Zonal Graph Quantization Optimizing Memory-Performance Trade-off in Vector Search

Nathan Aldyth Prananta Ginting
Faculty or Engineering and Technology
Sunway University
Subang Jaya, Malaysia
22099865@imail.sunway.edu.my

Jordan Chay Ming Hong
Faculty of Engineering and Technology
Sunway University
Subang Jaya, Malaysia
11111111@imail.sunway.edu.my

Jaeden Ting YiYong
Faculty of Engineering and Technology
Sunway University
Subang Jaya, Malaysia
11111111@imail.sunway.edu.my

***Abstract*—With the rapid development of Artificial Intelligence, new database types like vector databases have become essential. The Approximate Nearest Neighbour Search (ANNS) is a very important algorithmic operation in these systems. Graph-based algorithms are noted to be best in indexing amongst other algorithm due to their speed. Hierarchical Navigable Small World (HNSW) and other cutting-edge algorithms are faster and more accurate for searching, but they use a lot of memory and cost a lot to build, which makes them hard to scale for datasets with billions of records. This study analyses the memory-performance trade-off by comparing HNSW with partitioning-based approaches and presents a novel hybrid framework. Our methodology involves a comparative study of a proposed zonal framework, which partitions the dataset into smaller clusters and constructs independent, lightweight HNSW graphs within each zone to reduce complexity. The findings discovered throughout this research phase are expected to be able to demonstrate that a hybrid approach can reduce index memory size while maintaining recall rates competitive with standard HNSW. Our research results may highlight the viability of hybrid structures for creating scalable, resource-efficient vector search solutions, which addresses a pain point and may solve a critical challenge in deploying large-scale AI systems which handle heavy flow of data.**

Keywords—Vector Database, Approximate Nearest Neighbor Search (ANNS), HNSW (Hierarchical Navigable Small World), Indexing Algorithm, High-Dimensional Data, Performance analysis

I. Introduction

A. Background

The ability to perform efficient similarity search and analysis on vector data's with high dimensions is a fundamental in modern applications spanning from recommendation systems all the way to RAG (Retrieval Augmented Generation) in Large Language Models (LLMs). Approximate Nearest Neighbour Search (ANNS) algorithms are essential for these tasks as they provide balance between search speed as well as accuracy. Now these existing algorithms have 3 different base such as Tree based, Quantization based as well as the Graph Based methods. Recent times have indicated that graph algorithms have become the de-facto approach demonstrating a superior trade-off between the accuracy and efficiency in various benchmark tests which have been conducted, thus this motivates the creation and development of this paper and its proposed solution.

B. Problem Staetement

Despite the success of these algorithms, the leading graph-based algorithm such as the HNSW is exposed to a crucial drawback which are high memory consumption as well as the expensive compute price that this algorithm needs to run. Now with the current multi-layered architecture and its numerous edges, the memory footprint of the HNSW may grow inefficiently with the dataset size which makes it costly and expensive to run in applications with billions of scale. On the contrary, the quantization-based methods that partitions the data such as IVF are memory efficient when compared to the HNSW itself but has a higher chance of failure when pushed to achieve the high recall rates of graph-based algorithms.

C. Problem Significance

As vector datasets grow exponentially, the reliance on large memory usage continuously increases the cost of the system operations and may provide substantial obstacles when deploying the ANNS at scale. One of the biggest challenge in database engineering is the direct impact between the economic viability with the scalability of the system (AI/DB/VDB) itself which increases the significance from being a mere academic problem to a real-life application. Thus, developing these memory efficient indexing solution is a substantial approach to generalize the access to large-scale vector search technology.

D. Report Scope

This report will cover the design, and the experimental evaluation as well as implementation of the HNSW, IVF and our proposed ZGQ framework. The analysis will be constrained to the use of the in-memory performance using metrics such as query latency, recall@k, as well as the index build time and the final index size. Bigger and broader related challenges which involves disk-based search and the filtered search with Metadata constraints are considered out-of-scope for this specific particular study but may be discussed and talked about for future possible research reference

II. Literature Review

A. Introduction and Selection Criteria

The literature review section is going to examine the recent advancements as well as the development of the Approximate Nearest Neighbor Search (ANNS) algorithms, with a targeted topic of graph based as well as partitioning-based methods in high-dimensional vector databases. The main purpose of the literature review is to analyze the strength as well as the weaknesses of the current most effective techniques specifically the trade-offs between search efficiency (which includes its latency as well as the Query Per Second (QPS)), its accuracy (for recall purposes), as well as its resource consumption (such memory footprint and its index build time). Hence understanding these trade-offs is a needed measure to come up and develop a scalable solution for its billions of datasets.

The articles which are reviewed are selected based on the relevance of their topic with the ANNS indexing algorithms with the majority of the topic covering grounds such as HNSW, IVF as well as other proposed hybrid approaches. The priority goes as conferences articles and peer-reviewed journals which have been published within the past 5 years (late 2020-2025) from publishers such as ACM Digital library, Cornell University's ArXiv as well as the IEEE Explore for papers. With a total of 20 articles to form the base of knowledge for this research.

B. Thematic Analysis of ANNS Algorithms

The reviewed articles are organized into separate themes to view the major approaches as well as the challenges in ANNS. The themes can be found as follows:

1. Foundational Graph-Based ANNS (**HNSW, NSG**): Graph based methods have been rising in popularity due to their superior accuracy and efficiency ratio. The HNSW itself constructs a multi-leveled graph which enables searches to start in a scattered upper level, and as search goes on to navigate to denser lower levels which achieves a logarithmic search complexity. The key innovation lies in their separation of links by distance using a heuristic to select the nearest neighbors. However, HNSW comes with challenges revolving around memory and time due to the nature of the algorithm to create multiple-leveled layers which then adds more time to construction. Navigating Spreading-out Graph (NSG) is another high-performance graph method that uses a similar RNG-based edge selection strategy but builds upon a K-Nearest Neighbor Graph (KNNG) foundation, aiming for a smaller index size. A comprehensive survey by Wang et al. provides a detailed taxonomy and component-based analysis of various graph methods, finding that higher graph quality doesn't always yield better search performance

