Proceedings of the 3rd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS-2025)
IEEE Xplore Part Number: CFP25DN7-ART; ISBN: 979-8-3315-3884-2

2025 3rd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS) | 979-8-3315-3884-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICSSAS66150.2025.11081070

Zonal HNSW: Scalable Approximate Nearest Neighbor Search for Billion-Scale Datasets

1st A Akhil
Department of Computational Intelligence
SRM Institute of Science and Technology
Chennai, Tamil Nadu, India
az3805@srmist.edu.in

2nd Sivashankar G
Department of Computational Intelligence
SRM Institute of Science and Technology
Chennai, Tamil Nadu, India
sivashag@srmist.edu.in

***Abstract*—Effectively navigating billion-scale datasets for Approximate Nearest Neighbor (ANN) search remains a critical challenge in modern data science. We introduce *Zonal Hierarchical Navigable Small World* (ZHNSW), a novel framework designed to provide superior scalability and performance. While leveraging insights from established algorithms like HNSW, FAISS, and Annoy, ZHNSW introduces a key innovation: zonal partitioning. This technique decomposes the high-dimensional search space into multiple, smaller, localized HNSW graphs, thereby reducing complexity and enabling significant parallelization of search operations. Complementing this, ZHNSW employs adaptive zone selection strategies—encompassing fixed fraction, distance thresholding, and heuristic-based methods—which dynamically optimize the search process by focusing on the most relevant data regions. Extensive experiments on challenging benchmarks such as SIFT1B, DEEP1B, GLOVE-1.2M, and MUSIC100M, utilizing the NVIDIA DGX A100 platform, confirm ZHNSW's advantages. It demonstrates up to 3× improvement in search speed, a high recall@10 of 98.7%, and a 29% decrease in index size when compared against state-of-the-art alternatives, solidifying ZHNSW as a robust and highly efficient solution for contemporary data-intensive similarity search applications.**

***Index Terms*—Approximate Nearest Neighbor (ANN) Search, Zonal Hierarchical Navigable Small World (ZHNSW), High-Dimensional Indexing, HNSW (Hierarchical Navigable Small World), FAISS (Facebook AI Similarity Search), Annoy (Approximate Nearest Neighbors Oh Yeah)**

I. Introduction

The proliferation of large-scale data in computer vision, NLP, and recommendation systems necessitates efficient **Approximate Nearest Neighbor (ANN)** search algorithms [3], [4]. Applications like image retrieval and semantic search demand fast, accurate high-dimensional queries. As datasets reach billions of vectors, traditional exact k-NN searches ($O(n \cdot d)$ complexity [15]) become impractical, driving research into ANN methods balancing speed, accuracy, and memory [8].

Hierarchical Navigable Small World (HNSW) graphs are prominent for their query efficiency and accuracy [1], [16]. HNSW extends Navigable Small World (NSW) graphs with a multi-layered structure inspired by skip lists [5]. This hierarchy enables searches to start in sparse upper layers and navigate to denser lower layers for fine-grained proximity search, achieving sub-linear complexity and high recall suitable for large, high-dimensional data [2], [14].

However, HNSW faces challenges at billion-scale:

1) **Memory Overhead**: Full HNSW graphs are memory-intensive due to numerous multi-layer edges, especially for high recall [7].
2) **Graph Construction Time**: Building HNSW indexes is computationally expensive, growing inefficiently with dataset size [2].
3) **Search Latency**: Sub-linear search complexity still leads to higher latency in larger graphs [1].
4) **Inefficiency in Sparse Regions**: Greedy traversal can perform poorly in low-density areas with fewer connections [16].

Motivated by HNSW's limitations at billion-scale, this work proposes **Zonal HNSW (ZHNSW)**. ZHNSW achieves this via **zonal partitioning** into smaller zones, each with an independent, localized HNSW graph. This reduces graph complexity, enables **parallel searches**, and uses **adaptive zone selection** to optimize query performance.

II. Related Work

ANN search is essential for managing growing high-dimensional data. Early tree-based methods (KD-trees, Ball trees) offer logarithmic complexity but struggle with the curse of dimensionality in modern datasets [15], prompting exploration of alternatives. Locality Sensitive Hashing (LSH) provides probabilistic sub-linear similarity search but high recall often demands large memory (long codes/multiple tables), presenting a memory-recall trade-off [11].

Quantization techniques like Product Quantization (PQ) reduce storage and accelerate search by decomposing and quantizing sub-vectors for compact representations and efficient distance computations [3], [4]. Combined with inverted file indexes (e.g., in FAISS [4], [10]), they boost efficiency. However, quantization errors impact accuracy, requiring careful cluster optimization. Ongoing work focuses on optimizing cluster numbers and minimizing quantization error.

Graph-based methods excel in high-dimensional spaces. Navigable Small World (NSW) graphs introduced greedy routing on proximity graphs [3]. Hierarchical NSW (HNSW) extends this with a skip-list-inspired hierarchy for sub-linear search and high recall [1], [16]. Yet, HNSW faces memory and construction time challenges at billion-scale, motivating

research into optimized construction and parallelism [2], [14]. Dynamic HNSW variants address updates but add complexity in maintaining graph quality and balancing insertion/search speed.

