost well and retain high recall. *Pyanns*, however, struggles: its 2 MB huge-page layout makes frequent re-organisation expensive. *MNRU* again tracks *Hnsw* 's recall while sustaining higher QPS, confirming the utility of its auxiliary index for incremental deletions. Notably, while *GTI* aims to provide specific deletion solutions, our experiments reveal a critical design instability: it crashes when the deletion proportion exceeds 1%. In summary, deletion workloads reveal a second axis of the open-world trilemma: schemes that guarantee strong consistency via physical removals incur heavy update overhead, whereas logical tombstones preserve throughput but risk accuracy drift as the index grows stale. Effective streaming ANNS must balance these opposing costs—still an open problem, as evidenced by the mixed results of recent "deletion-friendly" designs such as *GTI* and *IP-Diskann*.



(a) Continuous Recall (b) Continuous QPS

Fig. 12. Tuning proportion of hot spots from 0.05 to 0.8

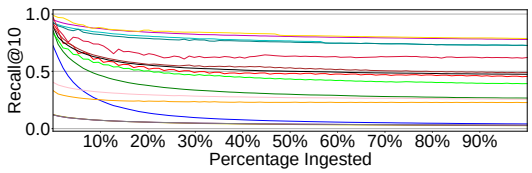


Fig. 13. Recall degrades for most algorithms as distribution evolves on *OpenImagesStreaming*

O10: Rebuilds can restore recall for graph indexes after small deletion bursts; for clustering indexes, rebuild offers limited benefit. Figure 11 compares static and dynamic maintenance under streaming inserts with proportional deletions. For graph-based methods in Figure 11a, the static variant experiences a noticeable recall drop after a deletion burst, while a one-shot offline rebuild can largely recover recall (pre-/post-rebuild markers in the figure). However, rebuild is an offline operation and incurs additional rebuild overhead. When deletions are amortized into micro-batches, dynamic implementations with local rebalancing (e.g., *FreshDiskANN*-style) maintain recall close to the post-rebuild level without requiring a global rebuild. In contrast, for clustering-based methods in Figure 11b, dynamic variants provide only limited improvement over static, and a post-hoc rebuild yields little additional gain under the same workload, suggesting that frequent rebuilds are generally not cost-effective in this setting.

5.4.2 Challenges of Handling Data Drifts. As outlined in Section 2.2, we explore data drifts in open-world streaming settings, including emerging hot spots, concept drift, multi-modal transitions, and data sifts, using the *WTE*, *OpenImagesStreaming*, *COCO*, and *FreewayML* datasets respectively.

O11: As hot spots intensify, concentrated locality raises throughput with little recall cost; adaptive structures hold up while static partitions lag. We use the *WTE* dataset to simulate controlled distribution drift by varying the probability of emerging hot spots. The first 50K data points are structured as "[*subject*] [*predicate*] [*object*]", where both *subject* and *object* are randomly selected aircraft names from War Thunder [1], and *predicate* is sampled from NLTK verbs [34]. In the subsequent stream, the *subject* is gradually replaced with nouns absent from the initial set, introducing a shift toward new semantic topics. Meanwhile, queries follow a fixed template—"What [*predicate*] [*object*]?"—and remain unaffected by the new hot spots, thereby creating a recall challenge as the stream evolves.

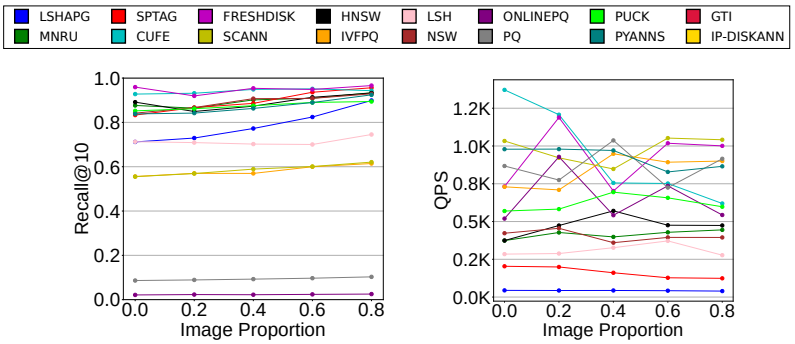
As shown in Figure 12, increasing the hot spot probability from 0.05 to 0.8 yields consistent QPS improvements for most graph-based algorithms (*Cufe*, *FreshDiskANN*, *Pyanns*, *Hnsw*, *MNRU*) with minimal recall degradation. This behavior reflects their ability to rapidly form and exploit localized hubs as hot spot topics become dominant [51]. When hot spot instances are sparse (e.g., 0.05), they are diffused throughout the index, making them harder to retrieve; in contrast, at higher proportions, they coalesce into searchable clusters, accelerating lookup. *Sptag* prioritizes recall at the expense of QPS. Most clustering-based methods, bound to initial clusters, struggle even when distributions remain stable. In contrast, *Puck* maintains both QPS and recall through real-time coarse cluster adaptation. Finally, *Lshapg* and *Lsh* deliver decent recall across all drift levels, suggesting LSH-based methods are well-suited for handling emergent topics.