2. Scaling challenges (Zonal HNSW, DiskANN, SPANN, XN-Graph): Discovering and understanding the scaling limitations of in-memory graphs is one of the main targets of this research. Algorithms like Zonal HNSW tackles billion-scale datasets by partitioning the fed data into separate "zones" and building smaller, independent HNSW graphs within each zone, reducing memory and enabling parallel search. Another algorithm called DiskANN became a first in disk-based search by storing the graph index and full vectors on SSDs, keeping only small sized vectors in memory which then uses beam search to mitigate I/O latency. Another recent disk-based search like SPANN offers an alternative disk-based approach by partitioning the dataset and using a small in-memory graph on cluster centroids to guide search within partitions loaded from disk. Despite its ability and simplicity on papers, SPANN can suffer in multi-threaded scenarios due to I/O friction. More recently, XN-Graph proposes a novel disk-optimized graph structure with extended neighborhood coverage to reduce search hops (I/O operations) and an "In-Memory First Search" strategy to minimize CPU idle time while waiting for SSD reads. A recent development in disk-based search is the FreshDiskANN which extends DiskANN to handle streaming data updates. Similarly, SPFresh focuses on incremental updates for SPANN

3. Efficient Vector Quantization (IVF-PQ, MRQ): Recent research and development in vector quantization techniques are tailored to reduce memory consumption and usage as well as to accelerate computations across vector distances. One of the methods is called Product Quantization (PQ) which is a foundational method that decomposes vectors into sub-vectors, quantizes each independently, and allows for fast distance estimation using look-up tables. It is often combined with an Inverted File (IVF) index, where data are partitioned into clusters, and search is restricted to a few clusters near the query. The Faiss library provides highly optimized implementations of IVF-PQ, often accelerated by GPUs. Minimization Residual Quantization (MRQ) improves upon quantization flexibility and efficiency by analyzing data distribution via PCA, splitting vectors into high-variance (quantized) and low-variance (residual) parts.

4. Algorithm Optimizations and Considerations (FINGER, Cache Efficiency, GPU, Learned Routing, Filtered Search): More thorough research focuses on optimizing specific aspects of ANNS. With the algorithm of Fast Inference for Graph-based Approximate Nearest Neighbor Search (FINGER) aims to accelerate the graph traversal (inference) phase. Whereas other optimization methods explore low-level optimizations like graph reordering for better CPU cache efficiency. Now GPU acceleration is also crucial for practical performance, we can implement this by using libraries like Faiss and dedicated algorithms like SONG demonstrating significant speedups. Some works have investigated these approaches using machine learning to learn better routing strategies on graphs. A major practical challenge is Filtered ANNS (FANNS), where queries involve both vector similarity and metadata constraints. Shi et al. provide a benchmark and taxonomy for FANNS, categorizing methods like UNG into filter-then-search, search-then-filter, or hybrid approaches

C. *Article Reviews*

This section provides a complete review of each of our chosen articles for reference and as our base of knowledge, where each article is reviewed to identify their Problem Statement, Methodology, as well as their key findings,

Malkov and Yashunin (2020) addresses that the currently available Approximate Nearest Neighbor Search (ANNS) algorithm such as Navigable Small World (NSW) suffers from a complexity and performance decrease when the given data's are clustered, thus the author then introduces a new algorithm namely the Hierarchical Navigable Small World or what we know today as the HNSW algorithm, which constructs a multi-leveled graph by randomly inserting elements which then decays the maximum layer exponentially by separating the links by distance scale. They have also discovered that the hierarchical search which starts at a scattered top layer and descends greedily to a denser bottom layer has achieved a true $O(\log N)$ or logarithmic complexity which proves that the HNSW algorithm is more efficient than its predecessor (NSW).

On another note, Wang Et Al. (2021) has identified a lack of an orderly organized component level comparison for the wide array of graph-based ANNS algorithms. The study has analyzed over 13 representative graph algorithms, which introduces a new classification of base graphs which we have known today as Directed Graph (DG), Relative Neighborhood Graph (RNG), K-Nearest Neighbor Graph (KNNG), and the more common Minimum Spanning Tree (MST). The most crucial finding from this research is how it accomplished to find that an algorithm's advantage relies highly on its Dataset. An example of this is how NSW outperforms HNSW on certain specific datasets where as HNSW is outperforms NSW on some. The research has also found that increasing graph quality does not equate to better search performance

Prokhorenkova and Shekhovtsov (2020) discovered that there is a lack of a robust and rigor theoretical guarantee that supports the strong performance of graph-based ANNS, more specifically in the low-dimensional area which can be denoted as $d \ll \log N$. The paper presented a theoretical analysis of the greedy-like algorithms on nearest-neighbor graphs. Their analysis proves that by properly adding long-range links, they are able to reduce the number of steps required for searching to $O\log^2 N$. This research has also introduced us to Beam Search which on paper improves the search accuracy and allows for graph to have lower number of layers which further saves memory.

Chen et al. (2021) faces the in-memory ANNS issue faced by algorithms like HNSW where it was discovered that the algorithm is deemed to be expensive for data with the amount reaching billions. The paper proposes a new algorithm called SPANN, which is a hybrid approach to cluster data centroids within its memory whilst also keeping the full vector data on SSD. The proposed algorithm uses a balanced hierarchy-based clustering to ensure the length of the vectors which then adds the boundaries to the vector to multiple clusters for better recall. The study found that the SPANN algorithm is determined to be faster than the disk-based graph called as DiskANN achieving over 90% recall on large volume datasets

Another study conducted by Manohar et al. (2024) addressed the scaling issues that current parallel graph-based algorithms face, this paper determined that most graph-based algorithms rely too heavily on locks as well as other sequential processes which leads to the algorithm being slow and inefficient when being ran on modern multi-core CPUs. This paper introduces ParlayANN. The algorithm consists of libraries of deterministic and parallel algorithms, where the algorithm uses Prefix Doubling (an act to insert points in parallel batches on an immutable reference of the graph itself). The algorithm implementation was shown to be efficient and is well-scalable on multi-core processors which outperforms its predecessors such as HNSW as well as DiskANN

A similar attempt at creating a zone-based algorithm has been done before by Akhil and Sivashankar (2025) where they would find the current HNSW algorithm would degrade when given a large lump of data. Their approach partitions the Dataset into smaller local zones using K-Means where each zone is going to hold and maintain an independent HNSW graph. Whilst their approach improves the scalability of the HNSW-based algorithm, it introduces memory overhead from managing several graphs.

Moving on to hardware focused optimization, Zhang et al. (2025) This paper wrote on the I/O latency bottleneck in which disk-based ANNS algorithm faces. A new method focusing on wider neighborhood coverage is introduced as. eXtended Neighborhood Graph which is often referred as XN-Graph. The paper focuses on implementing a Boundary-Adaptive partition which resulted in effective parallel graph construction complimented with an In-memory search. The paper claimed to have achieve a 4000 QPS with 90% recall.

Most ANNS faces poor performance with the vector-query are embedded with metadata constraints, where according to Gollapudi et al. (2023) the standard processing pipeline fails when given low specificity metadata. The paper proposed a new graph construction algorithm namely the FilteredVamana and StitchedVamana. These algorithms have an aim to modify the graph to be label (metadata)-aware. The edges are appended and removed based off the proximity and the labels that the vector holds. These algorithms resulted in high-throughput whilst also achieving high-recall in SSD based storage.