Clustering (e.g., k-means) partitions data to limit search scope, with separate indexes per cluster improving efficiency [12]. Combining clustering with HNSW enhances performance, but challenges include optimal cluster number determination and handling queries near boundaries. Adaptive cluster selection and inter-cluster search are crucial.

Hybrid approaches offer further improvements. DiskANN combines quantization and graph methods for disk-based storage of massive datasets [9]. GPU acceleration significantly speeds up distance computations and search operations [4], [13]. Learned index structures adapt to data distributions for optimized performance. Purpose-built systems like Milvus offer comprehensive vector data management [6]. Research continues on new hybrids for optimal speed, accuracy, memory, and scalability, with dynamic updates, hardware acceleration, and data structure exploitation as key future directions.

III. Algorithm

The Zonal Hierarchical Navigable Small World (ZHNSW) algorithm, whose overall architecture is depicted in Fig. 1, performs efficient large-scale ANN search by combining zonal partitioning, localized HNSW graph construction, and a multi-zone search strategy [1]. This hybrid approach reduces computational overhead by dividing the dataset into smaller, manageable zones, each with its own HNSW graph [2]. Search is accelerated by restricting exploration to relevant zones, with final results merged [12]. ZHNSW operates in three core phases: Zonal Partitioning, HNSW Graph Construction, and Search Procedure, all optimized for speed and accuracy.

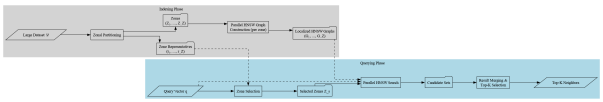

Fig. 1. ZHNSW indexing and querying architecture.

A. Zonal Partitioning

Zonal partitioning, the foundational step, divides the dataset $D = \{x_1, \ldots, x_N\}$ into **Z distinct, non-overlapping zones** (clusters): $D = \bigcup_{i=1}^{Z} Z_i, Z_i \cap Z_j = \emptyset$ for $i \neq j$. This reduces the search space, improving efficiency and latency by limiting queries to a subset of data. This leverages the locality in high-dimensional spaces, where similar vectors cluster [12], confining search and reducing candidate examination.

The choice of Z is critical, balancing search speed and recall:

- **Small Z (Fewer Zones):** Higher recall, lower search efficiency (more points per zone).
- **Large Z (More Zones):** Faster search, potentially lower recall (fragmentation, missed cross-zone neighbors).

a) Clustering Algorithms for Zone Formation: Standard clustering techniques partition the dataset. **K-Means clustering**, which iteratively assigns points to the nearest centroid minimizing intra-cluster variance $\sum_{i=1}^{Z} \sum_{x \in Z_i} ||x - c_i||^2$ (where c_i is centroid of Z_i), is often preferred for its large-scale efficiency and scalability.

1) Zone Representatives Calculation: Once the dataset is divided into zones, each zone is extbfsummarized using a extbfrepresentative vector. This representative acts as a reference point for **zone selection** during search.

The most common choice for a zone representative is the **centroid**, defined as the mean of all vectors within a zone:

$$r_i = \frac{1}{|Z_i|} \sum_{x \in Z_i} x$$

Where:

- r_i = Representative vector for zone Z_i.
- $|Z_i|$ = Number of points in zone Z_i.
- $x \in Z_i$ = Data points within the zone.

Alternative zone representatives include:

- **Medoid**: The actual data point closest to the centroid.
- **Boundary Points**: Points near the zone's edge to capture outliers.

The set of zone representatives $R = \{r_1, r_2, \ldots, r_Z\}$ is stored in a separate index. During query time, this index is used to identify the closest zones quickly.

2) Advanced Zone Boundaries (Optional): To improve the accuracy of zone assignment, extbfadvanced boundary methods can be used to refine the separation between zones.

a) Voronoi Diagram Approach: One advanced technique is to use extbfVoronoi diagrams, which partition space into regions based on the nearest centroid. Given a set of centroids $\{r_1, \ldots, r_Z\}$, the Voronoi cell V_i corresponding to r_i is defined as:

$$V_i = \{x : ||x - r_i|| \leq ||x - r_j||, \forall j \neq i\}$$

b) Geometric Boundary Adjustment: For datasets with complex geometries, **boundary points** can be identified using convex hulls or decision boundaries, which provides a more **flexible partitioning** approach.

3) Algorithm: Zonal Partitioning Process: **Input:**

- D : High-dimensional dataset
- Z : Number of zones
- Cluster_Algorithm : Clustering method

Output:

- $\{Z_1, Z_2, \ldots, Z_Z\}$: Partitioned zones
- $R = \{r_1, r_2, \ldots, r_Z\}$: Zone representatives

1) **Dataset Partitioning**: Apply the chosen clustering algorithm:

$$\{Z_1, Z_2, \ldots, Z_Z\} = \text{Cluster}(D, Z)$$

2) **Compute Zone Representatives**: For each zone Z_i, calculate the centroid:

$$r_i = \frac{1}{|Z_i|} \sum_{x \in Z_i} x$$

3) **(Optional) Define Zone Boundaries**: Construct Voronoi regions or geometric boundaries.
4) **Output**: Return the zones and the corresponding representatives.