O12: Concept drift steadily reduces recall across methods; structural adaptivity slows the decline but does not prevent it. To evaluate algorithm robustness under concept drift, we use the *OpenImagesStreaming* dataset [3], constructed by encoding cropped images from Google's Open Images [20] using CLIP [41]. The resulting stream features a gradual evolution of semantic classes, simulating real-world concept shifts. Figure 13 reports the recall trends throughout the stream. Most algorithms exhibit a continuous decline in recall as semantic drift accumulates. Graph-based methods with robust edge pruning—such as *FreshDiskANN*, *IP-Diskann*, *Cufe*, and *Pyanns* —show slower degradation. Their pruning strategies incrementally link new concepts to existing graph structures, preserving connectivity. In contrast, *Hnsw*, *MNRU*, and *Nsw* experience steep recall drops due to aggressive pruning that fails to accommodate evolving neighborhoods. *Lshapg* suffers the fastest recall loss, even from the beginning, as its LSB-tree initialization does not generalize to new concepts and its graph layer exacerbates the mismatch. Clustering-based methods like *Puck*, *Ivfpq*, and *Lsh* degrade more gradually, benefiting from space-partitioning that is less brittle than learned structures. *Sptag* exhibits delayed recall increases, suggesting that its update inefficiency causes stale index structures to dominate until late-phase updates take effect. Notably, *Nsw* performs slightly better than *Hnsw*, likely due to its flat structure that avoids overshooting evolving regions, unlike the hierarchical jumps in *Hnsw*. Overall, concept drift remains a severe challenge across algorithm classes, with no method demonstrating consistently robust recall under sustained semantic evolution.

O13: As one modality becomes dominant, recall improves only when the index adapts online; otherwise it stays flat or declines. To examine algorithm adaptability under multi-modal transitions, we use the *COCO* dataset, initially streaming 50K caption-only (text) vectors. During subsequent streaming, image embeddings are gradually introduced alongside captions, with the image proportion varied from 0 to 0.8. Query vectors—drawn from a separate set—include a similar mixture of images and captions. Figure 14 reports the resulting recall trends.

As shown in Figure 14a, algorithms based on navigable small world graphs (*Nsw*, *Hnsw*, *Pyanns*) and locality-sensitive hashing (*Lshapg*) exhibit steadily increasing recall as the image ratio grows. In contrast, most other methods maintain low, flat recall, with *GTI* showing a mild decline as image

dominance increases. Graph-based methods benefit from self-organizing structures that adapt to modality shifts by forming new clusters for incoming image data, enabling efficient traversal. Hash-based algorithms leverage the high-dimensional structure of image embeddings, which facilitates bucket formation with minimal contamination from prior textual data. As image density increases, recall improves due to better alignment between query and index modalities.



