

# Accelerating Vector Search at Scale: BAM-ANN with Batch-Aware Memory-Disk Hybrid Indexing

M M Mahabubur Rahman  
Computer Science  
Texas State University  
San Marcos, TX, USA  
toufik@txstate.edu

Jelena Tešić  
Computer Science  
Texas State University  
San Marcos, TX, USA  
jtesic@txstate.edu

**Abstract**—With the rapid advancement of Large Language Models (LLMs), Approximate Nearest Neighbor Search (ANNS) in vector databases has become a cornerstone of modern AI infrastructure. ANNS excels at efficiently retrieving similar objects from billion-scale datasets, achieving remarkable success in delivering fast, high-recall searches. However, in-memory ANNS algorithms are prohibitively expensive when scaling to large datasets, creating a growing need for cost-effective memory-disk hybrid solutions. This paper introduces BAM-ANN, an efficient graph-based memory-disk hybrid solution for batch query processing. Specifically, we introduce three key innovations: (1) a multi-tiered indexing strategy with iterative balanced clustering to efficiently manage billion-scale data under limited memory resources, (2) a learning-driven pruning mechanism that eliminates redundant I/O and computations while preserving high accuracy, and (3) a batch-aware query organization scheme that further accelerates the search process. Experimental results demonstrate that BAM-ANN achieves 90% recall@100 in approximately one millisecond, outperforming state-of-the-art solutions like SPANN and DiskANN by 1.25x to 2.5x in speed.

**Index Terms**—Similarity search, High-dimensional indexing, Deep descriptors search, Information retrieval, Vector search, Graph-based index, Billion-scale search

## I. INTRODUCTION

Approximate Nearest Neighbor Search (ANNS) in high-dimensional spaces identifies objects most similar to a given vector. This is a fundamental problem in algorithmic research and has numerous applications in various fields, including data mining [1], information retrieval [2], and AI-driven recommendation systems [3]. With the rapid advancements in large language models (LLMs) [4], ANNS supported by vector databases has become a cornerstone of modern AI infrastructure. Domain-specific knowledge of different data formats, such as text, images, and audio, is embedded into high-dimensional vector spaces and stored in vector databases as feature vectors. When a user submits a query, the ANNS engine retrieves objects that are semantically similar to the query, providing highly relevant and context-aware results for further processing by the LLM. There are many Approximate Nearest Neighbor Search (ANNS) algorithms in the literature, which include methods like graph-based models [5], hash-based techniques [6], tree structures [7], and quantization approaches [8], which mainly focus on building efficient indexing strategies. Although many ANNS algorithms rely on in-memory indices to achieve fast and accurate searches, this

comes at the cost of significantly higher memory usage. For instance, commercial recommendation systems like Alibaba often need terabyte-scale memory to handle billions of vectors [9]. This demand for extensive resources drives up the total cost of ownership (TCO), including the cost of acquiring and running the system. As a result, it becomes challenging to scale ANNS services for datasets with hundreds of billions of vectors. The scalability challenges arise primarily from the extensive memory footprint needed to maintain these indices in main memory. Therefore, there is a growing demand for hybrid ANNS solutions that efficiently combine minimal memory usage with cost-effective disk storage to handle large-scale datasets. Only a handful of approaches focus on hybrid ANNS solutions, notably SPANN [8], an inverted index approach, DiskANN [10], and HM-ANN [11], both of which are graph-based methods. Due to their cost-efficiency, high recall, and low latency, SPANN and DiskANN have emerged as state-of-the-art solutions for indexing billion-scale datasets.

In this paper, we present **BAM-ANN**, a highly efficient graph-based memory-disk hybrid architecture designed for billion-scale Approximate Nearest Neighbor Search (ANNS) with advanced query optimization. BAM-ANN delivers high recall, low latency, and scalable performance by intelligently organizing queries and minimizing unnecessary I/O through learning-based pruning, all while operating under limited memory resources. The key idea of BAM-ANN is to leverage a multi-tiered design that separates memory-resident centroids from disk-resident clusters, enabling efficient batch query processing. Specifically, we propose three key innovations to address the above challenges.

First, we employ an iterative balanced clustering algorithm that partitions billion-scale datasets into evenly sized clusters. Only the centroids are stored in memory, while the full clusters remain on disk. During index construction, BAM-ANN builds proximity graphs for each cluster and stores them in memory, significantly reducing memory overhead while maintaining fast access paths for search. This multi-tiered design allows the system to scale to billions of vectors using minimal RAM, as only small, representative structures are retained in memory.

Second, we incorporate a learning-based pruning strategy that predicts the most relevant clusters for each batch, thereby eliminating redundant I/O and unnecessary computations.