Another challenge of ANNS has been discovered on Cai et al. (2024) paper where it included the FilteredVamana and StitchedVamana algorithm which was developed a year prior. The paper introduces us to Unified Navigating Graph (UNG) which aims to resolve the lack of versatility (may be denoted as: $fail\ if\ |f_q| > 1$) that current algorithms have. The algorithm separates each vector into groups according to their labels; it then creates a Label Navigating Graph (LNG) which is a superset of Directed Acrylic Graph (DAG) which encodes the relationship within these label sets. The structure of UNG guarantees versality and fidelity.

A study conducted by Xu et al. (2023) has identified a challenge in maintaining the vector index performance when dealing with incremental updates within the vectors, this paper specifically highlights how naïve-in place modifications to cluster-based indices such as SPANN may obstruct the data distribution which results in a costly rebuild. The paper introduced and proposed SPFRESH which is an upgrade and expansion of SPANN which uses a system named as lightweight incremental RE-balancing (LIRE) to perform a re-balancing whenever a partition split happened.

An efficiency issue has also been identified by Gong et al. (2025) when using vector databases like Milvus to monitor streaming systems like Flink. It has been identified that detached architectures can cause high and low latency and inefficient vector updates. In order to resolve this issue, the author introduced VStream in the paper. The paper claimed that VStream itself is an integrated vector streaming engine that uses a dynamic partitioning to its advantage using Locality-Sensitive Hashing (LSH) as well as hierarchical storage. Thus, by combining the vector search directly into the streaming aspect, the proposed algorithm claimed to have achieved 373 times faster performance with lesser energy being used by CPU as well as lesser memory when compared to traditional detached systems

An often-identified issue in graph-based ANNS such as HNSW is how the inefficient memory access patterns that can further lead to poor CPU cache as well as frequent cache misses. Coleman et al. (2022) addresses this issue by creating a graph reordering technique that reorganizes the node layouts within its memory to ensure that frequently accessed nodes are stored near with each other. By implementing this optimization technique, the algorithm has achieved a 40% faster average query latency despite an added additional indexing overhead.

Another inefficiency in ANNS arises from the Distance Comparison Operation (DCO) this dominates most of the ANNS run time by performing a full and complete $O(D)$ computation even for negative candidates that will be discarded upon completion of the compute. Gao and Long (2023) addressed this issue with ADSampling, which is a randomized approach to apply offline random orthogonal transformation to all of the vectors as well as incrementally sampling the dimensions during the query run-time. The paper adapted a hypothesis testing where the algorithm adaptively stops comparison when it identified a candidate as a negative. Thus, the complexity of DCO is reduced from $O(D)$ down to $O(\log D)$.

Chen et al. (2023) aims to target the inefficiency in graph search. The paper observed that most distance computations does not influence the search update and thus it can be approximated. With this observation, the author proposed Fast Inference (FINGER), which is a method that aims to speed-up search phase by replacing an entire distance computation to a fast approximation. The method outlined decomposes the distance of $||q-d||\left(\frac{2}{2}\right)$ relative to its center node C it then uses an LSH-based method to approximate the $q_{res}^{t}{}^{d_{res}}$ term by approximating the angle between residual vectors. The approximation technique was proven sped-up HNSW query times by 20% to 60% compared to other recent optimized implementations.

Similarly to Prokhorenkova and Shekhovtsov (2020), Peng et al. (2023) this paper also focuses on the lack of theoretical performance guarantee in most proximity graphs, which gives either no guarantees or that it is an impractical one, (e. g $q \in D$). The paper simulated the environment of a practical setting where the Query q is close to its nearest neighbor $\bar{V}$ (i.e., $(q,\ v) < \tau$ and it also proposes the $\tau - MG$. The paper indicates that a greedy search on the $\tau - MG$ is guaranteed to find an exactly 1-N where if the condition holds, the search is going to have a lower time complexity of $O(n^{\frac{1}{m}}(\ln n)^2)$. This paper also proposes a variant called $\tau - MNG$ for practicality.

Another flaw with existing graph methods $\tau - MG$ was discovered with Xie et al. (2025)'s study. The paper discovered that the method requires a predefined global τ which resulted in an impracticality, and on top of that the paper also indicated that there are no graph method available as of its writing provides a theoretical guarantee for k-NN search where the condition is ($k > 1$). To solve this problem, the paper proposed what they called a Labeled Monotonic Graph (LMG). This method uses an approach where the edges are labeled with the minimum required τ for their availability in the database, and afterwards the search is then performed by using two methods:

1. Adaptive Search, which is going to find the 1-NN by dynamically increasing the query dependent (τ) from 0.
2. Refinement Search, which is then going to find the remaining k- 1 neighbors by then exploring the 1-NN's area's

This approach is the first of its kind in the vector query optimization field to provide a theoretical guarantee to find the exact intended k-NN for any query without needing a predefined variable such as τ.

Another bottleneck has been identified in Vecchiato et al. (2024). The paper focuses on the routing phase of cluster-based ANNS (IVF), where the selection of clusters seems to cause a bottleneck since it is mostly done by a simple heuristic (the one commonly used is distance to centroid). The paper rearranged this problem to be a Learning to Rank (LTR) Problem. Thus, the paper proposes a simple linear function to replace the static immutable centroids which further trains it to optimize queries. The paper displayed that the LTR approach consistently improves the top 1 accuracy of cluster-based search.

Yang et al. (2025) handles a different topic than the previous papers we have discussed, where this paper focuses on the super-linear index construction of SOTA-based graphs (Such as NSW, HNSW) the paper denotes that this can limit the scalability of the algorithm noting that the methods to accelerate the indexing and construction comes at the price of search performance. The paper analyses on how the construction is done, it also identified the k-Candidate neighbor set acquisition (k-CNA) phase as the main reason of the bottleneck. Thus, the paper proposes a step called "Refinement-before-search" the paper involves a pruning's strategy to the initial KNNNG before the k-Can search phase. Whereas for the HNSW it also replaces the node-by-node insertion with a global layer. The paper denotes that this solution. Hs accelerated index construction by up to 5.6 times faster compared to NSW and 4.6 faster to HNSW

Zhao et al. (2020) addresses the challenge of that a typical graph-based ANNS algorithms. Most graph-based ANN algorithms are deemed to be efficient and is effective to run on the CPUs, but despite this it faced difficulties when adapting to GPUs environment due to the iterative nature as well as the execution dependencies when traversing through a graph which hinders parallel processing. In order to mitigate this issue, the paper proposes an approach named as Search On neighborhood Graph (SONG). The proposed approach aims to enable parallel processing with a target to simplify the distance computation phase of the search. The SONG approach introduces a new ANN-specific optimization for GPUs with the purpose of eliminating dynamic memory allocation as well as to reduce the memory consumption. The paper achieved speed-ups of 50-180x compared to conventional CPU-based HNSW even outperforming FAIS