B. HNSW Graph Construction

Following zonal partitioning, **Hierarchical Navigable Small World (HNSW) graphs** are constructed independently for each zone. This pivotal step enables efficient ANN search within zones. Building smaller, independent HNSW graphs, instead of a single large one, improves computational efficiency (faster construction/search, less memory) and allows parallel graph construction.

1) HNSW Graph Structure and Construction Process: Standard HNSW graphs, multi-layered proximity structures combining small-world networks and skip-list concepts for fast high-dimensional search [16], are built for each zone.

For a partitioned dataset $D = \{Z_1, \ldots, Z_Z\}$, an HNSW graph G_i is built for each zone Z_i:

1) **Graph Initialization**: Initialize an empty HNSW graph for each zone.
2) **Greedy Insertion**: Add each point via greedy nearest-neighbor search to find its position.
3) **Optional Cross-Zone Linking**: Optionally, establish inter-zone links between spatially-close points across different zones to enhance recall.

2) Core Parameters: HNSW graph construction is influenced by two main hyperparameters:

- **M (Max Degree)**: Max neighbors per node (typically 16-64). Higher M improves recall but increases indexing time/memory and slows search; lower M is faster with lower recall.
- **efConstruction (Exploration Factor)**: Candidate neighbors explored during insertion (typically 100-400). Higher efConstruction yields a more accurate graph but slower insertion; lower efConstruction is faster but less accurate.

a) Mathematical Formulation of HNSW Construction: Given a partitioned dataset $D = \{Z_1, Z_2, \ldots, Z_Z\}$, with M (maximum degree) and efConstruction (exploration factor), the construction follows:

b) Step 1: Graph Initialization: Each zone Z_i starts with an empty graph:

$$G_i = \emptyset, \quad \forall i \in [1, Z]$$

c) Step 2: Greedy Insertion Process: For each point $x \in Z_i$:

1) **Find Entry Point**: Locate the nearest existing point via greedy search:

$$p = \arg\min_{y \in G_i} ||x - y||_2$$

2) **Navigate Layers**: Traverse from the top layer L to Layer 0, refining neighbors.
3) **Insert Point**: At Layer 0, insert x and update its M-nearest neighbors:

$$N(x) = \{y_1, \ldots, y_M : ||x - y_j|| \text{ is minimal}\}$$

4) **Update Connections**: For each neighbor $y \in N(x)$:

$$N(y) = N(y) \cup \{x\}, \quad \text{if } |N(y)| \leq M$$

3) Complexity Analysis:

- **Graph Construction:** For each zone Z_i, complexity is:

$$O(N_i \times \log N_i \times \text{efConstruction})$$

where N_i is the number of points in the zone and M is the maximum degree of nodes.

- **Memory Usage:** Grows linearly with nodes and neighbors:

$$O(N \times M)$$

C. Search Algorithm

The ZHNSW search algorithm locates top-k nearest neighbors for a query vector q by leveraging zonal partitioning and parallel HNSW searches. This combines ANN speed with distributed zone-based searching, reducing query time while maintaining high recall. The process has three main stages: Zone Selection, Parallel Search, and Result Merging, as illustrated in the detailed search pipeline in Fig. 2.

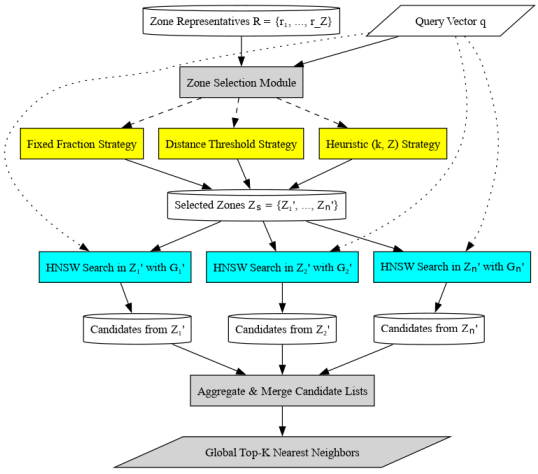

Fig. 2. ZHNSW search pipeline: zone selection, parallel HNSW search, and result merging.

1) Zone Selection: This first step identifies the n closest zones to q, crucial for search speed and accuracy.

a) Zone Identification Process: Given query q and zone representatives $R = \{r_1, \ldots, r_Z\}$ (for zones Z_i):

1) **Distance Calculation**: Compute $d(q, r_i) = \|q - r_i\|_2$ for each representative.
2) **ANN Search for Zones**: Use an ANN method (e.g., FAISS, HNSW) on R to find the n nearest zones.
3) **Select Zones**: Identify the n zones $\{Z_1, \ldots, Z_n\}$ with the smallest $d(q, r_i)$.