This module plays a critical role in scaling BAM-ANN to enterprise-scale workloads, such as recommendation engines or web-scale search systems, where thousands of queries may arrive concurrently.

Third, we propose a batch-aware query arrangement mechanism that groups incoming queries into mini-batches based on their proximity to centroids. Each mini-batch is then routed to the relevant graph indexes, and corresponding clusters are read in a single I/O operation, thereby improving both spatial locality and access efficiency. A result buffer temporarily accumulates intermediate neighbors, which are then refined to yield high-accuracy final results. Our experimental results show that BAM-ANN achieves 90% recall@100 in approximately one millisecond, outperforming state-of-the-art methods in both speed and accuracy.

## II. BACKGROUND AND RELATED WORK

Given a dataset  $X \in \mathbb{R}^{n \times m}$ , where the dataset consists of  $n$  vectors, each with  $m$ -dimensional features, and a query vector  $q \in \mathbb{R}^m$ , the objective of vector search is to find the nearest neighbor  $p^* \in X$ , defined as:  $p^* = \arg \min_{p \in X} \text{Dist}(p, q)$ , where  $\text{Dist}(p, q)$  denotes the distance metric between vectors  $p$  and  $q$ . Similarly, this definition can be extended to finding  $K$ -nearest neighbors. Due to the substantial computational cost and high query latency of exhaustive search, Approximate Nearest Neighbor Search (ANNS) algorithms are designed to efficiently retrieve approximate  $K$ -nearest neighbors from large datasets within acceptable time constraints. Most ANNS algorithms in the literature focus on achieving fast, high-recall searches in memory. These include hash-based methods [6], tree-based methods [7], graph-based methods [5], [12], and hybrid approaches [13], [14]. However, as vector datasets grow exponentially, memory limitations have emerged as a critical bottleneck for supporting large-scale vector search. Only a few approaches focus on ANNS solutions for billion-scale datasets, which aim to minimize memory costs.

SPANN [8] employs an inverted index approach that stores only the centroid points of the posting lists in memory, while the larger posting lists are kept on disk. This architecture achieves low latency and high recall by minimizing disk access and enhancing the quality of the posting lists. During index-building, a hierarchical balanced clustering algorithm ensures an even distribution of posting list lengths. Additionally, the posting lists are expanded by incorporating points from the closure of their respective clusters. In the search phase, a query-aware mechanism dynamically eliminates irrelevant posting lists, thereby improving the efficiency of the search process. IVFADC [15], FAISS [14], and IVFOADC+G+P [16] partition the vector space into  $K$  Voronoi regions using KMeans clustering, employ vector quantization techniques to reduce memory usage, and perform searches only within the areas closest to the query. Similarly, the Inverted multi-index (IMI) [17] uses Product Quantization (PQ) [18] to compress vectors. First, the IMI splits the feature space into multiple orthogonal subspaces. Then, it constructs a separate codebook for each subspace: the complete feature space is

represented as the Cartesian product of these subspaces. Multi-LOPQ [19] enhances this approach by utilizing locally optimized PQ codebooks to encode displacements within the IMI structure, while GNO-IMI [20] optimizes IMI by using non-orthogonal codebooks to generate centroids. Although these methods significantly reduce memory requirements to less than 64GB for one billion 128-dimensional vectors, they suffer from low recall performance. Specifically, the recall@1 is limited to approximately 60% due to the lossy nature of data compression.

Graph-based methods, such as DiskANN [10] and HM-ANN [11], adopt hybrid solutions for approximate nearest neighbor search. DiskANN stores PQ-compressed vectors in memory while keeping the navigating graph and full-precision vectors on disk. During a query, the graph is traversed best-firstly based on the distances of quantized vectors, followed by reranking candidates using the distances of the full-precision vectors. However, lossy data compression affects recall quality, even though reranking with full-precision vectors can recover some missing candidates. Additionally, the high-cost random disk accesses restrict the number of graph traversals and candidate rerankings. On the other hand, HM-ANN utilizes heterogeneous memory by storing pivot points, promoted during a bottom-up phase, in fast memory while placing the navigable small-world graph in slower memory without compressing the data. While this avoids the drawbacks of lossy compression, it results in significantly higher memory consumption, requiring more than 1.5 times the fast memory compared to DiskANN.

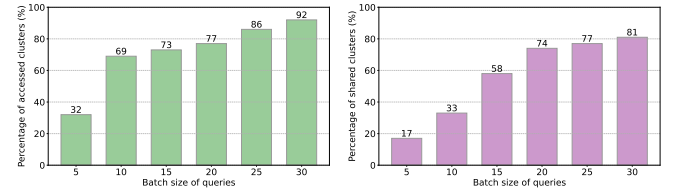


Fig. 1. Left: data locality among queries. Right: shared data access by queries.