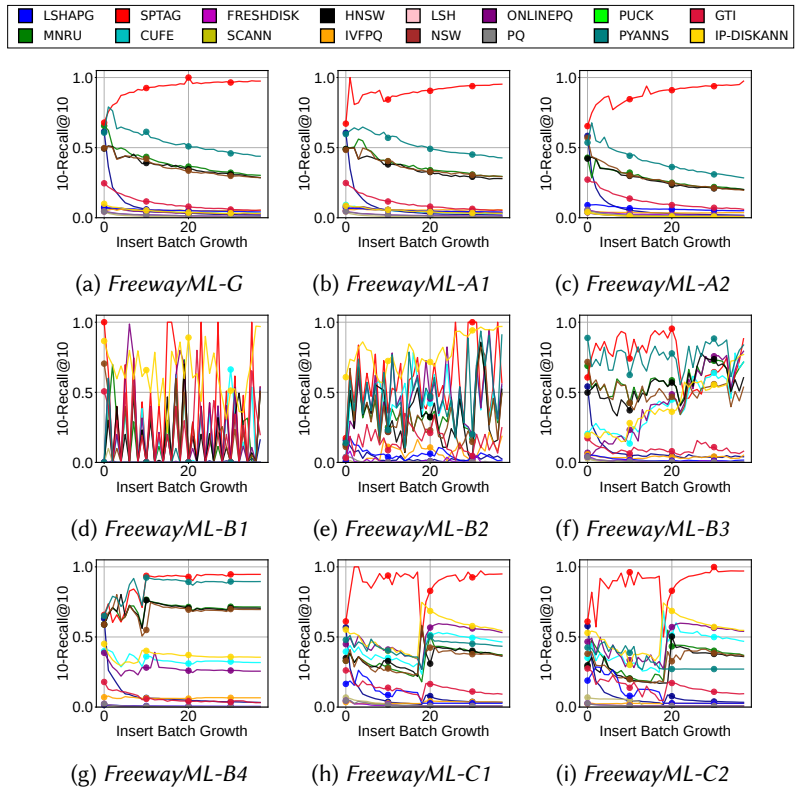
(a) Recall (b) QPS

Fig. 14. Tuning the proportion of randomly mixed images during the streaming phase from 0 to 0.8

These results suggest that under asymmetric multi-modal drift—where a new modality gradually dominates—both graph and hash structures can adapt effectively, provided the incoming modality exhibits strong internal consistency. However, conventional clustering or static partition-based methods struggle without explicit reconfiguration, highlighting a core challenge for modality-agnostic streaming systems.

O14: Abrupt and high-frequency shifts drive recall toward zero; reducing shift frequency mitigates the drop. In evaluating *FreewayML* datasets, we observed significant divergences in the continuous recall trend as a function of insert batch growth. For static (G) and gradual distribution shift scenarios (A1, A2), as depicted in Figures 15a, 15b, and 15c respectively, most algorithms exhibited relatively stable or slightly decreasing recall trends. This behavior reflects their general performance as data volume increases without drastic distribution changes, suggesting that index structures might become marginally "outdated" or suboptimally optimized over time, yet within acceptable limits. In contrast, *SPTAG* displayed a distinctive "abnormal" trend in these scenarios, with its recall significantly increasing as the data volume grew. This may be attributed to its underlying Space Partition Tree and Graph structures' dynamic update and online construction capabilities, enabling it to progressively optimize index quality with data accumulation and thereby improve query recall. It is crucial to note that this trend of recall increasing with data growth is not universally observed across all datasets; for instance, similar growth behavior may not be seen in traditional benchmarks conducted on static large-scale datasets, as referenced in 3, thereby highlighting the unique dynamic concept drift simulation characteristics inherent in these synthetic datasets.

Conversely, Figures 15d and 15e illustrate abrupt, high-frequency shifts (B1, B2), occurring every 5000 samples with fixed or random directions. Here, most algorithms demonstrated extreme sensitivity, leading to sharp recall fluctuations and often plummeting precipitously to near-zero. This starkly reveals the limitations of conventional ANNS algorithm indices in rapidly adapting to entirely new data patterns when confronted with severe concept drift. However, in Figures 15f and 15g, where the shift frequency was reduced (every 30,000 samples for B3, B4), the magnitude of

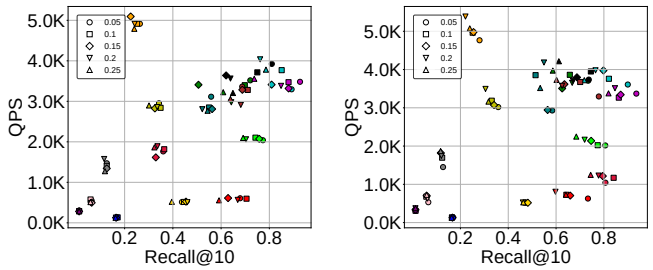


(a) *FreewayML-G* (b) *FreewayML-A1* (c) *FreewayML-A2*

(d) *FreewayML-B1* (e) *FreewayML-B2* (f) *FreewayML-B3*

(g) *FreewayML-B4* (h) *FreewayML-C1* (i) *FreewayML-C2*

Fig. 15. Continuous Recall vs. Insert Batch Growth across Synthetic Random Datasets



(a) Tuning Noise Intensities (b) Tuning Stochastic Drop Rates

Fig. 16. Handling Noises and Stochastic Drops. Tuning noise/drop proportion from 0.05 to 0.25.

recall drop was mitigated, with no algorithms falling to zero, suggesting that longer stable periods facilitate algorithm adaptation. Notably, in 15g (B4), the query set reverted to the initial distribution, severely testing the algorithms' ability to recall "historical patterns."

