LEANN: A Low-Storage Vector Index for Personal Devices

Yichuan Wang[1] Shu Liu[1] Zhifei Li[1] Yongji Wu[1] Ziming Mao[1] Yilong Zhao[1] Xiao Yan[2] Zhiying Xu[3]
Yang Zhou[1] Ion Stoica[1] Sewon Min[1] Matei Zaharia[1] Joseph Gonzalez[1]

Abstract

Embedding-based search is widely used in applications such as recommendation and retrieval-augmented generation (RAG). Recently, there has been a growing demand to support these capabilities over personal data stored locally on devices. However, maintaining the necessary data structure associated with the embedding search is often infeasible due to its high storage overhead. For example, indexing 100 GB of raw data requires 150 to 700 GB of storage, making local deployment impractical. Reducing this overhead while maintaining search quality and latency becomes a critical challenge. In this paper, we present LEANN, a storage-efficient approximate nearest neighbor (ANN) search index optimized for resource-constrained personal devices. LEANN combines a compact graph-based structure with an efficient on-the-fly recomputation strategy to enable fast and accurate retrieval with minimal storage overhead. Our evaluation shows that LEANN reduces index size to under 5% of the original raw data – up to $50\times$ smaller than standard indexes – while achieving 90% top-3 recall in under 2 seconds on real-world question-answering benchmarks.

1. Introduction

With the recent advances in AI (Lin et al., 2022; Izacard et al., 2021), embedding-based search now significantly outperforms traditional keyword-based search methods (Karpukhin et al., 2020; Zamani et al., 2023) across many domains such as question answering, recommendation, and large-scale web applications such as search engines (Craswell et al., 2020; Zhang et al., 2018). These systems rely on dense vector representations to capture semantic similarity and use approximate nearest neighbor (ANN) search to retrieve relevant results efficiently. Recently, there has been growing interest in enabling such capabilities on edge devices like laptops or phones, enabling applications like personalized search, on-device assistants, and privacy-preserving retrieval over local data (Wang & Chau, 2024; Lee et al., 2024; Yin et al., 2024).

[1]UC Berkeley [2]CUHK [3]AWS. Correspondence to: <yichuan_wang@berkeley.edu,wuyongji317@gmail.com>.

Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

However, ANN data structures introduce substantial storage overheads, often 1.5 to $7\times$ the size of the original raw data (Shao et al., 2024). While such overheads are acceptable in large-scale web application deployments, they pose a significant bottleneck when deploying ANN search on personal devices or when using large datasets. For example, a $2\times$ storage overhead on a personal laptop is impractical. To make ANN search viable in these settings, we seek to reduce storage overhead to under 5% of the original data size. At the same time, any such reduction must preserve high search accuracy while maintaining reasonable search latency to ensure responsive, real-time search experiences.

Existing solutions, however, fall short of this goal. Most ANN indices store full embeddings and index metadata on disk (Wang et al., 2021), requiring terabytes of storage to index hundreds of gigabytes of documents, far exceeding the capacity of edge devices. While compression techniques such as product quantization (PQ) (Jégou et al., 2011) can reduce storage, they often come at the cost of degraded search accuracy or require increased search latency to achieve comparable results.

In this paper, we tackle the challenge of reducing ANN storage overhead and present LEANN, a novel graph-based vector index designed for storage-constrained environments. Built on top of Hierarchical Navigable Small World (HNSW) (Malkov & Yashunin, 2018), a widely adopted, state-of-the-art graph-based ANN index, LEANN introduces system and algorithm optimizations that reduce total index storage to under 5% of the original data size, while preserving low query latency and high retrieval accuracy. At its core, LEANN is driven by two key insights.

The first insight is that in graph-based indexes like HNSW, a single query typically explores only a small subset of the embedding vectors to identify its nearest neighbors. As such, instead of storing these embeddings on disk, we can recompute them on-the-fly at search time. However, naive recomputation can still incur a high latency overhead. To address this challenge, LEANN introduces a two-level traversal algorithm that interleaves an approximate and an exact distance queue, while prioritizing the most promising candidates in the search process, thus reducing the number of recomputations. Additionally, LEANN also incorporates a dynamic batching mechanism that aggregates embedding computations across search hops, improving GPU utilization and thus minimizing recomputation latency.