## III. MOTIVATION

In this section, we present optimization opportunities for memory-disk hybrid solutions and motivate the design of BAM-ANN.

### A. Optimization opportunities for Hierarchical indexing

Clustering-based hierarchical indexes [8], [21] have proven to be an effective approach for reducing the computational cost of Approximate Nearest Neighbor Search (ANNS) while maintaining high accuracy. This method involves partitioning a dataset into multiple shards and grouping highly similar vectors into the same cluster. Indexes are then built for each cluster, allowing faster traversal during search queries. Existing solutions like SPANN leverage Hierarchical Balanced Clustering (HBC), while DiskANN employs k-means clustering to divide the data. However, these methods face challenges when dealing with extremely large datasets that cannot fit into memory or when frequent index updates are required. These limitations restrict the applicability of traditional clustering approaches like HBC and k-means. To address these issues,

BAM-ANN introduces a multi-tiered index structure that adopts an iterative mini-batch k-means clustering approach, which balances cluster sizes to ensure efficient partitioning and reduces the memory overhead. BAM-ANN allows the clustering model to be stored for future updates, which is beneficial for recommendation and search engines, which often rely on consistent embedding models for vectorization. Therefore, the overall data distribution tends to remain stable, and new data can be clustered without degrading the quality of existing clusters. Next, we enhance the capabilities of hnsplib [22] to create a multi-tiered index, where the centroids of each cluster serve as intermediate nodes and the associated HNSW graphs act as the leaf nodes. This multi-tiered index is stored in memory, while the cluster data is kept on disk. The iterative and balanced nature of the process significantly accelerates both the initial indexing and subsequent index updates. It ensures the system can efficiently handle dynamic datasets while maintaining high performance and accuracy. We discuss the index structure in detail in Section IV-A.

#### B. Optimization opportunities for query processing

To improve the efficiency of handling ANNS queries, we investigated the data access patterns of index clusters. Using the subset of the DEEP1B dataset [23], which contains 100 million vectors, we partitioned the dataset into 100 clusters of approximately equal size. We then generated 1,000 queries to identify the most similar vectors, configuring each query to search the top 10 nearest clusters by default. Our analysis focused on two key aspects of cluster access patterns. First, Figure 1(left) illustrates how the variation of the batch size influences the percentages of clusters accessed by different queries. Second, Figure 1(right) analyzes the overall frequency of the cluster access. Thus, we make following observations:

**Observation 1:** Many clusters are accessed by different queries over time, demonstrating strong data locality. For example, as Figure 1(left) shows, when the batch size exceeds 30, 92% of all clusters are accessed by at least one query. The figure indicates that queries naturally distribute across the dataset, ensuring that most clusters are utilized effectively.

**Observation 2:** There is a high degree of overlap in cluster access among queries. As shown in Figure 1(right), 81% of the queries share access to at least one cluster, and certain popular clusters are frequently accessed by multiple queries within a batch. This overlap suggests that optimizing query processing to reuse in-memory clusters can significantly improve efficiency. These findings emphasize the importance of designing query processing schemes that exploit data locality and cluster reuse. By reusing frequently accessed in-memory clusters, we can reduce redundant data loading and improve the overall performance of ANNS query handling, especially in scenarios involving large-scale datasets and high query throughput. We explain the query processing scheme in detail in Section IV-B.

#### C. Optimization opportunities for data pruning

The difficulty of queries can vary significantly, causing the search scope for each query to change dynamically [24]. As a result, configuring a fixed number of clusters for all queries is

inefficient, as it may lead to unnecessary scanning of irrelevant clusters. Inverted indexing methods [8] typically create smaller clusters and a larger number of centroids, where much of the computational overhead comes from traversing these centroids.

In SPANN, instead of searching a fixed number of  $K$  posting lists for every query, a posting list is searched only if the distance between its centroid and the query is comparable to the distance between the query and the closest centroid. While this approach improves upon fixed cluster search by reducing redundant scans, it still falls short of fully leveraging each query’s unique nature.

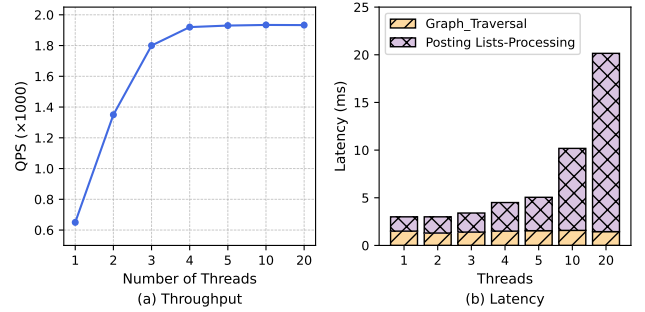


Fig. 2. (a) Throughput and (b) Latency in SPANN retrieval phase.