Finally, Figures 15h and 15i simulate recurrent shift patterns (C1, C2) where data distributions alternate between two states. In 15h (C1, with a small shift), most algorithms adapted reasonably well to the alternating changes, exhibiting relatively minor recall fluctuations. Conversely, in 15i (C2, with a large shift of 50.0), many algorithms displayed extremely volatile recall, often plummeting dramatically. This underscores the significant challenge of recognizing and reusing historical patterns under extreme distribution changes. Throughout both the B and C series of mutation

patterns, *SPTAG* consistently demonstrated a tendency for recall improvement or relative stability, in some cases outperforming other algorithms.

5.4.3 Challenges of Handling Noise and Stochastic Loss.

O15: Noise and random losses corrupt neighborhood structure and sharply reduce recall; methods relying on edge pruning or adaptive cluster assignment are especially vulnerable. Noisy data and packet losses are common in real-time streaming applications, such as sensor monitoring, and they can have a devastating impact [9, 24]. In this section, we evaluate the robustness of ANNS in the presence of noise at varying intensities (defined as the proportion of Gaussian points) or extra stochastic data loss during the streaming process.

Unlike the distribution drifts, where the drifted data remain semantically interpretable, noisy data introduces a radically different distribution, which proves to be more disruptive for ANNS as shown in Figure 16a. For graph-based algorithms, despite achieving higher recall, *FRESHDISKANN*, *PYANNS*, and *CUFE* experience a more significant performance drop compared to *HNSW*, *MNRU*, and *NSW*. This suggests that the robust edge pruning process in the former algorithms is less capable of handling disruptive distributions, as noisy data is more closely linked to the original data. Although enjoying the benefits of robust edge pruning in most cases, adaptive adjustments of the process would be of greater benefit to *FRESHDISKANN*. A similar phenomenon can be interpreted from Figure 16b, imposing stochastic data loss could be more destructive to *FRESHDISKANN*, *PYANNS*, and *CUFE* as the robust pruning procedure heavily relies on the completeness of original data. On the other hand, *PUCK*'s recall significantly drops as well as noise intensity increases, indicating the vulnerability to noises of its adaptive cluster assignment. The performance degradation of specific strategies necessitates more robust and comprehensive algorithms to open-world streaming challenges.

5.4.4 Challenges of Handling Concurrent Queries.

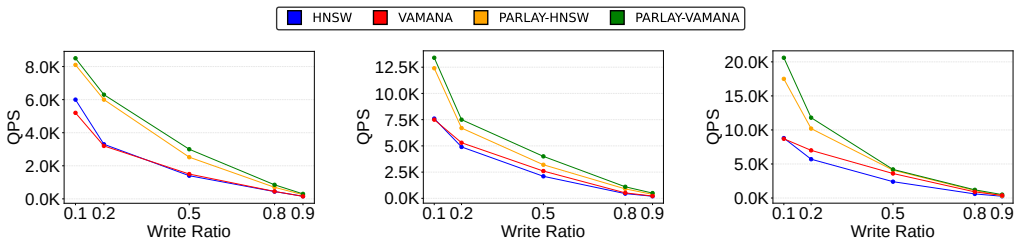
O16: In concurrent read–write, throughput is write-bound; higher write ratios lower read QPS, and small micro-batches waste work on synchronization while larger batches raise QPS. This section analyzes ANNS algorithms under concurrent read-write conditions. Most ANNS algorithms are not optimized for such scenarios, lacking inherent consistency guarantees without external read-write locks. The ParlayANN series (*PARLAY-HNSW* and *PARLAY-VAMANA*) is an exception, employing an asynchronous, snapshot-based construction. This design ensures read operations do not access modified graph structures and write operations apply only to snapshots.

We compare *HNSW* and *VAMANA*, the foundational algorithms upon which *PARLAY-HNSW* and *PARLAY-VAMANA* are built. Parameters are adjusted to achieve an approximate 95% Recall@10 in continuous scenarios. Our experimental setup strictly enforces strong consistency, requiring all prior batch modifications to be immediately visible to subsequent read requests. For ParlayANN, leveraging its snapshot-based design, we tested without system-level read-write locks, implementing specific modifications to ensure strong consistency. Conversely, for the base *HNSW* and *VAMANA*, we employed read-write locks to enforce this. Experiments use micro-batch sizes of 100, 200, and 500, simulating "heavy write," "balanced," or "heavy read" scenarios based on specific write-to-read ratios (e.g., 90% writes, 50% writes, 10% writes) on the *SIFT* dataset.