However, even without storing embeddings, the index metadata (e.g., graph structure) itself can lead to non-trivial storage overhead relative to the original data size. For example, a typical HNSW index uses a node degree of 64, meaning each node stores 64 neighbor links. With 4 bytes per link, this results in 256 bytes of metadata per node, which normally accounts for more than 25% storage overhead of a common 256-token document chunk (Shao et al., 2024).

The second insight is that much of the graph index metadata is redundant: not all nodes and edges contribute equally to search accuracy. Based on this observation, LEANN introduces a high-degree preserving graph pruning strategy that removes low-utility edges while preserving high-degree "hub" nodes that are essential for maintaining effective search paths. By retaining only structurally important components of the graph, LEANN significantly reduces the size of the index without sacrificing the quality of the retrieval.

We implement LEANN on top of FAISS (Douze et al., 2025) and evaluate it on four popular information retrieval (IR) benchmarks: NQ (Kwiatkowski et al., 2019), HotpotQA (Yang et al., 2018), TriviaQA (Joshi et al., 2017), and GPQA (Rein et al., 2024). These benchmarks have been widely used in evaluations of information retrieval systems. Our experiments span both an NVIDIA A10 workstation (NVIDIA, 2025) and an M1-based Mac (AWS, 2025b). The results show that LEANN reduces storage consumption by more than 50× compared to state-of-the-art indexes while achieving competitive latency to achieve high accuracy. In summary, we make the following contributions:

- We conduct the first study on enabling low-latency, high-accuracy vector search with minimal storage overhead.
- We present LEANN, a compact graph-based ANN index that prunes redundant graph metadata by prioritizing preserving high-degree nodes, and avoids storing embeddings by recomputing them on the fly. To minimize recomputation latency, LEANN also introduces a two-level search strategy with dynamic batching.
- We show that LEANN can deliver 90% top-3 recall using less than 5% storage overhead relative to the raw data size, while the end-to-end search time is still less than 2 seconds on four benchmarks and various hardware platforms.

2. Background and Motivation

In this section, we provide background on approximate nearest neighbor (ANN) search indexes, with a focus on graph-based approaches, and outline the requirements for deploying vector index on consumer devices.

2.1. ANN Search

Vector search systems use high-dimensional embeddings to support semantic search over unstructured data. The core task is top-k nearest neighbor (NN) search: given a dataset $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^m$ and a query $q \in \mathbb{R}^m$, the goal is to retrieve the k closest vectors in X to q under a distance metric $\text{Dist}(\cdot, \cdot)$. Formally, this amounts to selecting $\mathcal{S} = \text{Top}_k(\text{Dist}(x, q))$ for $x \in X$.

Exact NN search is often too slow at scale, so approximate nearest neighbor (ANN) methods (Malkov & Yashunin, 2018; Lempitsky, 2012) are used to trade off accuracy for speed. Effectiveness is typically measured by Recall@k, defined as Recall@k $= |\mathcal{S} \cap \mathcal{S}'|/k \geq R_{\text{Target}}$, where $\mathcal{S}'$ is the set returned by the ANN algorithm. Many applications such as retrieval-augmented generation (RAG) require high recall (e.g., ≥ 0.9) to preserve quality (Shen et al., 2024).

To accelerate ANN search, vector indexes organize embeddings using data structures that reduce the number of comparisons required. Generally, a vector index consists of two primary components: (1) the stored embedding vectors themselves, representing the data, and (2) the index structure (such as graph connections or cluster assignments) built upon these vectors to expedite the search. Two common classes of ANN indices are:

Cluster-based Index. (e.g., IVF (Lempitsky, 2012)) partitions the dataset into clusters (cells) using algorithms such as K-means (Choo et al., 2020). Each cluster stores vectors that are semantically similar. At query time, only the most relevant clusters are searched. While efficient, these methods often incur high recomputation costs to traverse all relevant dense clusters.

Graph-based Index. (Indyk & Motwani, 1998) constructs a proximity graph by connecting each vector to its nearest neighbors. These indexes are among the most effective for vector search, requiring traversal of significantly fewer embeddings to achieve the target recall compared to alternatives such as IVF (Malkov & Yashunin, 2018). We refer to the search procedure as the best-first search (BFS) algorithm for handling ANN queries, which we detail in Section 2.2.