TABLE I. COMPARISON TABLE

Reference	Problem Statement	Algorithm/Methodology	Key Findings
[1] Malkov & Yashunin (2020)	NSW algorithm performance degrades with clustered data and has polylogarithmic complexity.	Hierarchical Navigable Small World (HNSW): Builds a multi-layer graph with exponentially decaying layer probability for element insertion, separating links by distance scale. Search starts from the top layer and descends greedily.	HNSW achieves O(log N) search complexity, outperforming NSW, especially on clustered data.
[2] Wang et al. (2021)	Lack of systematic comparison and component-level analysis for diverse graph-based ANNS algorithms.	Comprehensive survey and experimental comparison of 13 graph algorithms. Introduced a taxonomy based on base graphs (DG, RNG, KNNG, MST) and analyzed algorithms via fine-grained components.	Algorithm performance is highly dataset-dependent. Increased graph quality does not necessarily lead to better search performance.
[3] Prokhorenkova & Shekhovtsov (2020)	Lack of theoretical guarantees for graph-based ANNS performance, especially in low dimensions ($d \ll \log n$).	Rigorous theoretical analysis of greedy graph search. Investigated adding long-range links (shortcuts) and using Beam Search (dynamic candidate list).	Properly added long-range links can reduce search steps to $O(\log^2 N)$. Beam Search improves accuracy and allows for potentially sparser graphs near the query.
[4] Chen et al. (2021)	High memory cost of in-memory ANNS algorithms (like HNSW) for billion-scale datasets.	SPANN: Hybrid approach storing centroids in memory and full vectors/posting lists on SSD. Uses balanced hierarchical clustering and augments lists by adding points in cluster closures for better recall.	SPANN outperforms DiskANN on billion-scale datasets, achieving high recall (>90%) with lower latency in single-thread scenarios.
[5] Manohar et al. (2024)	Scalability limitations of existing parallel graph-based ANNS implementations due to locks and sequential bottlenecks on multi-core CPUs.	ParlayANN: Library of deterministic parallel graph-based ANNS algorithms. Uses techniques like prefix doubling for parallel batch insertions on immutable graph snapshots, minimizing locks.	ParlayANN provides efficient and scalable graph construction and search on multi-core processors, outperforming prior parallel implementations like HNSW and DiskANN.
[6] Akhil & Sivashankar (2025)	Performance degradation of HNSW on very large (billion-scale) datasets.	Zonal HNSW (ZHNSW): Partitions the dataset into smaller zones using K-Means clustering. Each zone maintains an independent HNSW graph index.	ZHNSW improves the scalability of HNSW for extremely large datasets but introduces memory overhead due to managing multiple graph indices.
[7] Zhang et al. (2025)	I/O latency bottleneck in disk-based ANNS algorithms where the index resides on SSD.	eXtended Neighborhood Graph (XN-Graph) with wider neighbor coverage; Boundary-adaptive Balanced Partition (BBP) for construction; In-Memory First Search (IMF Search) strategy prioritizing cached nodes.	Achieves high QPS (>4000) and recall (>0.9) on billion-scale disk-based search, significantly reducing latency compared to DiskANN/SPANN/Starling due to fewer hops and optimized I/O. Extremely fast index construction.
[8] Gollapudi et al. (2023)	Poor performance of standard ANNS when queries include metadata filters (filtered ANNS), especially for filters with low specificity.	FilteredVamana, StitchedVamana: Graph construction algorithms that incorporate label/metadata information. Edges are added/pruned based on both vector proximity and shared labels between points.	Significantly outperform post-processing/inline filtering methods for filtered ANNS (often >10x speedup), achieving high throughput and recall even for low-specificity filters on SSDs.
[9] Cai et al. (2024)	Lack of versatility in current filtered ANNS	Unified Navigating Graph (UNG): Framework combining a vector proximity	UNG guarantees versatility (handles various label queries), fidelity (searches

	methods to handle diverse label constraints efficiently and guarantee results from the filtered set.	graph with a Label Navigating Graph (LNG - a DAG encoding label set containment relationships) to support versatile filtered search.	only within the filtered space), completeness (provides sufficient answers), and adaptability (integrates with most graph bases).
[10] Xu et al. (2023)	Difficulty maintaining index performance and avoiding costly rebuilds for cluster-based indices (like SPANN) under incremental vector updates.	SPFRESH: Extension of SPANN enabling efficient in-place updates. Uses LIRE (Lightweight Incremental RE-balancing) protocol to locally rebalance partitions after splits caused by insertions, avoiding global rebuilds.	Enables efficient, low-overhead in-place updates for billion-scale vector indices, maintaining search performance without the high cost and downtime of periodic full index reconstruction.
[11] Gong et al. (2025)	High latency and inefficient vector updates when using detached vector databases (e.g., Milvus) with streaming processing systems (e.g., Flink).	VStream: An integrated distributed streaming vector search engine. Uses dynamic partitioning based on LSH and hierarchical storage management integrated directly within the streaming engine.	Achieves significantly higher throughput (claimed 373x faster) and lower resource consumption (CPU, memory) compared to traditional detached architectures for vector search on streaming data.
[12] Coleman et al. (2022)	Inefficient memory access patterns in graph-based ANNS (like HNSW) lead to poor CPU cache utilization and frequent cache misses during search.	Graph Reordering: Techniques to reorganize the memory layout of graph nodes, placing frequently co-accessed nodes (based on traversal patterns) physically closer in memory.	Improves cache hit rates during graph traversal, resulting in significantly faster average query latency (up to 40% improvement observed) despite a small additional indexing overhead.
[13] Gao & Long (2023)	Dominance of Distance Comparison Operations (DCOs) in ANNS runtime; performing full O(D) computation even for candidates to be discarded.	ADSampling: Randomized DCO method using offline random orthogonal transformation and online incremental dimension sampling. Employs hypothesis testing to adaptively stop computation early for negative candidates.	Reduces expected DCO complexity for negative candidates from O(D) to O(log D), significantly speeding up AKNN algorithms with negligible accuracy loss by avoiding full computations for most candidates.
[14] Chen et al. (2023)	Many exact distance computations during graph search do not influence the final search update or path, representing wasted computation.	FINGER (Fast Inference): Approximates distance calculations during graph search. Decomposes vectors relative to the current search node and uses LSH to estimate the angle between residual vectors, approximating the full distance.	Speeds up HNSW query times by 20%-60% compared to optimized baselines by replacing many exact distance computations with faster approximations without significantly impacting accuracy.
[15] Peng et al. (2023)	Lack of practical theoretical performance guarantees in most graph-based ANNS; existing guarantees often rely on impractical assumptions (e.g., τ).	τ-Monotonic Graph (τ-MG): Guarantees finding 1-NN if $\delta(q, \bar{p}_1) < \tau$, with O(n^(1/d)(ln n)^2) complexity. Defines τ-monotonic property. Proposes efficient τ-MNG variant.	Provides a theoretical guarantee for 1-NN under a specific distance condition with better complexity than some prior guaranteed methods. τ-MNG variant shows strong empirical performance.
[16] Xie et al. (2025)	Existing graph methods lack theoretical guarantees for k-NN (k>1) search and need an impractical predefined global τ for 1-NN guarantees.	Labeled Monotonic Graph (LMG): Edges labeled with minimum required τ. Two-phase search: Adaptive Search (finds 1-NN with dynamic query-dependent $\tilde{\tau} \geqslant 0$) + Refinement Search (explores 1-NN vicinity for k>1). Proposes practical ALMG variant.	First approach with theoretical guarantee for finding exact k-NN for arbitrary queries without predefined τ. ALMG performs competitively in practice.
[17] Vecchiato et al. (2024)	Bottleneck in cluster-based ANNS (e.g., IVF) routing phase due to simple heuristics (like distance-to-centroid) for selecting clusters to search.	Reformulates cluster selection as a Learning to Rank (LTR) problem. Trains a (simple linear) function to rank clusters based on query features, replacing or augmenting the heuristic.	LTR approach consistently improves Top-1 accuracy and overall search performance of cluster-based ANNS by learning a better routing strategy.
[18] Yang et al. (2025)	Super-linear index construction time for state-of-the-art graph ANNS (NSW, HNSW) limits scalability. Acceleration often hurts search performance.	Identifies k-Candidate Neighbor Acquisition (k-CNA) as bottleneck. Proposes "Refinement-before-search": Pruning initial KNNG before k-CNA (RNG-based) & Replacing node-by-node insertion with global layering (HNSW).	Significantly accelerates index construction (up to 5.6x for NSW, 4.6x for HNSW) without degrading search performance by optimizing the expensive neighbor acquisition phase.