Results in Figure 17 show Vamana consistently outperforms *HNSW* across all batch sizes. *VAMANA*'s single-layer graph construction yields faster build times and quicker insertion speeds. As illustrated in Figure 4, while *HNSW* generally offers superior batch query speeds, its update operations introduce significant overhead that reduces overall QPS in concurrent read-write scenarios. This creates a "bottleneck effect" where the slower update process dictates overall throughput, even when queries are batched. Predictably, as the write ratio increases, the overall read QPS declines across

all algorithms, underscoring the performance impact of write contention. Furthermore, ParlayANN algorithms, while strong for large-scale offline index construction, are designed to amortize parallel overhead with large internal batches. Consequently, in streaming scenarios with very small, frequent micro-batches, their performance may lag behind *Vamana* and *Hnsw* due to insufficient batch sizes to cover this overhead.

Secondly, larger batch sizes notably increase QPS for all algorithms. This stems from batch processing reducing per-operation overhead by amortizing synchronization costs and improving resource utilization. This highlights that traditional read-write isolation schemes, when faced with frequent, small streaming ANNS updates, are particularly susceptible to substantial overhead and contention, leading to suboptimal throughput.



(a) Batch Size : 100 (b) Batch Size : 200 (c) Batch Size : 500

Fig. 17. Search QPS vs. Write Ratio across Different Batch Size

5.5 Investigating Impacts of Optimizations

O17: Optimization must be ingestion-aware; extra update work lifts query speed but lowers recall. In this section, we assess the impact of ANNS algorithmic optimizations and then profile system-level optimizations under open-world streaming settings. Algorithmic optimizations include computational and machine learning approaches, evaluated using *Hnsw* and *Lsh*, respectively. We then profile hardware performance with `CANDOR-Bench` 's profiler on *FreshDiskANN* and its system-level optimizations, *Cufe* and *Pyanns*.

Computation optimizations include data compression methods, *Locally-Adaptive Vector Quantization* (LVQ), and randomized computation techniques, AdSampling. These methods introduce overhead during updates and trade accuracy for improved search efficiency. We evaluate their impacts on *Hnsw*. As shown in Table 4, LVQ achieves up to 2.9× the search throughput of unoptimized *Hnsw*. However, on the real-world dataset *SIFT*, LVQ experiences significantly degraded recall, likely due to insufficient data ingestion due to extra update overhead, which weakens its ability to accurately capture distribution patterns. AdSampling provides a modest efficiency improvement at the cost of slightly reduced recall, likely due to data loss induced by additional update overhead. These issues underscore the critical need to enhance ingestion efficiency to fully unlock the potential of ANNS optimizations in open-world streaming environments. On the other hand, machine learning optimization, Spectral Hashing [52] (used with *Lsh*), requires an offline training phase to approximate its loss functions. However, it fails to improve either accuracy or efficiency, as machine learning struggles in streaming scenarios, resulting in more chaotic bucket assignments and a more exhaustive search across all buckets. Algorithmic optimizations should consider the trade-off complicated by the update efficiency and the destructive impacts from various distributions of streaming data in open-world.

Table 4. Evaluation results of algorithmic optimizations

Algorithms		Recall @ 10			Search Throughput		
		SIFT	*WTE*	*OpenImagesStreaming*	*SIFT*	*WTE*	*OpenImagesStreaming*
Comp.	*Hnsw*	0.67	0.75	0.52	6986.25	1469.43	2554.71
	Hnsw LVQ	**0.12**	0.53	0.43	**10930.83**	**4323.21**	**7365.72**
	Hnsw AdSampling	0.62	0.69	0.41	7012.59	1620.97	2918.42
ML	*Lsh* w/o ML	0.063	0.74	0.28	569.93	336.95	185.82
	Lsh w/ ML	0.012	0.007	0.002	166.65	50.24	140.68

Table 5. Actionable Guidance for ANNS under Dynamic Streaming Workloads

Core Requirement	**Application Scenario**	**Observation**	**Recommended Algorithm**
Maximize Write Throughput	Real-time Rec. Systems / High-Speed Data Pipelines	O5, O7, O8	*FreshDiskANN*, *Cufe*, *Puck*
Maximize Long-Term Recall (Gradual Drift)	Long-term Content Rec. / User Interest Tracking	O12	*FreshDiskANN*, *Cufe*
Robustness & Recovery (Abrupt Drift)	Breaking News Stream / Malicious Traffic Detection	O14	None Consistently Robust
Query Robustness	IoT Sensors / Industrial Monitoring (Noisy Data)	O15	*Hnsw*, *Nsw*, *MNRU*
Maintain Index Compactness	Content Expiration / Privacy Data Deletion	O9, O10	*FreshDiskANN*, *Cufe*, *IP-Diskann*
Overall Throughput	High-Concurrency Search Engine / Write-Intensive Workloads	O16	*Vamana*
Adaptation to New Clusters	Social Media Trend Monitoring / Emerging Event Tracking	O11, O13	LSH-based, Graph-based

5.6 Guidance and Discussion

Table 5 offers a compact selector for in-memory streaming ANNS, mapping typical streaming requirements to representative methods.