2.2. Best-first Search (BFS) in Graph-based index

In Algorithm 1, we illustrate how BFS operates on a graph-based index. The search begins by placing the entry node p into a *min-priority queue* C, referred to as the *candidate queue*, which prioritizes nodes closer to the query vector x_q. In each iteration (Lines 4 to 9), the algorithm selects the closest node c from C and explores its neighbors. For each unvisited neighbor n, we extract its embedding, compute its

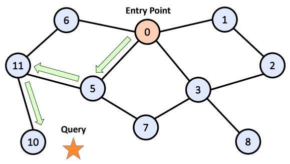


Figure 1. Best-First Search in graph-based index

distance to the query x_q, and insert n into the visited set V, the candidate queue C, and the result set R.

The search terminates when the candidate queue C becomes empty or when the closest node in C is farther from the query than the farthest node in the result set R, indicating that further exploration is unlikely to improve the outcome. The parameter ef controls how many candidates the search algorithm considers internally before returning the top-K results. ef serves as a *quality knob*: increasing ef enables the algorithm to explore more candidates, thereby improving recall at the cost of higher latency. An illustrative example of this traversal process is shown in Figure 1.

2.3. Deploying vector index on Consumer Devices

Local Vector Index System Requirement. Consumer devices, such as smart home appliances and personal workstations (Wang & Chau, 2024; Lee et al., 2024; Seemakhupt et al., 2024; Yu et al., 2025), face strict storage constraints (Xue et al., 2024; ObjectBox Ltd., 2024; Totino, 2025a). Yet, many generative AI tasks rely on similarity search over dense embeddings, which can be up to 7× larger than the original data (Zilliz AI FAQ, 2025; Microsoft Learn, 2025; Shao et al., 2024). Unlike datacenter servers with abundant memory (Castro, 2024; Douze, 2020), edge devices must share limited storage with apps and media (Totino, 2025b), making it impractical to store large-scale, uncompressed indexes.

At the same time, these devices often serve user-facing tasks like large-scale document retrieval (Lee et al., 2024; Wang & Chau, 2024) or offline semantic recall (Cai et al., 2024), where latencies under 10 seconds are acceptable. Only delays beyond 10 seconds start to impact usability.

This combination of tight storage budgets (e.g., using less than 5% of the original raw data size) and relatively relaxed latency requirements motivates a distinct design space for on-device vector search: a highly storage-efficient VDB that leverages on-device compute (e.g., GPU) to achieve acceptable time-to-accuracy (on the order of seconds).

Existing System Limitations on Consumer Devices. Most vector search indexes like HNSW and IVF are designed with an exclusive focus on retrieval time to accuracy, but not on minimizing storage footprint. While these approaches enable fast and accurate search, it become infeasible on consumer devices, where storage resources are limited.

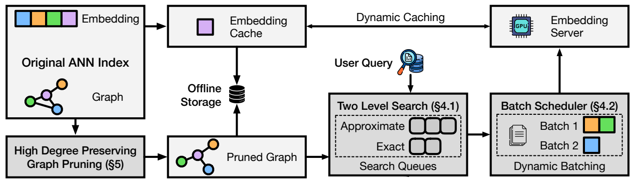


Figure 2. LEANN system architecture and workflow.

Quantization-based methods, such as PQ (Jégou et al., 2011), are the main approach for reducing storage by approximating embeddings using compact codebooks. While these techniques can shrink the embedding size dramatically, the inherent information loss from this lossy compression often degrades retrieval accuracy. This degradation means that critical vector distinctions can be permanently lost during quantization, making it impossible to achieve high target recall using only the compressed data, a limitation we experimentally demonstrate in Section 6 and which is documented in the literature (Subramanya et al., 2019).

To our knowledge, there is no prior system for vector index that has explicitly targeted consumer devices where storage footprint is a first-class objective. Our goal in this work is to design a vector search system that significantly reduces storage overhead, both for embeddings and index structures, while meeting the latency and recall requirements.

3. Overview

In this section, we give an overview of the core techniques and show how LEANN incorporates these techniques.