As shown in Figure 2(a), SPANN retrieval peaks at four CPU threads and fails to scale further. The overall query latency consists of (i) in-memory graph traversal and (ii) SSD-based posting list access. Figure 2(b) shows that overall latency rises almost linearly with thread count, mainly due to I/O contention from simultaneous access to large SSD posting lists, while graph traversal latency remains stable.

To address this limitation, we introduce a learning-based approach to determine the optimal number of clusters for each query dynamically and a batch-aware query organization scheme that further accelerates the search process. We use the Light Gradient Boosting Machine (LightGBM) model [25] due to its lightweight design and ability to deliver fast and efficient performance, making it ideal for large-scale datasets. This adaptive method ensures that the search process is tailored to each query, significantly improving throughput and reducing unnecessary computations. We then reorganize queries by cluster, ensuring that all queries targeting a specific cluster are executed together when its sub-index is loaded into memory. This strategy greatly reduces the latency associated with posting list processing. The details of the pruning process and query optimization process are described in Section IV-B.

### IV. DESIGN AND IMPLEMENTATION

In this section, we outline the design of BAM-ANN, providing a detailed explanation of the index construction process and the retrieval mechanism. Figure 3 illustrates the architecture of BAM-ANN, consisting of an offline index-building module and an online vector search module.

#### A. Offline processing

Similar to traditional ANNS algorithms, BAM-ANN constructs its indices in an offline process shown in Figure 4.

**Iterative Balanced Clustering** Let the dataset  $X \in \mathbb{R}^{n \times m}$

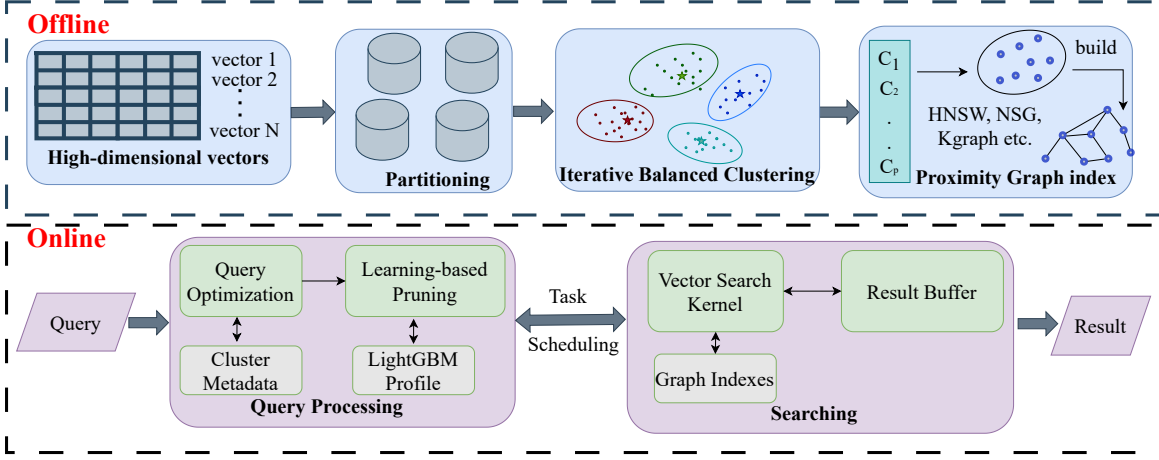


Fig. 3. BAM-ANN Architecture

represent  $n$  data points, each with  $m$ -dimensional features. The objective is to partition  $X$  into  $k$  clusters, denoted as  $C_1, C_2, \dots, C_k$ , such that intra-cluster similarity is maximized, while ensuring balanced sizes. This can be formulated as:

$$\min_{\{C_i\}} \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|_2^2 + \lambda \cdot \text{Var}(|C_i|), \quad (1)$$

where  $\mu_i$  is the centroid of cluster  $C_i$ , and  $\text{Var}(|C_i|)$  penalizes size imbalance.  $\lambda$  is a regularization coefficient.

First, we partition the dataset into small, manageable chunks that can be loaded into memory. Then BAM-ANN employs an iterative mini-batch  $k$ -means clustering approach to iteratively cluster each chunk, minimizing memory overhead. To maintain balanced cluster sizes, we redistribute data points from overloaded clusters to underpopulated ones based on their distances to other centroids. The clustering process is performed offline to ensure no impact on real-time query handling. The resulting balanced clusters  $\{C_1, C_2, \dots, C_k\}$  are stored efficiently, with only the centroids  $\{\mu_1, \mu_2, \dots, \mu_k\}$  retained in memory. The clustering model parameters and cluster structures are saved for future use, enabling efficient updates when new data becomes available.