[19] Zhao et al. (2020)	Difficulty adapting iterative, dependency-heavy graph ANNS algorithms for efficient parallel execution on GPUs.	SONG (Search On Neighborhood Graph): Framework decoupling graph search into parallelizable stages (Seed Selection, Neighborhood Expansion, Result Merging) for GPUs. Includes GPU-specific optimizations (e.g., eliminating dynamic memory allocation).	Achieves massive speedups (50-180x) over single-thread CPU HNSW and outperforms Faiss GPU library by enabling efficient parallel graph search on GPUs.
[20] Li et al. (2023)	Lack of a comprehensive survey classifying and analyzing the growing field of deep learning (DL) applications in ANNS.	Survey categorizing DL-based ANNS into "Learning to Index" (DL hashing, partitioning, graph construction) and "Learning to Search" (DL ranking, routing, termination). Proposes "End-to-End Learning" (jointly optimizing index and search) as a future direction.	Provides a systematic overview of how DL is used in ANNS. Highlights the potential benefits of integrating DL deeper into the ANNS pipeline through end-to-end optimization.

D. *Synthesis, Gaps, Opportunities*

The reviewed literatures displayed a divergence in ANNS research. Graph-based methods like HNSW and NSG provides state of the art recalls as well as minimum latency for in-memory search yet, it still struggles with memory consumption and build times at extreme scales. Disk-based solutions like DiskANN, SPANN, and XN-Graph address the scale issue by leveraging SSDs but introduce I/O latency as the new bottleneck. In this case, the more recent Quantization methods like IVF-PQ and MRQ excel in memory efficiency but traditionally trade off accuracy, especially at high recall data memory targets.

Recent publicized hybrid approaches have attempted to combine these strengths. Zonal HNSW approaches the data where it partitions the data like IVF but uses HNSW locally, improving intra-zone search efficiency. However, keeping the local graphs entirely separate, risks missing neighbours near zone boundaries, potentially limiting the approach's maximum recall. Disk-based approach such as DiskANN as well as SPANN also represent hybrid concepts but are optimized for disk I/O rather than in-memory performance trade-offs.

While partitioning reduces memory and enable the usage of parallelism and local graph searches improve accuracy over brute-force scans, there still remain a gap in these hybrid frameworks that tightly integrates partitioning with graph structures specifically in optimizing the in-memory memory-vs-performance trade-off. Current proven works such as Zonal HNSW has demonstrated the promise of partitioning graphs, but further enhancements exist such as an enhancement for inter-zone connectivity or search strategies to mitigate the recall limitations inherent in hard partitioning, without resorting to the full memory cost of a monolithic HNSW graph.

E. *Summary of Literature Review*

The common trade-offs in ANNS algorithm design are highlighted in this review, especially those involving memory usage, construction time, search latency, and recall. While partitioning and quantisation techniques are memory-efficient but may sacrifice accuracy, graph-based techniques provide excellent performance but have scalability problems. Although existing solutions are frequently designed for disc I/O or maintain partitions' relative independence, hybrid approaches show promise. In order to overcome the drawbacks of both pure graph and pure partitioning approaches, this review presented us with a new research opportunity to create a novel hybrid in-memory framework that cleverly combines partitioning with localised graph search to achieve a superior balance between memory efficiency and high recall. Zonal Graph Quantisation (ZGQ), our suggested work, attempts to close this gap. Using the Template

III. Research Methodology

A. *Introduction to the Methodology*

In order to solve the memory-performance trade-off problem in billion-scale Approximate Nearest Neighbor Search (ANNS) systems, the Zonal Graph Quantization (ZGQ) solution was designed, and evaluated using the methodological framework described in this section. This methodology's main objective is to compare ZGQ to its predecessors which are well-known algorithms in order to quantitatively show its benefits in balancing search performance, construction cost, and memory efficiency.

B. *Research Design*

A quantitative experimental study design is used in the research. It is specifically a comparative study with the goal of comparing the suggested ZGQ framework's performance metrics to two main baseline indexing models:

1. *Hierarchical Navigable Small World* (HNSW) is a pure graph-based index.
2. *Pure Partitioning-Based Index*: Inverted File (IVF), usually enhanced for efficiency with Product Quantization (PQ) (IVF-PQ).