Graph-based Recomputation. In the HNSW structure that LEANN relies on, each query only requires recomputing a small number of nodes, i.e., those in the candidate set C defined in Algorithm 1. This observation motivate LEANN to recompute a subset of embeddings on-the-fly instead of pre-storing all of them. Concretely, instead of loading pre-computed embeddings as in Line 9, we modify the system to recompute them during query execution without changing any algorithm.

Main Techniques. There are two challenges associated with this paradigm. First, graph-based recomputation offers storage savings by not storing all embeddings on disk, naive recomputation of embeddings at query time still results in high search latency. Second, while LEANN eliminates the need to store dense embeddings by recomputing them on-the-fly, the remaining graph metadata, particularly node connectivity information, can still contribute significantly to overall storage usage (e.g., >10%).

LEANN offers two main techniques to address the challenges mentioned before. First, LEANN uses a two-level graph traversal algorithm and a dynamic batching mechanism to reduce recomputation latency (Section 4). Second, LEANN deploys a high degree preserving graph pruning technique to greatly reduce the storage needed for graph metadata (Section 5).

System Workflow. The end-to-end workflow incorporating the optimizations discussed above is shown in Figure 2. Given a dataset of items, LEANN first computes the embeddings of all items to build a vector index for the dataset using an off-shelf graph-based index.

After the index is built, LEANN discards the embeddings (dense vectors) of the items, while pruning the graph for offline storage with our high degree preserving graph pruning algorithm (Section 5). The pruning algorithm aims to preserve important high degree nodes, as we observe that node access patterns are highly skewed in practice: a small subset of nodes, often "hub" nodes of high degree, are frequently visited, while many others contribute little to search quality. To serve a user query at runtime, LEANN applies a two-level search algorithm (described in Section 4.1) to traverse the pruned graph, identifying and prioritizing promising nodes for efficient exploration. These selected nodes are then sent to the embedding server (an on-device component utilizing the original embedding model for recomputation, as illustrated in Figure 2) to obtain their corresponding embeddings. To further improve GPU utilization and reduce latency, LEANN employs a dynamic batching strategy to schedule embedding computation tasks on the GPU (Section 4.2). Furthermore, when additional disk space is available, LEANN leverages it to cache "hub" nodes. At runtime, LEANN computes embeddings only for nodes not present in the disk cache, while loading cached nodes directly from disk.

4. Efficient Graph-based Recomputation

4.1. Two-Level Search with Hybrid Distance

Since the primary bottleneck in graph-based recomputation is the number of recomputations, a natural optimization strategy is to approximate the search path in order to reduce the recomputation overhead. The Two-Level Search introduces a multi-fidelity framework for distance computation, guiding the search process by strategically varying computational intensity across different stages. This approach balances efficiency and accuracy by employing lightweight approximate calculations for initial candidate filtering, followed by exact computations only for the most promising candidates.

Algorithm 2 illustrates our approach. During each expansion step, we first apply a lightweight method to compute approximate distances for all neighboring nodes at Line 12. We maintain an approximate queue (AQ), a priority queue that stores approximate distances for all encountered nodes. Rather than computing exact distances for every neighbor of the expansion node, we selectively refine only the top-ranking candidates in AQ at Line 14. Since higher-ranked nodes in AQ are more likely to lead toward the top-k targets, this strategy reduces computation while maintaining search quality. To implement this strategy, we define a re-ranking ratio a and compute exact distances for the top $a\%$ of nodes in AQ, filtering out nodes already in EQ (Exact Queue) to avoid redundant computations.

In the final step of each iteration, we transfer nodes with computed exact distances from set M to the EQ, which serves as the candidate pool for subsequent expansions. This selective computation in Line 14 strategy significantly reduces overall processing requirements while maintaining high search quality.

The insight behind this approach is that re-ranking vectors with top approximate distances is sufficient to achieve high recall while effectively pruning nodes in opposite searching directions. Notably, approximate distance computation can be several orders of magnitude cheaper than recomputing exact embeddings. In our evaluation, we use PQ to perform approximate distance calculations, achieving an extremely efficient compression ratio (compressing 200GB of embeddings to just 2GB) while still providing reasonable accuracy to guide the search. This efficiency is possible because we still employ exact embedding calculations to compensate for any accuracy loss.