6 Conclusion

In this paper, we introduce `CANDOR-Bench`, a novel benchmark framework that enables plug-and-play evaluation of existing ANNS algorithms on open-world streaming environments without requiring code changes. Our extensive experiments reveal that ANNS algorithms exhibit inconsistent performance patterns and scalability challenges compared to static or simplified streaming settings, especially when considering the constraints and dynamics of in-memory environments. Specifically, factors such as micro-batching, data drifts, and other open-world streaming challenges complicate the trade-offs between update and search. This work highlights directions for future ANNS research: (1) adaptive and robust algorithms, (2) dynamic streaming-centric systems, and (3) efficient and concurrent update mechanisms. To gain a more holistic understanding of streaming ANNS performance, we also plan to extend our analysis to diverse disk-based ANNS algorithms, and investigate the unique optimizations and challenges posed by GPU-accelerated and distributed environments.

Acknowledgments

This work is supported by National Key Research and Development Program of China (No. 2023YFB4502300) and NSFC-RGC under Grant No.62461160333.

References

[1] 2024. New | War Thunder Wiki. https://wiki.warthunder.com/
[2] AbdelrahmanMohamed129. 2023. GitHub - AbdelrahmanMohamed129/DiskANN at farah. https://github.com/AbdelrahmanMohamed129/DiskANN/tree/farah
[3] Cecilia Aguerrebere, Mark Hildebrand, Bhati Ishwar Singh, Theodore Willke, and Mariano Tepper. 2024. Locally-Adaptive Quantization for Streaming Vector Search. *arXiv:2402.02044* (2024). doi:10.48550/arxiv.2402.02044
[4] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache Locality is not Enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *Proceedings of the VLDB Endowment* 9, 4 (2015), 288–299. doi:10.14778/2856318.2856324
[5] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2018. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. arXiv:1807.05614 [cs.IR] https://arxiv.org/abs/1807.05614
[6] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–31. doi:10.1145/3709693
[7] baidu. 2023. GitHub - baidu/puck: Puck is a High-Performance ANN Search Engine. https://github.com/baidu/puck
[8] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM* 18, 9 (1975), 509–517. doi:10.1145/361002.361007
[9] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. 2021. SAND: Streaming Subsequence Anomaly Detection. *Proceedings of the VLDB Endowment* 14, 10 (2021), 1717–1729. doi:10.14778/3467861.346786
[10] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-Efficient Billion-Scale Approximate Nearest Neighbor Search. *arXiv:2111.08566* (2021). doi:10.48550/arxiv.2111.08566
[11] Kunal Dahiya, Deepak Saini, Anshul Mittal, Ankush Shaw, Kushal Dave, Akshay Soni, Himanshu Jain, Sumeet Agarwal, and Manik Varma. 2021. DeepXML: A Deep Extreme Multi-Label Learning Framework Applied to Short Text Documents. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 31–39. doi:10.1145/3437963.3441810
[12] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27. doi:10.1145/3589282
[13] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of 25th International Conference on Very Large Data Bases*, Vol. 99. 518–529.
[14] Jim Gray and Andreas Reuter. 1992. *Transaction Processing: Concepts and Techniques*. Elsevier.
[15] 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 and Data Mining*. 311–320. doi:10.1145/3219819.3219885
[16] Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik PA Lensch. 2022. GGNN: Graph-Based GPU Nearest Neighbor Search. *IEEE Transactions on Big Data* 9, 1 (2022), 267–279. doi:10.1109/TBDATA.2022.3161156
[17] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, and Zhenshan Cao Yanliang Qiao Ting Wang Bo Tang Charles Xie Liu, Frank. 2022. Manu: A Cloud Native Vector Database Management System. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3548–3561. doi:10.14778/3554821.3554843
[18] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547. doi:10.1109/TBDATA.2019.2921572
[19] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. doi:10.1109/TPAMI.2010.57
[20] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. 2020. The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale. *International Journal of Computer Vision* 128, 7 (2020), 1956–1981. doi:10.1007/s11263-020-01316-z
[21] L. Lamport. 1979. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Comput.* 28, 9 (Sept. 1979), 690–691. doi:10.1109/TC.1979.1675439
[22] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyuan Ni, Ning Wang, and Yuan Chen. 2018. The Design and Implementation of a Real Time Visual Search System on JD E-commerce Platform. In *Proceedings of the 19th International Middleware Conference Industry*. 9–16. doi:10.1145/3284028.3284030
[23] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate Nearest Neighbor Search on High Dimensional Data—Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488. doi:10.1109/TKDE.2019.2909204
[24] Ju Lin, Yun Wang, Kaustubh Kalgaonkar, Gil Keren, Didi Zhang, and Christian Fuegen. 2021. A Time-Domain Convolutional Recurrent Network for Packet Loss Concealment. In *Proceedings of IEEE International Conference on*