C. *Steps Involved*

1. Finding Research Objectives
   The main objective of our applied methodology is to discover and observe the actual performance of our solution when applied to high-dimensional vector datasets, where the ZGQ hybrid indexing approach should offer a better balance between memory efficiency and search performance (Recall@k and Latency) than its monolithic predecessor as such; HNSW and conventional IVF-PQ.

2. Literature Review and Analysis:
   The current limitations and research gap were determined by a thorough examination of the existing ANNS algorithms (graph-based, quantization-based, and hybrid approaches) obtained through a structured Literature Review (Section II). The architectural design of ZGQ was directly influenced by this analysis, which inspired our combination of localized HNSW graphs and IVF-style partitioning to lessen the construction cost and monolithic memory overhead associated with full HNSW indices.

3. Validation
   Our proposed solution, so-called Zonal Graph Quantization (ZGQ) is a hybrid in-memory indexing solution which differs from methods such as Zonal HNSW by combining a quantization step and tight optimization design for memory-performance trade-offs. Which uses the following:

   - Partitioning: Which uses K-means clustering to divide vector space into Z non-overlapping zones (clusters) which mimics the IVF approach
   - Local Indexing: Which involves in constructing independent and smaller HNSW graphs between each of the Z zone.
   - Quantization: Which involves applying an optimal vector quantization (which may include product quantization or residual quantization) to further compress the index size.
   - Search: Which means querying the nearest n_{probe} Zone centroids whilst also running a parallel lightweight HNSW searches within the identified zones

D. Evaluation Metrics

The evaluation will compare our proposed solution called Zonal Graph Quantization (ZGQ) against established algorithms such as HNSW (Graph Baseline) and IVF-PQ/MRQ (Partitioning Baseline) based on the following quantitative metrics:

TABLE II. EVALUATION METRICS

Metric	Units	Goal	Relation to Problem
Recall@k	Accuracy / Hit Rate (%)	Max.	Measures search accuracy and quality.
Query Latency	Speed (Queries Per Second /milliseconds)	Min.	Measures query performance and efficiency.
Index Size	Memory Footprint (GB)	Min.	Directly addresses the HNSW memory overhead problem.
Index Build Time	Time (Seconds / Hours)	Min.	Directly addresses the HNSW construction cost problem.

E. Data Collection and Experimental Setup

The evaluation of our proposed solution namely the Zonal Graph Quantization (ZGQ) will utilize synthetic datasets with real world benchmarks to ensure that the validation done are comprehensive and controlled with realistic conditions.

1. Synthetic Data Generation: Controlled and artifical datasets are going to be created using a custom Python script (generate_test_data.py). Where this script allows the systematic variation of key parameters revolving the datasets/
   a. Number of Vectors (n_vectors): experiments are conducted at different scales of datasets such as from 10,000 vectors with an upwards trajectory of 100,000 vectors or even more which can be used to analyze the scalability of the solution
   b. Vector dimensions (dim): Where a standard vector dimension will be used and in this case we have decided to use 128 as the size of the dimension to simulate common embedding size.
   c. Number of Queries (n_queries): Where a fixed set of query vectors be it 100 or 1,000 will be generated for consistent evaluation across different algorithms.
   d. The generation process for all of the data's follows a standard normal distribution (numpy.random.randn) which is then followed by an L2 normalization which ensures that all of the vectors are lying on a hypersphere which is a common ANNS benchmark practice.
2. Experimental Environment: The device which was used to demonstrate and run the experiments to get our current figures follows the specification of:
   a. **Hardware**: A Workstation with an CPU namely the Intel core I5-12500H complimented with an NVIDIA graphics card namely the RTX3050 complimented with 16 Gigabytes of Ram and 512 Gigabytes of SSD storage
   b. **OS**: Using WSL2 Ubuntu 24, under Windows 11
3. Execution Configuration: Where the program and simulation will be executed under the condition where:
   - Index constuction: Where the simulation will utilize multithreading capabilities to use the available CPU cores to measure the practical build times
   - All experiments are run with consistent system settings to ensure that the scenario's are reproduceable
   - Background processes are minimized to reduce noise during the program running

F. Baseline Algorithm Implementation Details

To ensure that the comparisons being made are fair and accurate across different settings and environments, we have used well-established and optimized libraries to ensure that the baseline algorithms are well implemented. Where we have the following baseline implementation details as outlined:

1. **HNSW Baseline**:
   a. **Library**: `hnswlib` (version 0.8.0)
   b. **Justification**: hnswlib is the base implementation of the HNSW that we have used in our research due to its speed and optimization being in C++ wrapped around Python keywords

c. **Configs**:
   i. M = 16 (Number of Partitions)
   ii. Ef_construction = 200 (Controls the index construction quality)
   iii. Ef_search = 50 (Controls the search quality which then varied in the experiments itself)

2. **IVF and IVF-PQ Baseline:**
   a. **Library:** Custom implementation using `Scikit-Learn` for K-means clustering.
   b. **Components:**
      i. K-Means clustering = which uses `cluster.kmeans` (version 1.3.0+)
      ii. P-Quantization = Custom implementation following standard PQ algorithms.
   c. **Configs:**
      i. $\boldsymbol{n_{list}}$= 100 **(**No. of Clusters**)**
      ii. $\boldsymbol{n_{probe}}$ = 10 (No. of clusters to search which are varied throughout the experiments)
      iii. **For IVF-PQ =** 16 Subspaces with 8 bits per subspace

G. Proposed Algorithm Implementation Details

Our proposed solution, which is the Zonal Graph Quantization (ZGQ) are implemented by combining the following self-implemented components:

1. **Partitioning Tables**:
   a. Library: `sklearn.cluster.kmeans` with the version 1.3.0+ as base
   b. Purpose: Divides vector space into Z non-overlapping zones
   c. Configuration: Z = 4 Zones (the default value which then can be tuned to different simulation scenarios)
2. **Local Graph Quantization**:
   a. Library: `hnswlib` version 0.8.0+
   b. Purpose: To construct independent HNSW graphs within each different zone.
   c. Configuration: Using reduced M = 8 to compare to M = 16 for full HNSW which has the goal to decrease memory usage
3. **Quantization Module**:
   a. Implementation: Custom implementation from the repository, product_quantizer.py
   b. Purpose: To compress the vectors representations within each of the dedicated zones
   c. Configuration: Using M = 16 Sub-spaces as well as 8 bits per sequence
4. **Search Orchestration:**
   a. Implementation: Custom implementation from the repository, search.py
   b. Purpose: Coordinate zone selection with parallel graph search
   c. Process:
      i. Compute Query distances
      ii. Select top n_{probe}zones
      iii. Perform HNSW search in the selected zone
      iv. Merge and then rank the result

H. Tools and Technologies used

The implementation of our algorithm uses the following libraries to ensure that all the intended functions work as is intended without sacrificing any implementation