4.2. Dynamic Batching to Fully Utilize GPU

During the search process, GPU resources are often underutilized because each expansion step only triggers recomputation for a small number of nodes, typically equal to the degree of the current node v. This problem is further exacerbated when using the Two Level Search algorithm (see Line 16), where the candidate set is even more selective, resulting in smaller batch sizes. As a result, naive graph-based recomputation fails to meet the minimum batch size required to saturate GPU throughput, leading to inefficient use of hardware resources at runtime.

To address this, LEANN introduces a dynamic batching strategy that slightly relaxes the strict data dependency in best-first search in Algorithm 1. While this introduces minor staleness in the expansion order, it significantly increases the batch size for the embedding model, thereby reducing the end-to-end latency per query.

Specifically, LEANN breaks the strict data dependency in best-first search, where the current node to be expanded depends on the immediate results of the previous expansion, by dynamically collecting a group of the closest candidates from the priority queue. The algorithm accumulates their neighbors (i.e., the nodes requiring recomputation) until a

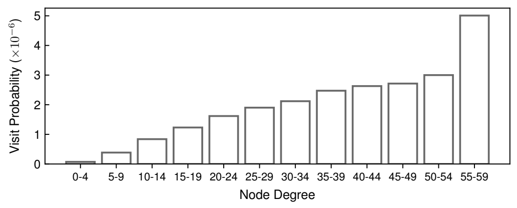


Figure 3. Node access probability per query

target batch size is reached, such as 64 for the A10 GPU. This batch size can be easily determined through lightweight offline profiling. This dynamic batching mechanism also integrates naturally with the Two-Level Search described in Section 4.1, where we simply accumulate the size of $|M|$ until it meets the predefined batch threshold.

5. Storage-optimized Graph Structure

With two-level search and dynamic batching to reduce recomputation latency, we now examine how LEANN minimizes graph metadata overhead through high-degree preserving pruning. As noted in Section 3, even when embeddings are recomputed on demand, the graph structure used for search can still introduce substantial storage cost. For example, in the datastore analyzed by (Severo et al., 2025), the index accounts for over 30% of total storage.

To address this, LEANN introduces a user-defined disk budget C. If the graph metadata exceeds this threshold, LEANN triggers a pruning algorithm that reduces edge count while preserving retrieval accuracy.

Our key insight is that preserving hub nodes is sufficient to retain search performance, inspired by the skewed node access pattern observed in Figure 3. These high-degree nodes serve as the backbone of the graph's connectivity, and maintaining their edges ensures navigability even under aggressive pruning. We formalize this in Algorithm 3.

On the one hand, we apply differentiated degree thresholds to nodes based on their importance. We impose stricter connectivity constraints on most nodes by reducing the number of connections to m (Line 10), while allowing a small fraction (i.e., $a\%$) of high degree nodes to retain higher connectivity with a threshold of M (Line 8). Given a storage budget C, LEANN automatically tunes the values of m and M through offline profiling across multiple datasets. Following insights from (Ren et al., 2020; Munyampirwa et al., 2024), we identify these high-influence nodes based on degree ordering, as shown in Line 4. Empirically, we find that setting $a\%$ to 2%, preserving only 2% of the highest-degree nodes, is sufficient to maintain high retrieval accuracy.

On the other hand, while we restrict the number of outgoing connections during node insertion, all nodes may still receive incoming connections up to the maximum threshold M (as shown in Line 13, not m). This design ensures that each node can form bidirectional edges with high-degree hub nodes, preserving graph navigability with minimal impact on search quality.

6. Evaluation

Workloads We construct a datastore for retrieval based on the RPJ-Wiki dataset (Computer, 2023), a widely used corpus containing 76 GB of raw Wikipedia text. Following prior work (Shao et al., 2024), we segment the text into 256-token passages and generate one embedding per passage using Contriever (Izacard et al., 2021), resulting in 768-dimensional vectors. In total, we process 60 million chunks, producing 171 GB of embedding data.

Besides retrieval itself, we also consider the predominant downstream task of RAG. We adopt the widely deployed LLaMA model family for generation and report downstream task accuracy with the *Llama-3.2-1B-Instruct* model . For evaluation, we adopt four standard benchmarks widely used in RAG and open-domain retrieval: NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), GPQA (Rein et al., 2024), and HotpotQA (Yang et al., 2018).