Complexity: Distance Calculation $O(Z)$, *ANN Search* $O(\log Z)$.

2) Parallel Search within Zones: After selecting n zones, parallel HNSW searches are conducted within their respective graphs.

a) Search Process in Each Zone (Z_i):

1) **Initialization**: Start at the HNSW graph's entry point.
2) **Greedy Search**: Traverse using best-first search, moving to the nearest unvisited neighbor until convergence.
3) **Candidate Pool**: Maintain a pool of top-k candidates.

For HNSW graph G_i of zone Z_i, the entry point is $p = \arg\min_{y \in G_i} \|q - y\|_2$. Neighbors $N(p)$ are explored, adding closest points to query q to candidate pool S. Stops when no closer neighbors are found or pool size k is reached.

3) Merging and Top-k Selection: Results from the n searched zones are merged to find the overall top-k nearest neighbors.

a) Merging and Ranking Process:

1) **Aggregate Candidates**: Collect all candidates $\bigcup_{i=1}^{n} S_i$.
2) **Final Distance Calculation**: Compute $d(q, x) = \|q - x\|_2$ for each $x \in \bigcup S_i$.
3) **Top-k Selection**: Identify the k candidates with minimal $d(q, x)$.

Algorithm 1 ZHNSW Search Algorithm

Require: Query q, zone reps R, HNSW graphs G, neighbors k, zones to search n
Ensure: Top-k nearest neighbors
0: Compute distances: q to each $r \in R$.
0: Identify n closest zones via ANN search on R.
0: Perform parallel HNSW searches in selected zones.
0: Merge candidate pools from all searched zones.
0: **return** Top-k nearest neighbors. =0

4) ZHNSW Search Algorithm:

5) Overall Complexity Analysis: For a dataset of size N, Z zones, n searched zones, and k neighbors:

- **Zone Selection:** $O(\log Z)$ (dominated by ANN search on representatives)
- **Parallel Search:** $O(n \cdot \log(N/Z))$ (assuming balanced zones)
- **Merge and Selection:** $O(n \cdot k)$

Total complexity: $O(\log Z + n \cdot (\log(N/Z) + k))$.

IV. Zone Selection Strategies

Zone selection is a critical step in the ZHNSW framework as it directly impacts both the efficiency and accuracy of the nearest neighbor search. Since the data is divided into multiple zones, the algorithm must decide which zones to search to find the top-k nearest neighbors. An effective zone selection strategy ensures that the search is both efficient (by minimizing unnecessary zones) and accurate (by covering all likely candidates).

A. Fixed Fraction Strategy

The **Fixed Fraction Strategy** is a straightforward and efficient method for selecting zones during the nearest neighbor search in the **ZHNSW** framework. It works by choosing a **fixed proportion** of the total zones, regardless of the query's location or data distribution. This strategy simplifies the search process by maintaining a constant computational overhead, making it suitable for systems where **speed** and **simplicity** are prioritized.

1) Mathematical Formulation: Given:

- Z = Total number of zones (from the zonal partitioning step).
- f = Fixed fraction (a constant between 0 and 1, typically defined by the user).
- n = Number of zones to search.

The number of zones selected for the search is computed using the formula:

$$n = \text{round}(Z \times f) \tag{1}$$

where:

- n is the **number of zones** to search.
- Z is the **total zones** created during zonal partitioning.
- f is the **fraction** of zones to search. For example, if $f = 0.1$, the algorithm searches 10% of the total zones.

2) Algorithmic Process: The Fixed Fraction Strategy follows a deterministic procedure:

1) **Compute Total Zones**: Identify the total number of zones Z formed during the **zonal partitioning** stage. Each zone represents a subset of the dataset and is associated with a representative point.
2) **Calculate Zones to Search**: Multiply the total number of zones Z by the fixed fraction f and round the result to the nearest integer:

$$n = \text{round}(Z \times f)$$

3) **Identify Nearest Zones**: Perform an **approximate nearest neighbor (ANN)** search to find the **n closest zones** to the query vector q using the precomputed **zone representatives**.
4) **Execute Parallel Search**: Within the selected zones, perform **parallel HNSW searches** to retrieve candidate points.

3) Example Calculation: Consider a dataset that is divided into **100 zones**. If the **fixed fraction** is set to **0.1** (i.e., 10%), the algorithm will:

1) **Compute the Number of Zones**:

$$n = \text{round}(100 \times 0.1) = 10$$

2) **Select the Nearest 10 Zones**: Identify the **10 zones** that are closest to the query vector q using an **ANN** search on the zone representatives.
3) **Conduct Parallel Searches**: Perform a **parallel** nearest neighbor search within these **10 zones** to find the **top-k** nearest neighbors.