1. Core Libraries:
   - NumPy = 1.24.3
   - Scikit-learn = 1.3.0
   - Hnswlib = 0.8.0
   - Numba = 0.58.0
2. Visualization:
   - Matplotlib = 3.7.2
   - Seaborn = 0.12.2
   - Pandas = 2.0.3
3. Dev Tools:
   - Pytest = 7.4.0
   - Jupyterlab = 4.0.5

With Git and GitHub as version control and Repository

I. Experimental Procedures:

The evaluation of our system follows an organized and systematic approach to ensure that our experiment follows fulfills research standards,

1. Index Construction Phase
   - Where for each algorithm that we consider as baseline (HNSW, IVF, IVF-PQ, ZGQ) are going to have the indexes built from the training vectors.
   - Measurement:
     - Build Time (Seconds)
     - Index Size (MB)
     - Peak Memory during Construction
2. Query Execution Phase
   - Where for each of the constrained index ,we are going to fetch a fixed set of query vectors ranging between 100 – 1000 queries itself.
   - Upon fetching, we are going to execute 10 warm-up queries to initialize and re-fetch the cached data.
   - Measurement:
     - Execute all queries sequentially
     - Record latency for each query (ms)
     - Compute the following:
       - Average Query Latency
       - Throughput (QPS)
       - 95th Percentile latency
3. Parameter Sweeping
   - To generate performance curves, we have randomized and varied the Key Search params. Where we have decided on the following parameters for each baseline algorithm:
     - HNSW:
       $Ef_{Search} \in \{10, 20, 50, 100, 200, 400\}$
     - IVF/ZGQ:
       $N_{probe} \in \{1, 2, 5, 10, 20, 50\}$
   - Where for each of the parameters:
     - Execute the Query
     - Record all the identified metrics
     - Plot the Recall@10 vs Query Latencies
4. Recall Computation
   - For each of the query result set:

- Compare the predicted neighbors against Ground Truth using Brute-force
- Compute the Recall@k using: Numbers of Correct neighbors / K
- Average the result across all queries to get the overall Recall@k

5. Comparative Analysis
   - Performance is then compared across different baseline algorithms with:
     - Recall levels: 50, 70, 90, 95}%
     - Comparison Vars:
       - Recall query latency
       - Build Time comparison
       - Memory-recall trade-offs
       - Index Size at optimum params.

J. *Visual Flowchart:*

The diagram below displays the experimental flow of the methodology as stated:

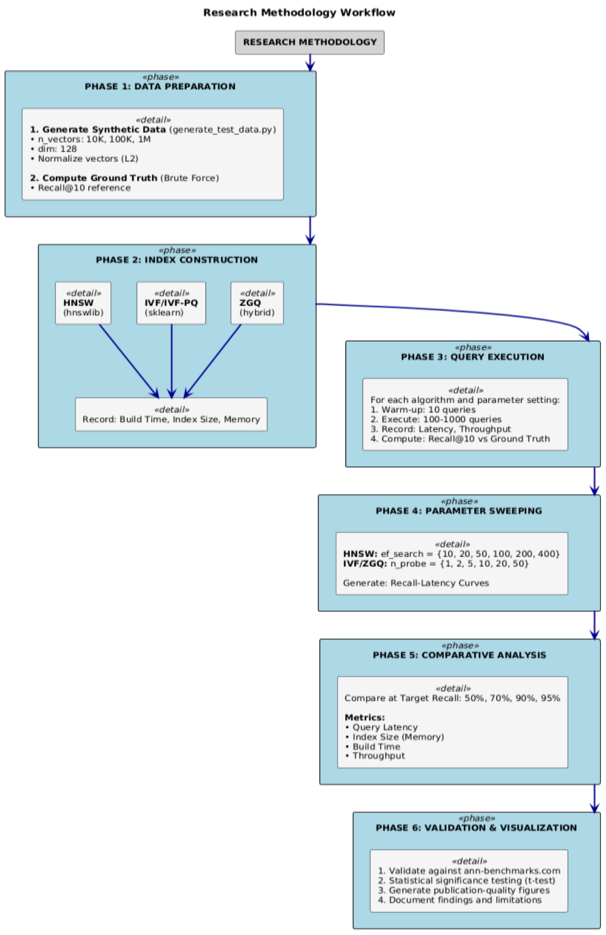
Research Methodology Workflow
RESEARCH METHODOLOGY
«phase»
PHASE 1: DATA PREPARATION
«detail»
1. Generate Synthetic Data (generate_test_data.py)
• n_vectors: 10K, 100K, 1M
• dim: 128
• Normalize vectors (L2)
2. Compute Ground Truth (Brute Force)
• Recall@10 reference
«phase»
PHASE 2: INDEX CONSTRUCTION
«detail»
HNSW
(hnswlib)
«detail»
IVF/IVF-PQ
(sklearn)
«detail»
ZGQ
(hybrid)
«detail»
Record: Build Time, Index Size, Memory
«phase»
PHASE 3: QUERY EXECUTION
«detail»
For each algorithm and parameter setting:
1. Warm-up: 10 queries
2. Execute: 100-1000 queries
3. Record: Latency, Throughput
4. Compute: Recall@10 vs Ground Truth
«phase»
PHASE 4: PARAMETER SWEEPING
«detail»
HNSW: ef_search = {10, 20, 50, 100, 200, 400}
IVF/ZGQ: n_probe = {1, 2, 5, 10, 20, 50}
Generate: Recall-Latency Curves
«phase»
PHASE 5: COMPARATIVE ANALYSIS
«detail»
Compare at Target Recall: 50%, 70%, 90%, 95%
Metrics:
• Query Latency
• Index Size (Memory)
• Build Time
• Throughput
«phase»
PHASE 6: VALIDATION & VISUALIZATION
«detail»
1. Validate against ann-benchmarks.com
2. Statistical significance testing (t-test)
3. Generate publication-quality figures
4. Document findings and limitations

Fig. 1. Methodology Flowchart

K. *Challenges and Limirations:*

When conducting the experiment, we stumbled upon several challenges that we would need to address/mitigate accordingly, the challenges and mitigation strategies are outlined as below:

1. Synthetic Data Limitation
   - Challenge: Synthetic Data may not capture actual real world distribution characteristics
   - Mitigation: We have normalized our vectors in a hypersphere which mimic common embedding properties.
2. Hardware Constraints:
   - Challenge: Due to hardware constraints where in this case we only have 16 GB Ram, it prevented us from running at a truly large scale
   - Mitigation: We ran the test at 10,000 as well as 100,000 scales complimented with a linear search to simulate large-scale data performance
3. Parameter Tuning:
   - Challenge: Different algorithms have different optimal parameters, thus finding these optimal params is computationally expensive.
   - Mitigation: We can use standard parameter ranges from our literature review.