**Multi-tiered Index Construction** To further enhance indexing efficiency, we extend the functionality of `hnswlib` [22] to construct a multi-tiered index. The centroids of the clusters  $\{\mu_1, \mu_2, \dots, \mu_k\}$  serve as the intermediate nodes, forming the top layer of the hierarchy. The HNSW graphs  $\{G_1, G_2, \dots, G_k\}$ , corresponding to each cluster, act as the leaf nodes in the hierarchy. The multi-tiered index is stored in memory (shown in Figure 4), allowing rapid traversal of centroids and efficient identification of relevant clusters

for a given query. Meanwhile, the detailed cluster data  $X_i$  remains on disk, optimizing memory usage. The cluster data corresponding to a node in the multi-tiered index can be dynamically loaded from the disk during the search process.

### B. Online processing

The online processing phase is the retrieval process executed in response to a query.

**Query Optimization** As shown in Figure 5(a), the conventional retrieval approach in clustering-based methods loads clusters into memory individually for each query. While this method supports query-level parallelism, it can lead to repetitive access to the same cluster during multi-cluster searches. This redundancy becomes inefficient when handling a large number of queries. From our observations in Section III-B, there is an 81% overlap in the neighbor distribution among clusters when the batch size exceeds 30. To exploit this locality, we reorganize the batch of queries by cluster, as shown in Figure 5(b).

This reordering allows us to load a cluster into memory once and process all relevant queries on that cluster simultaneously, maintaining a heap to store local results. Once all relevant clusters are processed, the top- $k$  results are extracted and returned. This approach significantly reduces redundant access and improves efficiency.

**Learning-based Pruning** Searching for a fixed number of clusters can still result in accessing irrelevant clusters. For instance, if a query is close to the center of a specific cluster, the top- $k$  nearest neighbors are likely to lie within that cluster alone, making searches in other clusters unnecessary. To address this, we introduce a learning-based pruning approach to filter out irrelevant clusters dynamically. As illustrated in Figure 5(c), this pruning mechanism reduces the number of searches and cluster accesses (shown in grey).

In our experiments using the DEEP1B dataset with 10,000 queries and 100 clusters, the pruning process reduced the number of searches by 38% while still accurately retrieving the top  $k$  results. To dynamically prune irrelevant clusters, we employ *LightGBM*, a lightweight and efficient gradient-boosting framework designed for high performance on large datasets. *LightGBM* is particularly suitable for this task due to

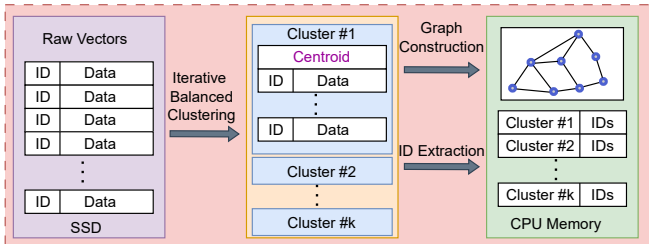


Fig. 4. Multi-tiered indices in BAM-ANN

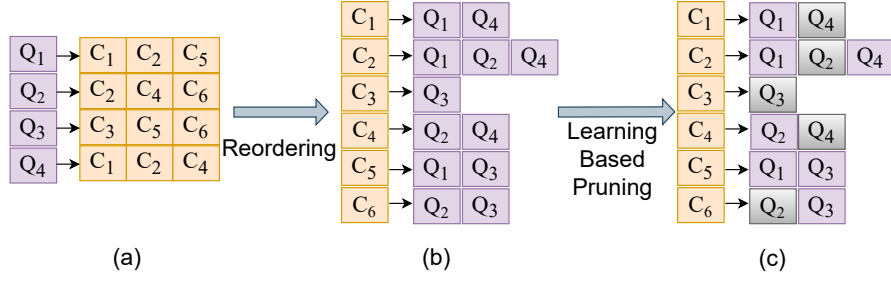


Fig. 5. Query processing layout with (a) Conventional, (b) Optimized parsing, and (c) Learning-based pruning approach

its fast training speed, low memory consumption, and ability to handle large-scale input features efficiently. The input features for the LightGBM model include the query vector, its spatial relationship with the clusters, and the total number of clusters. To effectively capture these relationships, we use *relative distances* as key features. Let  $D_1$  represent the distance between the query and the closest cluster (top-1), and  $D_k$  denote the distance to the  $k$ -th nearest cluster. A key discriminative feature used for LightGBM is:  $\phi(q) = \left\{ \frac{D_2}{D_1}, \frac{D_3}{D_1}, \dots, \frac{D_k}{D_1} \right\}$