Acoustics, Speech and Signal Processing (ICASSP). 7148–7152. doi:10.1109/ICASSP39728.2021.9413595

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2014. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312* (2014). doi:10.48550/arxiv.1405.0312

[26] Hang Liu, H. Howie Huang, and Yang Hu. 2016. iBFS: Concurrent Breadth-First Search on GPUs. In *Proceedings of the 2016 International Conference on Management of Data*. 403–416. doi:10.1145/2882903.2882959

[27] Duo Lu, Siming Feng, Jonathan Zhou, Franco Solleza, Malte Schwarzkopf, and Uğur Çetintemel. 2025. VectraFlow: Integrating Vectors into Stream Processing. In *Proceedings of the 14th Conference on Innovative Data Systems Research*.

[28] Ruiyao Ma, Yifan Zhu, Baihua Zheng, Lu Chen, Congcong Ge, and Yunjun Gao. 2025. GTI: Graph-Based Tree Index with Logarithm Updates for Nearest Neighbor Search in High-Dimensional Spaces. *Proceedings of the VLDB Endowment* 18, 4 (2025), 986–999. doi:10.14778/3717755.3717760

[29] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate Nearest Neighbor Algorithm Based on Navigable Small World Graphs. *Information Systems* 45 (2014), 61–68. doi:10.1016/j.is.2013.10.006

[30] Yu. A. Malkov and D. A. Yashunin. 2018. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2018), 824–836. doi:10.1109/TPAMI.2018.2889473

[31] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. Introduction to Information Retrieval. (2008).

[32] Magdalen Dobson Manohar, Zheqi Shen, Guy Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2024. ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Edinburgh, United Kingdom) *(PPoPP '24)*. Association for Computing Machinery, New York, NY, USA, 270–285. doi:10.1145/3627535.3638475

[33] Marqo. 2024. Context is All You Need: Multimodal Vector Search with Personalization. https://www.marqo.ai/blog/context-is-all-you-need-multimodal-vector-search-with-personalization Accessed: 2025-03-25.

[34] NLTK. 2009. Natural Language Toolkit — NLTK 3.4.4 Documentation. https://www.nltk.org/

[35] Numpy. 2024. NumPy. https://numpy.org/

[36] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs. arXiv:2308.15136 [cs.DS] https://arxiv.org/abs/2308.15136

[37] PAPI. 2023. PAPI. https://icl.utk.edu/papi/

[38] PyTorch. 2023. PyTorch. https://pytorch.org/

[39] qdrant. 2023. GitHub - qdrant/qdrant: High-Performance, Massive-Scale Vector Database and Vector Search Engine for the Next Generation of AI. https://github.com/qdrant/qdrant

[40] Zheng Qin, Zheheng Liang, Lijie Xu, Wentao Wu, Mingchao Wu, Wuqiang Shen, and Wei Wang. 2025. FreewayML: An Adaptive and Stable Streaming Learning Framework for Dynamic Data Streams. In *Proceedings of 2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1690–1703. doi:10.1109/ICDE65448.2025.00130

[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskeve. 2021. Learning Transferable Visual Models from Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, Vol. 139. PMLR, 8748–8763. doi:10.48550/arXiv.2103.00020

[42] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, Dmitry Baranchuk, Edo Liberty, Frank Liu, Ben Landrum, Mazin Karjikar, Laxman Dhulipala, Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Yizhuo Chen, Weiguo Zheng, Zihao Wan, Jie Yin, and Ben Huang. 2024. Results of the Big ANN: NeurIPS'23 Competition. arXiv:2409.17424 [cs.IR] https://arxiv.org/abs/2409.17424

[43] 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:2105.09613* (2021). doi:10.48550/arXiv.2105.09613

[44] Vinicius M. A. Souza, Farhan A. Chowdhury, and Abdullah Mueen. 2020. Unsupervised Drift Detection on High-Speed Data Streams. In *Proceedings of 2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 102–111. doi:10.1109/BigData50022.2020.9377880

[45] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. 2018. The End of an Architectural Era: It's Time for a Complete Rewrite. In *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*. 463–489.