L. *Summary:*

Our proposed methodology aims to provide a comprehensive checklist to evaluate our proposed Zonal Graph Quantization (ZGQ) algorithm against its established predecessors such as HNSW as well as IVF-PQ. Where our methodology includes:

1. An established library implementation for baseline algorithms to ensure fairness
2. A controlled and systematic data generation
3. A statistical validation done to ensure valid result

On top of that, our evaluation process follows the steps which are as outlined below:

1. Generate datasets across various data scales
2. Create indices for all the algorithms to measure its build metrics
3. To run the fixed queries to record latency and the QPS
4. To compute the recall latency against ground truth
5. To generate recall latency trade-offs with other metrics
6. To perform a comparative analysis at the target recall

On top of doing a systematic approach and analysis we would also acknowledge our known limitations which include synthetic data characteristics, our hardware constraints as well as our different optimization implementation. Despite the limitations we still carried out our experiment mitigating through normalization techniques, as well as linear scaling complimented with visualization of the results.

Our methodology is ensured to enable a fair and comprehensive result as well as a reproducible evaluation of our proposed algorithms which address our algorithm's effectiveness in addressing the memory with performance trade-offs in large scale Vector databases with ANNS systems. Our results will provide quantitative evidence to support or refute the hypothesis that hybrid approaches can maintain competitive recall while significantly reducing memory footprint compared to monolithic HNSW indices

ACKNOWLEDGMENT

We acknowledge IEEE Explore, ACM Digital Library, NeurIPS as well as ArXiv for the resources that we have gathered be it from research papers or as well as code implementation repository, we also acknowledge the writer's that we have cited in this paper for their contribution towards the research in the vector database field since this research might not have happened without them and their contribution towards the AI/ML/RAG/VDBMS community.

We acknowledge our usage of the Python 3 programming language that is maintained by the Python Software Foundation (PSF) as well as the teams who are behind the libraries that we have used. On top of that we acknowledge our usage of GitHub as well as Git for the research as without these tools the collaboration within may not have happened.

REFERENCES

[1] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, Apr. 2020. doi: 10.1109/TPAMI.2018.2889473.

[2] M. Wang, X. Xu, Q. Yue, and Y. Wang, "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 1964–1978, Jul. 2021. doi: 10.14778/3476249.3476255.

[3] L. Prokhorenkova and A. Shekhovtsov, "Graph-based nearest neighbor search: From practice to theory," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, Online, 2020, pp. 7803–7813.

[4] Q. Chen *et al.*, "SPANN: Highly-efficient billion-scale approximate nearest neighbor search," in *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021, pp. 5199–5212.

[5] M. D. Manohar *et al.*, "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 (PPoPP)*, Edinburgh, UK, Mar. 2024, pp. 272–285. doi: 10.1145/3627535.3638481

[6] A. Akhil and G. Sivashankar, "Zonal HNSW: Scalable approximate nearest neighbor search for billion-scale datasets," in *2025 3rd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, 2025. doi: 10.1109/ICSSAS66150.2025.11081070.

[7] C. Zhang, J. Wang, W.-L. Zhao, and S. Xiao, "Highly efficient disk-based nearest neighbor search on extended neighborhood graph," in *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Padua, Italy, Jul. 2025, pp. 2513–2523. doi: 10.1145/3726302.3729996.

[8] S. Gollapudi *et al.*, "Filtered - DiskANN: Graph algorithms for approximate nearest neighbor search with filters," in *Proceedings of the ACM Web Conference 2023 (WWW '23)*, Austin, TX, USA, Apr./May 2023, pp. 3406–3416. doi: 10.1145/3543507.3583552

[9] Y. Cai, J. Shi, Y. Chen, and W. Zheng, "Navigating labels and vectors: A unified approach to filtered approximate nearest neighbor search," *Proceedings of the ACM on Management of Data*, vol. 2, no. 6, pp. 246:1–246:27, 2024. doi: 10.1145/3736716

[10] Y. Xu *et al.*, "SPFRESH: Incremental in-place update for billion-scale vector search," in *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, Koblenz, Germany, Oct. 2023, pp. 545–561. doi: 10.1145/3600006.3613155

[11] S. Gong, H. Sun, L. Liu, and L. Chen, "VStream: A distributed streaming vector search system," *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 31:1–31:26, Mar. 2025. doi: 10.1145/3746654

[12] B. Coleman, S. Segarra, A. Smola, and A. Shrivastava, "Graph reordering for cache-efficient near neighbor search," in *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022, pp. 35831–35843.

[13] J. Gao and C. Long, "High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 137:1–137:27, Jun. 2023. doi: 10.1145/3589282

[14] . H. Chen, W.-C. Chang, J.-Y. Jiang, H.-F. Yu, I. S. Dhillon, and C.-J. Hsieh, "FINGER: Fast inference for graph-based approximate nearest neighbor search," in *Proceedings of the ACM Web Conference 2023 (WWW '23)*, Austin, TX, USA, Apr./May 2023, pp. 3225–3235. doi: 10.1145/3543507.3583318

[15] Y. Peng, B. Choi, T. N. Chan, J. Yang, and J. Xu, "Efficient approximate nearest neighbor search in multi-dimensional databases," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 54:1–54:27, May 2023. doi: 10.1145/3588908

[16] J. Xie, J. X. Yu, and Y. Liu, "Graph based K-nearest neighbor search revisited," *ACM Transactions on Database Systems*, vol. 50, no. 4, pp. 14:1–14:30, Jun. 2025 (Expected). doi: 10.1145/3736716.

[17] Vecchiato, F. M. Nardini, N. Tonellotto, and R. Perego, "A learning-to-rank formulation of clustering-based approximate nearest neighbor search," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Washington DC, USA, Jul. 2024, pp. 43–52. doi: 10.1145/3626772.3657731.

[18] S. Yang, J. Xie, Y. Liu, J. X. Yu, X. Gao, Q. Wang, Y. Peng, and J. Cui, "Revisiting the index construction of proximity graph-based approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 18, no. 6, pp. 1825–1838, 2025 (Expected). doi: 10.14778/3714571.3714643

[19] W. Zhao, S. Tan, and P. Li, "SONG: Approximate nearest neighbor search on GPU," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, Dallas, TX, USA, Apr. 2020, pp. 1033–1044. doi: 10.1109/ICDE48307.2020.00096

[20] M. Li *et al.*, "Deep learning for approximate nearest neighbour search: A survey and future directions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 9, pp. 8997–9021, Sep. 2023. doi: 10.1109/TKDE.2022.3220683.