Which encodes the relative spatial relationships between the query and the centroids. The LightGBM model uses this feature vector to prune irrelevant clusters. The ratio  $D_k/D_1$  is a critical input feature for training the model. This ratio abstracts the spatial correlation between the query and the clusters, enabling the model to identify and prune irrelevant clusters accurately. To prepare the training data, we sample *1 million vectors* from each billion-scale dataset. We perform an exhaustive search for ground truth labels to determine the *minimum number of clusters* that cover the top- $k$  nearest neighbors. This preprocessing step is efficiently executed on an *NVIDIA RTX A6000 GPU* and completes in less than two hours. We train the LightGBM model using a *learning rate of 0.05* over *500 iterations*, which takes only *5 minutes* on a standard CPU. LightGBM’s optimized training process ensures rapid model generation, making it ideal for deployment in large-scale systems. The trained LightGBM model is highly efficient and lightweight, with a memory footprint of approximately *1 MB*. During inference, the model processes queries in just a few *tens of microseconds*, allowing it to prune unnecessary clusters in real-time dynamically. LightGBM allowed us to substantially reduce the number of cluster accesses, thus improving the efficiency of the retrieval step.

**Vector Search Engine** We enhance the HNSW search algorithm to operate efficiently within the multi-tiered index. Once queries are grouped into batches based on their assigned clusters, the search algorithm navigates through the corresponding HNSW graphs. To maximize efficiency, a task scheduler is introduced to enable cluster-level parallelism, ensuring that multiple clusters can be processed simultaneously. During this process, the local search results for each query are stored in a heap. Once all relevant clusters, as determined by the LightGBM model, have been searched, the algorithm consolidates the local results. It returns the final top- $k$  nearest neighbors from the heap, concluding the search process.

## V. EVALUATION

**System Information** All experiments were carried out on an Ubuntu 20.04.3 server with an 11th-generation Intel® Core™ i9-11900K @ 3.5GHzX16 CPU with 128GB RAM and NVIDIA GeForce RTX 3070 8GB mem GPU.

TABLE I  
DATASETS

Dataset	Dimension	Base Size	Query set	Data Type
DEEP1B [23]	96	358 GB	10,000	Image
SIFT1B [26]	128	119 GB	10,000	Image
SPACEV1B [27]	100	93 GB	29,316	Web Search

**Datasets** The benchmark data are three billion-point benchmark datasets: DEEP1B [23], SIFT1B [26], and SPACEV1B [27]. The details of these datasets are summarized in Table I. We assess the performance of BAM-ANN by comparing it with state-of-the-art memory-disk hybrid methods, SPANN and DiskANN.

**Metrics** To evaluate the efficiency and accuracy of the search process, we measure recall and latency. Recall quantifies the accuracy of the search results by measuring the proportion of true nearest neighbors successfully retrieved. We report average per-query latency across a batch, which is standard in ANNS literature and facilitates comparison with prior works (SPANN, DiskANN). For interactive live systems, the wall-clock latency is equivalent to the entire batch time, since all queries must complete before results are returned. BAM-ANN is primarily designed for high-throughput scenarios, but smaller batch sizes can be employed to trade throughput for lower end-to-end delay. By analyzing Latency against recall, we can assess how the system balances search speed with accuracy, showcasing its ability to deliver high recall while maintaining low latency. This trade-off is crucial for large-scale applications where speed and accuracy are essential.

**Performance** In Figure 6, we present the performance of BAM-ANN compared to SPANN and DiskANN across the three benchmark datasets. The results demonstrate that BAM-ANN significantly outperforms SPANN and DiskANN, particularly when operating under tight latency constraints. For the SPACEV1B dataset, DiskANN struggles to achieve high recall (90%) within 4ms, highlighting its limitations in low-latency scenarios. In contrast, BAM-ANN and SPANN achieve a recall of 90% in just around 1ms, showcasing their efficiency in balancing speed and accuracy. In the DEEP1B dataset, BAM-ANN demonstrates its superiority by being more than 2.5 times faster than DiskANN while achieving a recall of