Algorithm 2 Fixed Fraction Strategy

Require: q : Query vector, Z : Total zones, f : Fixed fraction, R : Zone representatives, G : HNSW graphs
Ensure: Top-k Nearest Neighbors
0: $n = \text{round}(Z \times f)$ {Number of zones to search}
0: $closest\_zones = \text{ANN\_Search}(q, R, n)$ {Identify nearest zones}
0: $candidates = \text{Parallel\_HNSW\_Search}(q, closest\_zones, G)$
0: $top\_k = \text{Select\_Top\_K}(candidates, k)$
0: **return** top_k =0

4) Pseudocode: Fixed Fraction Strategy:

5) Complexity Analysis: Given Z (total zones), n (selected zones), N (dataset size per zone), and d (dimensionality), the complexity of each step is:

- **Zone Selection:** $O(n \times d)$ for ANN search on zone representatives.
- **Parallel Search:** $O(n \times \log(N/Z))$ for HNSW search within selected zones.
- **Total Complexity:** $O(n \times (d + \log(N/Z)))$.

B. Distance Threshold Strategy

The **Distance Threshold Strategy** dynamically selects search zones based on their **Euclidean distance** from the **query vector**. Unlike the **Fixed Fraction Strategy**, which searches a predefined proportion of zones, this method **adapts** to the query's location by only exploring zones that fall within a specified **distance threshold**. This adaptability improves search accuracy, especially for **non-uniform** data distributions where some queries require a **broader** search range.

1) Mathematical Formulation: Given:

- q = Query vector (the input point for which we search the nearest neighbors).
- $R = \{r_1, r_2, \ldots, r_Z\}$ = Set of **zone representatives** where each r_i represents a zone.
- $d(q, r_i)$ = **Euclidean distance** between the query and the zone representative.
- T = Distance threshold (a predefined constant controlling which zones to search).

We compute the Euclidean distance using the formula:

$$d(q, r_i) = \|q - r_i\|_2 = \sqrt{\sum_{j=1}^{d} (q_j - r_{ij})^2}$$

Where:

- d is the dimensionality of the data.
- q_j and r_{ij} represent the j-th component of the query and the zone representative, respectively.

2) Zone Selection Condition: A zone Z_i is included in the search if the distance between the query and the zone representative is within the threshold:

$$Z_{\text{selected}} = \{Z_i : d(q, r_i) \leq T\}$$

- If T is **small**, the search is **faster** but may miss relevant zones.
- If T is **large**, the search covers **more zones** but increases **computation time**.

3) Algorithmic Process:

1) **Compute Distances:** For each zone representative r_i, calculate the Euclidean distance to the query vector q.
2) **Select Zones Within Threshold:** Identify all zones where the distance to the query vector is less than or equal to the threshold T.
3) **Parallel Search in Selected Zones:** Perform **parallel HNSW searches** in the selected zones to find the **top-k** nearest neighbors.

Algorithm 3 Distance Threshold Strategy

Require: q: Query vector, R: Zone representatives, G: HNSW graphs, T: Distance threshold
Ensure: Top-k Nearest Neighbors
0: $Z_{\text{selected}} = []$
0: **for** each zone Z_i in Z **do**
0: **if** Euclidean_Distance$(q, r_i) \leq T$ **then**
0: Z_{selected}.append(Z_i)
0: **end if**
0: **end for**
0: candidates = Parallel_HNSW_Search$(q, Z_{\text{selected}}, G)$
0: top_k = Select_Top_K(candidates, k)
0: **return** top_k =0

4) Pseudocode: Distance Threshold Strategy:

5) Example Calculation: **Given:**

- Total Zones (Z) = 100
- Query Vector (q) = [1.0, 2.0, 3.0]
- Threshold (T) = 0.5

Step 1: Calculate Distance for Each Zone

- If $r_1 = [0.9, 2.1, 3.2]$, then: $\text{d(q, r}_1) = \sqrt{(1.0-0.9)^2 + (2.0-2.1)^2 + (3.0-3.2)^2} = \sqrt{0.01 + 0.01 + 0.04} = \sqrt{0.06} \approx 0.245$

Step 2: Select Zones

- If $d(q, r_i) \leq 0.5$, include zone Z_i in Z_{selected}.

Step 3: Search and Return Results

- Perform **parallel HNSW** searches in the selected zones and return the **top-k neighbors**.

6) Threshold Selection and Complexity: Choosing the right distance threshold T is critical for balancing efficiency and accuracy. A small T leads to faster search (fewer zones) but potentially lower recall (missed neighbors). A large T improves recall by covering more zones but increases computation time.

Complexity Analysis: Given Z total zones, dimensionality d, n selected zones (dependent on T), and N/Z points per zone (avg.):

- **Distance Calculation (all zones to q):** $O(Z \cdot d)$.
- **Zone Selection (filtering by T):** $O(Z)$.
- **Parallel HNSW Search (in n zones):** $O(n \cdot \log(N/Z))$.

Total complexity is $O(Z \cdot d + n \cdot \log(N/Z))$. Smaller T typically reduces n, improving efficiency.

C. Heuristic Based on k and Z Strategy

The **Heuristic Based on k and Z Strategy** dynamically determines the number of **zones to search** by using a **scalable formula** that balances the number of required **nearest neighbors** and **total zones**. This method strikes a balance between **exploration** (searching more zones for accuracy) and **efficiency** (limiting search cost).