[46] 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 (2013), 1930–1941. doi:10.14778/2556549.2556574

[47] veaaaab. 2023. GitHub - veaaaab/DiskANN at bigann23_streaming. https://github.com/veaaaab/DiskANN/tree/bigann23_streaming

[48] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627. doi:10.1145/3448016.345755

[49] Kongtao Wang, Zhiyong Chen, and Hui Liu. 2014. Push-Based Wireless Converged Networks for Massive Multimedia Content Delivery. *IEEE Transactions on Wireless Communications* 13, 5 (2014), 2894–2905. doi:10.1109/TWC.2014.040914.131422

[50] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978. doi:10.14778/3476249.3476255

[51] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph- and Tree-Based Indexes for High-Dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. *IEEE Data Engineering Bulletin* 46, 3 (2023), 3–21.

[52] Yair Weiss, Antonio Torralba, and Rob Fergus. 2008. Spectral Hashing. *Advances in Neural Information Processing Systems* 21 (2008).

[53] Wentao Xiao, Yueyang Zhan, Rui Xi, Mengshu Hou, and Jianming Liao. 2024. Enhancing HNSW Index for Real-Time Updates: Addressing Unreachable Points and Performance Degradation. *arXiv:2407.07871* (2024). doi:10.48550/arxiv.2407.07871

[54] Donna Xu, Ivor Tsang, and Ying Zhang. 2018. Online Product Quantization. *IEEE Transactions on Knowledge and Data Engineering* 30, 11 (2018), 2185–2198. doi:10.1109/tkde.2018.2817526

[55] Haike Xu, Magdalen Dobson Manohar, Philip A. Bernstein, Badrish Chandramouli, Richard Wen, and Harsha Vardhan Simhadri. 2025. In-Place Updates of a Graph Index for Streaming Approximate Nearest Neighbor Search. *arXiv:2502.13826* (2025). doi:10.48550/arXiv.2502.13826

[56] 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 ACM Symposium on Operating Systems Principles*. 545–561. doi:10.1145/3600006.3613166

[57] Zhaozhuo Xu, Weijie Zhao, Shulong Tan, Zhixin Zhou, and Ping Li. 2022. Proximity Graph Maintenance for Fast Online Nearest Neighbor Search. *arXiv:2206.10839* (2022). doi:10.48550/arXiv.2206.10839

[58] Song Yu, Shengyuan Lin, Shufeng Gong, Yongqing Xie, Ruicheng Liu, Yijie Zhou, Ji Sun, Yanfeng Zhang, Guoliang Li, and Ge Yu. 2025. A Topology-Aware Localized Update Strategy for Graph-Based ANN Index. *arXiv:2503.00402* (2025). doi:10.48550/arXiv.2503.00402

[59] Yuanhang Yu, Dong Wen, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2022. GPU-Accelerated Proximity Graph Approximate Nearest Neighbor Search and Construction. In *Proceedings of 2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 552–564. doi:10.1109/ICDE53745.2022.00046

[60] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. 423–438. doi:10.1145/2517349.2522737

[61] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. 2012. Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing* (Boston, MA) *(HotCloud'12)*. USENIX Association, USA, 10.

[62] Hamed Zamani, Fernando Diaz, Mostafa Dehghani, Donald Metzler, and Michael Bendersky. 2022. Retrieval-Enhanced Machine Learning. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2875–2886. doi:10.1145/3477495.3531722

[63] Weijie Zhao, Shulong Tan, and Ping Li. 2020. Song: Approximate Nearest Neighbor Search on GPU. In *Proceedings of 2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1033–1044. doi:10.1109/ICDE48307.2020.00094

[64] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *Proceedings of the VLDB Endowment* 16 (04 2023), 1979–1991. doi:10.14778/3594512.3594527

[65] Konstantinos Zioutos, Haridimos Kondylakis, and Kostas Stefanidis. 2023. Healthy Personalized Recipe Recommendations for Weekly Meal Planning. *Computers* 13, 1 (2023), 1. doi:10.3390/computers13010001

Received July 2025; revised October 2025; accepted November 2025