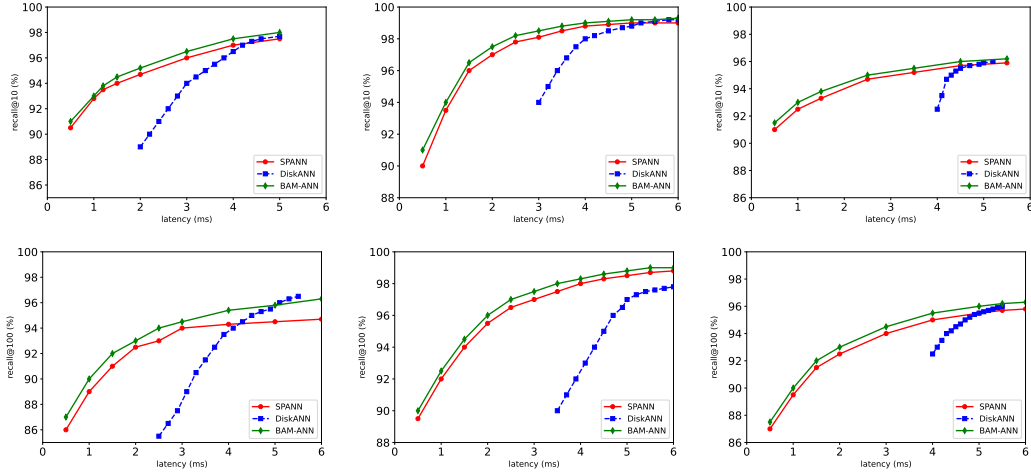


Fig. 6. Recall vs latency on (left to right) DEEP1B, SIFT1B and SPACEV1B datasets.

90%. This substantial improvement highlights BAM-ANN’s ability to deliver high-quality results at a significantly reduced latency. Overall, the results confirm that BAM-ANN excels in scenarios with strict latency budgets, outperforming state-of-the-art methods like SPANN and DiskANN in efficiency and recall across all evaluated datasets.

## VI. ABLATION STUDY

To understand the contribution of each component in BAM-ANN, we conduct an ablation study by systematically removing or modifying individual modules. This analysis highlights the effect of balanced clustering, query optimization, and learning-based pruning on the system’s overall performance. We evaluate the following simplified variants:

- **w/o Multi-tiered Indexing:** This variant uses standard MiniBatchKMeans instead of our balanced clustering approach. It results in uneven cluster sizes, leading to skewed load distribution and higher latency.
- **w/o Query optimization:** Queries are processed individually rather than in batches. This results in repeated disk reads for the same clusters, increasing I/O overhead.
- **w/o Learning-based Pruning:** A baseline where each query searches exactly five nearest clusters, without any dynamic pruning or learning-based optimization.

TABLE II  
ABLATION STUDY ON BAM-ANN USING DEEP1B.

Variant	Recall @100	Latency (ms)	QPS	Clusters Accessed
BAM-ANN (Full)	0.942	<b>1.12</b>	<b>8934</b>	3.2
w/o Multi-tiered Indexing	0.925	1.38	7231	<b>3.1</b>
w/o Query Optimization	0.942	2.87	6210	3.2
w/o Learning-based Pruning	<b>0.945</b>	1.68	5342	5.0

For experiments, we partitioned DEEP1B into 100 balanced clusters ( $\approx 3.58$  GB each). At runtime, the LightGBM pruning reduced the average number of clusters scanned to 3.2 clusters per query. The framework is not limited to 100 clusters; balanced clustering can scale to thousands or millions of partitions, supporting datasets at the 100-billion scale by recursively applying hierarchical clustering and maintaining only centroids and graphs in memory.

Table II reports the performance of each variant. The results confirm that each component in BAM-ANN significantly improves either recall, latency, or both. Removing multi-tiered indexing leads to reduced recall and higher latency due to uneven workload distribution. Disabling query optimization increases disk I/O overhead, resulting in significantly slower query processing. Removing LightGBM-based pruning increases unnecessary cluster access, resulting in a higher latency. Learning-based pruning trades a small amount of recall for a significant gain in latency and throughput (Table II). These findings justify the integrated design of BAM-ANN and its suitability for billion-scale vector search tasks with strict performance constraints.

## VII. CONCLUSION

As large language models (LLMs) continue to advance, Approximate Nearest Neighbor Search (ANNS) in vector databases has become a key part of modern AI systems. ANNS helps retrieve similar items quickly from massive datasets, ensuring fast and accurate results. However, scaling in-memory ANNS algorithms for very large datasets is costly, creating a need for more affordable memory-disk hybrid approaches. This paper presents BAM-ANN, a graph-based memory-disk hybrid solution for efficient batch query processing. During index building, BAM-ANN uses an iterative balanced clustering algorithm to distribute clusters evenly in size. It then creates neighborhood graph indexes for each cluster, storing only the cluster centroids and graph structures in memory while keeping most of the data on disk. During the search phase, BAM-ANN groups queries into mini-batches based on their proximity to centroids. A learning-based pruning strategy dynamically narrows the search to the most relevant clusters, reducing unnecessary data access. Our experiments show that BAM-ANN achieves 90% recall@100 in about one millisecond, outperforming state-of-the-art methods like SPANN and DiskANN, with speed improvements ranging from 1.25x to 2.5x. These results demonstrate that BAM-ANN offers a balanced solution that is efficient, accurate, and scalable, making it ideal for large-scale vector search tasks.

## ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation (Award No. 2334268).

## REFERENCES

- [1] Y. Tagami, “Annexml: Approximate nearest neighbor search for extreme multi-label classification,” in *Proceedings of the ACM KDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 455–464.
- [2] J. Zhang, Z. Liu, W. Han, S. Xiao, R. Zheng, Y. Shao, H. Sun, H. Zhu, P. Srinivasan, W. Deng, Q. Zhang, and X. Xie, “Uni-retriever: Towards learning the unified embedding based retriever in bing sponsored search,” in *Proceedings of the ACM KDD International Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4493–4501.
- [3] J.-T. Huang, A. Sharma, S. Sun, L. Xia, D. Zhang, P. Pronin, J. Padmanabhan, G. Ottaviano, and L. Yang, “Embedding-based retrieval in facebook search,” in *Proceedings of the ACM KDD International Conference on Knowledge Discovery and Data Mining*, 2020, pp. 2553–2561.
- [4] Y. Ge, W. Hua, K. Mei, J. Tan, S. Xu, Z. Li, Y. Zhang *et al.*, “Openagi: When llm meets domain experts,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [5] M. M. Rahman and J. Tešić, “Stratified graph indexing for efficient search in deep descriptor databases,” *International Journal of Multimedia Information Retrieval*, vol. 13, no. 3, p. 35, 2024.
- [6] B. Zheng, Z. Xi, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen, “Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search,” *Proceedings of the VLDB Endowment*, vol. 13, no. 5, pp. 643–655, 2020.
- [7] A. J. Gallego, J. R. Rico-Juan, and J. J. Valero-Mas, “Efficient k-nearest neighbor search based on clustering and adaptive k values,” *Pattern recognition*, vol. 122, p. 108356, 2022.
- [8] Q. Chen, B. Zhao, H. Wang, M. Li, C. Liu, Z. Li, M. Yang, and J. Wang, “Spann: Highly-efficient billion-scale approximate nearest neighborhood search,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 5199–5212, 2021.
- [9] C. Fu, C. Xiang, C. Wang, and D. Cai, “Fast approximate nearest neighbor search with the navigating spreading-out graph,” *Proc. VLDB Endow.*, vol. 12, no. 5, p. 461–474, jan 2019. [Online]. Available: <https://doi.org/10.14778/3303753.3303754>
- [10] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi, “Diskann: Fast accurate billion-point nearest neighbor search on a single node,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [11] J. Ren, M. Zhang, and D. Li, “Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 672–10 684, 2020.
- [12] N. Ono and Y. Matsui, “Relative nn-descent: A fast index construction for graph-based approximate nearest neighbor search,” in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 1659–1667.
- [13] M. Rahman and J. Tešić, “Hybrid approximate nearest neighbor indexing and search (hannis) for large descriptor databases,” in *2022 IEEE International Conference on Big Data*, 2022, pp. 3895–3902.
- [14] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with GPUs,” *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.
- [15] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, “Searching in one billion vectors: Re-rank with source coding,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 861–864.
- [16] D. Baranchuk, A. Babenko, and Y. Malkov, “Revisiting the inverted indices for billion-scale approximate nearest neighbors,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 202–216.
- [17] A. Babenko and V. Lempitsky, “The inverted multi-index,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 6, pp. 1247–1260, 2014.
- [18] H. Jegou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2010.
- [19] Y. Kalantidis and Y. Avrithis, “Locally optimized product quantization for approximate nearest neighbor search,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 2321–2328.
- [20] A. Babenko and V. Lempitsky, “Efficient indexing of billion-scale datasets of deep descriptors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2055–2063.
- [21] M. Zhang and Y. He, “Grip: Multi-store capacity-optimized high-performance nearest neighbor search for vector search engine,” in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2019, pp. 1673–1682.
- [22] 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, 2018.
- [23] D. Baranchuk, A. Babenko, and Y. Malkov, “Revisiting the inverted indices for billion-scale approximate nearest neighbors,” *CoRR*, vol. abs/1802.02422, 2018. [Online]. Available: <http://arxiv.org/abs/1802.02422>
- [24] P. Zhang, B. Yao, C. Gao, B. Wu, X. He, F. Li, Y. Lu, C. Zhan, and F. Tang, “Learning-based query optimization for multiprobe approximate nearest neighbor search,” *The VLDB Journal*, vol. 32, no. 3, pp. 623–645, 2022.
- [25] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” *Advances in neural information processing systems*, vol. 30, 2017.
- [26] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, “Searching in one billion vectors: Rerank with source coding,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011, pp. 861–864.
- [27] Microsoft, “Spacev1b,” <https://github.com/microsoft/SPTAG/tree/main/datasets/SPACEV1B>, 2021.