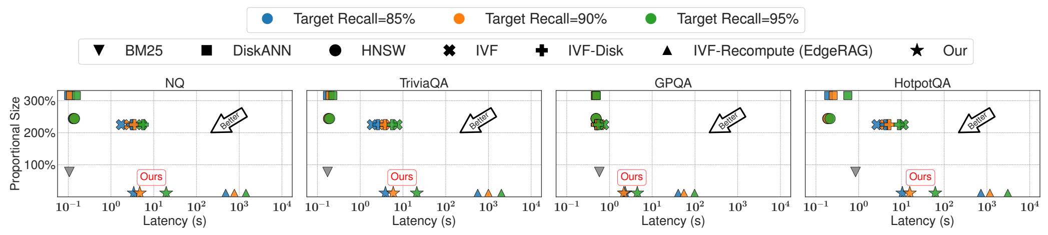
Testbed. We evaluate our system and baselines on two hardware platforms. The first is an NVIDIA A10 server hosted on an AWS g5.48xlarge instance (AWS, 2025a), equipped with a 96-core CPU and an NVIDIA A10G GPU. The second is a Mac environment, provided via an AWS EC2 M1 Mac instance (AWS, 2025b), featuring an Apple M1 Ultra processor (Arm64).

Metrics. We compare LEANN against alternative baselines in three main dimensions: storage, latency, and accuracy. For accuracy, we evaluate both the search (retrieval) accuracy and downstream task accuracy.

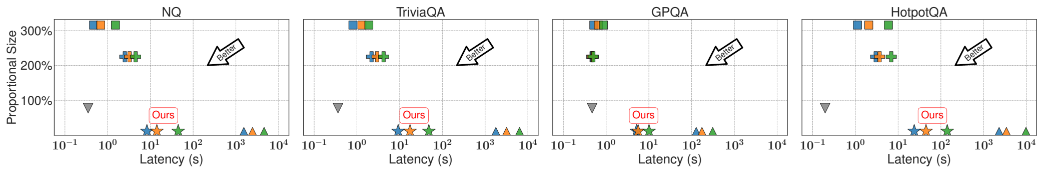
To evaluate retrieval accuracy, we report Recall@k as defined in Section 2. In open-domain settings, ground-truth labels for retrieved passages are typically unavailable. Following standard practice (Jégou et al., 2011; Schuhmann et al., 2021; Zhu et al., 2024), we use the results from exhaustive search as a proxy for ground truth. In our experiments, we set $k = 3$ following prior work standard setup (Shao et al., 2024; Asai et al., 2023), and report Recall@3.

To evaluate downstream RAG accuracy, we use Exact Match (EM) and F1 score. EM measures the fraction of predictions that exactly match the ground-truth answers, while F1 captures the harmonic mean of precision and recall, awarding partial credit for token-level overlap between predicted and ground-truth answers.

Baselines. We compare LEANN against the following: **HNSW (in-memory)** (Malkov & Yashunin,



(a) A10



(b) Mac

Figure 4. **[Main Result]:** Latency–storage trade-offs in RAG applications across four datasets and two hardware configurations. The y-axis indicates the storage overhead, defined as the size of the ANN index relative to the raw data size (76 GB of uncompressed text in our setup). We vary the target recall to evaluate latency under different retrieval accuracy levels. Since recall is not applicable to BM25, it appears as a single data point in each figure. Additionally, we omit the PQ-compressed method, as it fails to reach the target recall threshold despite being a vector-based approach. As shown in Figure 5, both BM25 and PQ result in poor downstream accuracy.

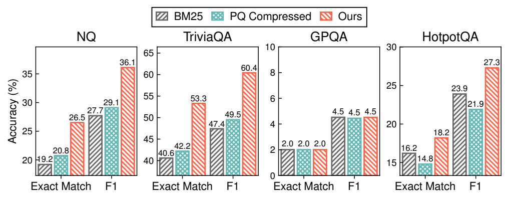


Figure 5. **[Main Result]:** Comparison of Exact Match and F1 scores for downstream RAG tasks across three methods: keyword search (BM25), PQ-compressed vector search, and our proposed vector search system. Our method is configured to achieve a target recall of 90%, while the PQ baseline is given extended search time to reach its highest possible recall. Here we use *Llama-3.2-1B* as the generation model.