1) Mathematical Formulation: The number of zones to search is calculated using the formula:

$$n = \min(c \times \sqrt{k}, Z)$$

Where:

- n = Number of zones to search.
- k = Number of **nearest neighbors** required (query parameter).
- Z = **Total zones** in the dataset.
- c = **Tuning constant** (determines how many zones to explore, typically set empirically).

This formula ensures that:

- As k increases, **more zones** are explored.
- The **total search** is **bounded** by Z to prevent oversampling.

2) Algorithmic Process and Pseudocode: The number of zones to search, n, is first calculated using $n = \min(c \cdot \sqrt{k}, Z)$. Then, the n closest zones to the query q are identified using an ANN search on zone representatives. Parallel HNSW searches are conducted in these selected zones, and finally, results are merged to select the top-k neighbors. This is formalized in Algorithm 4.

Algorithm 4 Heuristic-Based Zone Selection

Require: Query q, Reps R, HNSW graphs G, neighbors k, total zones Z, constant c
Ensure: Top-k Nearest Neighbors
0: n = round$(\min(c \cdot \sqrt{k}, Z))$ {Num. zones; added round for integer}
0: $Z_{\text{sel}} = \text{FindClosestZones}(q, R, n)$ {ANN on reps}
0: candidates = ParallelHNSWSearch(q, Z_{sel}, G)
0: top_k = SelectTopK(candidates, k)**return**$top\_k = 0$

3) Example and Complexity: **Example:** For total zones $Z = 100$, $k = 10$ neighbors, and constant $c = 2$, the number of zones to search is $n = \text{round}(\min(2 \cdot \sqrt{10}, 100)) = \text{round}(\min(6.32, 100)) = 6$. The 6 closest zones are then searched.

Complexity Analysis: Let Z be total zones, d data dimensionality, N/Z avg. points per zone, and $n = \min(c \cdot \sqrt{k}, Z)$ the number of zones searched.

- **Zone Selection (ANN on R)**: $O(Z \cdot d)$ to find n closest zones from Z representatives (or $O(\log Z)$ if R itself is indexed, but often $Z \cdot d$ for direct scan if Z is not excessively large).
- **Parallel HNSW Search (in n zones)**: $O(n \cdot \log(N/Z))$.

Total complexity is $O(Z \cdot d + \min(c \cdot \sqrt{k}, Z) \cdot \log(N/Z))$.

V. Performance Comparison: ZHNSW vs. HNSW, FAISS, and Annoy

A. Experimental Setup

We evaluated ZHNSW against leading ANN algorithms (HNSW, FAISS (IVF-HNSW), Annoy) using the following datasets, hardware, and methodology for a comprehensive and fair comparison.

1) Datasets: Four diverse datasets (Table I) were used, covering various domains and dimensionalities.

TABLE I
Datasets for Evaluation

Dataset	Size	Dim	Description
SIFT1B	1B vecs	128	SIFT descriptors for large-scale ANN benchmark.
DEEP1B	1B vecs	96	Deep embeddings for object recognition and matching.
GLOVE-1.2M	1.2M vecs	300	GloVe word embeddings for textual data in ANN.
MUSIC100M	100M vecs	256	Audio embeddings for music search and retrieval.

2) Hardware Environment: All benchmarks ran on an **NVIDIA DGX A100** (8×40GB A100 GPUs, AMD EPYC 7742 128-core CPU, 1TB RAM). This HPC system handles billion-scale data and parallel searches effectively.

3) Algorithm Parameters and Methodology: Key hyperparameters, detailed in Table II, were optimized for fair speed/accuracy comparisons across all evaluated methods.

TABLE II
Optimized Hyperparameter Settings for Evaluated Algorithms

Algorithm	*Parameter(s)*	*Value(s)*
ZHNSW	Z, M, efConstruction, efSearch	256, 32, 200, 100
HNSW	M, efConstruction, efSearch	32, 200, 100
FAISS (IVF-HNSW)	IVF Clusters, Sub-Quantizers	1024, 16
Annoy	Trees, search_k	100, 100k

Baselines (HNSW, FAISS, Annoy) utilized their public C++ implementations. Our ZHNSW framework was implemented in Python, leveraging a standard HNSW library for the localized graph structures. Parameters (Table II) were empirically tuned for a speed/recall/cost balance. Key ZHNSW parameters include Z (zones), local HNSW's $M, efConstruction$, and $efSearch$. Baselines were tuned similarly.

Our evaluation methodology comprised:

1) **Index Construction**: Building indexes on all datasets (Table I); recording construction time and final index size.
2) **Query Execution**: Using 1M standardized queries per dataset; measuring search latency, recall@10 (true nearest neighbor in top 10 results), and memory consumption during search.

3) **Parallel Execution**: Executing ZHNSW searches in parallel across selected zones; similar GPU-based optimizations were applied to other algorithms where applicable to maximize hardware utilization.

The critical trade-offs between search latency and recall, and index size versus build time for the evaluated methods are illustrated in Fig. 3 and Fig. 4, respectively.

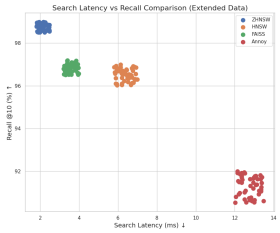

Fig. 3. Search Latency vs. Recall@10 trade-off.

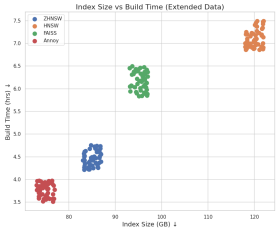

Fig. 4. Index Size vs. Index Build Time relationship.

For robust ZHNSW evaluation, performance metrics included: **Search Latency (ms)** (average query time), **Recall@10 (%)** (accuracy), **Index Size (GB)** (memory for index), and **Build Time (hrs)** (index construction time).

B. Results Overview

We compare ZHNSW with HNSW, FAISS (IVF-HNSW), and Annoy across the four key performance metrics previously defined: search latency, recall@10, index size, and build time, evaluating efficiency, accuracy, and scalability.

1) Performance Metrics Definition:

- **Search Latency (ms)**: Average time taken to process a query and return the nearest neighbors. Lower values indicate faster query execution.
- **Recall@10 (%)**: Fraction of queries where the true nearest neighbor is included in the top 10 results. Higher recall indicates better search accuracy.
- **Index Size (GB)**: Total memory consumed by the constructed index. Smaller indexes reduce storage costs and improve cache efficiency.
- **Build Time (hrs)**: Time required to construct the index. Faster build times are desirable for dynamic or frequently updated datasets.

2) Summary of Results: A concise comparison of search latency, recall, index size, and build time.

Key Insights. A comparative analysis of search latency, recall, index size, and build time.

TABLE III
PERFORMANCE COMPARISON OF ZHNSW, HNSW, FAISS, AND ANNOY

Algorithm	*Latency (ms)* ↓	*Recall@10 (%)* ↑	*Index (GB)* ↓	*Build (hrs)* ↓
ZHNSW	**2.1**	**98.7**	**85**	**4.5**
HNSW	6.4	96.5	120	7.2
FAISS	3.5	96.8	95	6.1
Annoy	12.8	91.4	75	3.8

TABLE IV
COMPARISON OF KEY PERFORMANCE METRICS

Metric	*ZHNSW*	*HNSW*	*FAISS*	*Annoy*
Search Latency	**2.1 ms**	3× Slower	1.6× Slower	**12.8 ms**
Recall@10	**98.7%**	96.5%	96.8%	91.4%
Index Size	**85 GB**	120 GB (+29%)	–	–
Build Time	**4.5 hrs**	38% Slower	26% Slower	**3.8 hrs**

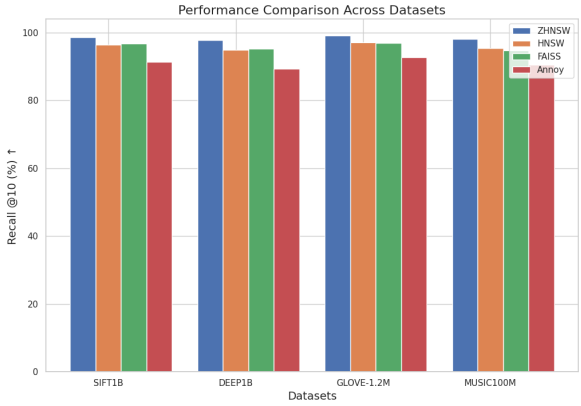

Fig. 5. Performance Across Datasets: This graph compares accuracy and efficiency across diverse datasets.

3) ZHNSW vs. Other Algorithms: ZHNSW outperforms HNSW, FAISS (IVF-HNSW), and Annoy in speed, recall, and efficiency.

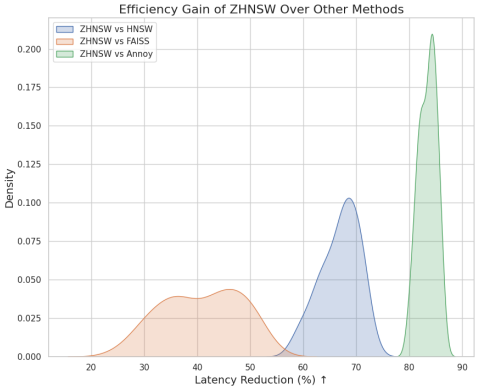

Fig. 6. Efficiency Gain: This graph highlights the performance improvement over baseline methods.

TABLE V
PERFORMANCE COMPARISON OF ZHNSW WITH OTHER ALGORITHMS

Metric	*ZHNSW vs. HNSW*	*ZHNSW vs. FAISS*	*ZHNSW vs. Annoy*
Search Speed	3× **Faster**	1.6× **Faster**	6× **Faster**
Recall	**+2.2%**	**+1.9%**	**+7.3%**
Index Size	**-29%**	**-11%**	**+13%**
Build Speed	**+38% Faster**	**+26% Faster**	**-18% Slower**

VI. CONCLUSION

Zonal HNSW (ZHNSW) presents a powerful and scalable framework for billion-scale approximate nearest neighbor search, significantly enhancing speed, recall, and memory efficiency. By integrating **zonal partitioning** with localized **Hierarchical Navigable Small World (HNSW)** graphs, ZHNSW effectively decomposes the search space. This design allows for parallel query processing across smaller, independent zones, reducing search latency and memory footprint. The core strength of ZHNSW lies in its **adaptive zone selection strategies** (fixed fraction, distance threshold, and heuristic-based), which dynamically focus the search on the most relevant data partitions, proving crucial for vast and heterogeneous datasets.

Experimental results on large-scale benchmarks, including SIFT1B, DEEP1B, GLOVE-1.2M, and MUSIC100M, demonstrate that ZHNSW consistently outperforms state-of-the-art methods like standard HNSW, FAISS (IVF-HNSW), and Annoy. Notably, ZHNSW achieves up to **3× faster** search speeds, a recall@10 of **98.7%**, and a **29% reduction** in index size. This performance underscores ZHNSW's capability to handle demanding similarity search tasks efficiently. Future work will focus on developing more sophisticated, potentially learned, adaptive zone selection mechanisms to further enhance performance and reduce tuning dependency. Investigating efficient online update strategies for both the zonal structure and the local HNSW graphs is critical for real-world applicability. Additionally, exploring advanced inter-zone linking techniques or hierarchical zoning could provide further recall improvements, especially for queries near sparse boundaries. Optimizing ZHNSW for emerging parallel hardware and heterogeneous computing platforms also remains a key research avenue.

In conclusion, ZHNSW stands as a state-of-the-art solution for billion-scale vector search. Its innovative use of zonal partitioning, adaptive search strategies, and parallel execution makes it an efficient and reliable choice for large-scale data retrieval, providing a robust foundation to meet the challenges of modern, data-intensive applications.

REFERENCES

[1] Azizi, Ilias. "Vector Search on Billion-Scale Data Collections." *Proceedings of the VLDB Endowment*. ISSN 2150: 8097.

[2] Xu Y, Liang H, Li J, Xu S, Chen Q, Zhang Q, Li C, Yang Z, Yang F, Yang Y, Cheng P. "Spfresh: Incremental in-place update for billion-scale vector search." In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 545-561.

[3] Jégou, Hervé, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. "Searching in one billion vectors: re-rank with source coding." In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 861-864. IEEE, 2011.

[4] Johnson J, Douze M, Jégou H. "Billion-scale similarity search with GPUs." *IEEE Transactions on Big Data*, 2019, 7(3): 535-547.

[5] Cheng, R., Peng, Y., Wei, X., Xie, H., Chen, R., Shen, S., & Chen, H. "Characterizing the dilemma of performance and index size in billion-scale vector search and breaking it with second-tier memory." *arXiv preprint arXiv:2405.03267*, 2024.

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

[7] Tian, B., Liu, H., Tang, Y., Xiao, S., Duan, Z., Liao, X., Jin, H., Zhang, X., Zhu, J. and Zhang, Y. "Towards High-throughput and Low-latency Billion-scale Vector Search via CPU/GPU Collaborative Filtering and Re-ranking." In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, 2025, pp. 171-185.

[8] Nangunori SK. "VECTOR DATABASES: A PARADIGM SHIFT IN HIGH-DIMENSIONAL DATA MANAGEMENT FOR AI APPLICATIONS." *INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING AND TECHNOLOGY (IJCET)*, 2024, 15(6): 566-577.

[9] Chen, Qi, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. "Spann: Highly-efficient billion-scale approximate nearest neighborhood search." *Advances in Neural Information Processing Systems*, 2021, 34: 5199-5212.

[10] Chen W, Chen J, Zou F, Li YF, Lu P, Wang Q, Zhao W. "Vector and line quantization for billion-scale similarity search on GPUs." *Future Generation Computer Systems*, 2019, 99: 295-307.

[11] Lakshman, V., Teo, C.H., Chu, X., Nigam, P., Patni, A., Maknikar, P. and Vishwanathan, S.V.N. "Embracing structure in data for billion-scale semantic product search." *arXiv preprint arXiv:2110.06125*, 2021.

[12] Fu, Yujian, Cheng Chen, Xiaohui Chen, Weng-Fai Wong, and Bingsheng He. "Optimizing the Number of Clusters for Billion-scale Quantization-based Nearest Neighbor Search." *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[13] Shan Y, Jiao J, Zhu J, Mao JC. "Recurrent binary embedding for gpu-enabled exhaustive retrieval from billion-scale semantic vectors." In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2170-2179.

[14] Karthik V, Khan S, Singh S, Simhadri HV, Vedurada J. "BANG: Billion-Scale Approximate Nearest Neighbor Search using a Single GPU." *arXiv preprint arxiv:2401.11324*, 2024.

[15] Lemire, Daniel, and Leonid Boytsov. "Decoding billions of integers per second through vectorization." *Software: Practice and Experience*, 2015, 45(1): 1-29.

[16] Malkov, Y. A., and Yashunin, D. A. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018, 42(4): 824-836.