2018) uses `faiss.IndexHNSWFlat` with M=30, $efConstruction$=128; **IVF (in-memory)** uses `faiss.IndexIVFFlat` with $nlist$=$\sqrt{N}$=8192 for a N = 60M datastore (Research, 2025); **DiskANN** (Subramanya et al., 2019) is a graph-based system with M=60, $efConstruction$=128, keeping only PQ tables in memory; **IVF-Disk** uses memory-mapped IVF via `faiss.contrib.ondisk`; **IVF-based recomputation** (Seemakhupt et al., 2024) omits embedding storage by recomputing them online using IVF; **PQ Compression** (Jégou et al., 2011) applies Product Quantization to compress embeddings while retaining graph structure; **BM25** (Craswell et al., 2021; Rekabsaz et al., 2021) is a classical lexical ranking baseline.

6.1. Main Results - Storage and Latency

Figure 4 presents the storage consumption and end-to-end RAG query latency across all baseline systems and LEANN. The results show that LEANN is the only system that reduces storage to less than 5% of the original raw text size while maintaining reasonable latency, which we discussed in Section 2.3, such as achieving 90% recall on GPQA in under 2 seconds.

We report storage consumption as a proportion of the raw text size (76GB), referred to as proportional size in Figure 4. Since all methods operate on the same fixed datastore, based on a given dataset (RPJ-Wiki), the storage consumption for each method remains constant across different hardware platforms and query datasets. The figure shows that HNSW stores all dense embeddings along with a graph structure, resulting in substantial storage overhead. DiskANN incurs even higher storage overhead due to its sector-aligned design: the data of each node, including its embedding (768×4 bytes) and edge list (degree 60, stored as 60×4 bytes), is padded to a 4 KB SSD sector, resulting in the highest storage footprint among all methods. On the other hand, IVF and IVF-Disk have similar storage overhead, dominated by the embedding file. The additional metadata (e.g., centroids) associated with the IVF index is relatively small and scales with $\sqrt{N}$, making its overhead negligible. For BM25, storage is determined by the vocabulary size and the associated posting lists. In our setting, the BM25 index size is comparable to that of the original corpus. In contrast, LEANN stores only a compact graph structure, resulting in less than 5% additional storage. Among the baselines, Edge-RAG achieves the lowest storage footprint, as it only stores the IVF centroids on disk, which adds negligible overhead.

For latency evaluation, we measure per-query latency under different target recall levels across all combinations of query datasets and hardware platforms. For BM25, we report a single number for its latency value using the default keyword search configuration. Unlike embedding-based search methods, BM25 is a lexical search technique and does not operate over dense embeddings. As a result, recall is not applicable for evaluating its effectiveness because it is defined based on approximate nearest neighbor retrieval. We omit results for HNSW and IVF on the Mac platform, as both methods require loading the full dense embedding matrix into memory, which leads to out-of-memory (OOM) errors. Specifically, the Mac system has 128GB of RAM, while the index size exceeds 170GB. We also exclude the PQ-compressed baseline, as it fails to achieve the target recall even with an arbitrarily long search time.

Figure 4 shows that LEANN consistently outperforms EdgeRAG, an IVF-based recomputation method, achieving significantly lower latency, ranging from 21.17× to 200.60×, across all the datasets and hardware platforms. This advantage is partly due to the asymptotic difference in recomputation complexity: the number of recomputed chunks in LEANN grows polylogarithmically with N, while it grows as $\sqrt{N}$ in Edge-RAG(Wang et al., 2021). Graph-based baselines such as HNSW and DiskANN represent upper bounds on latency performance, as they store all embeddings in RAM or on disk. While LEANN trades some latency for substantial storage savings, its performance remains well within an acceptable range. This latency degradation is acceptable for two main reasons as we discussed in Section 2.3: (1) second-level latency is acceptable for large-scale local document or image retrieval tasks, and (2) many downstream tasks on local devices, such as image or text generation, typically take over tens of seconds to complete (Contributors, 2025; Li et al., 2024), making the additional latency introduced by LEANN reasonable in practice.

6.2. Main Result - Accuracy for Downstream RAG Task

We evaluate the downstream accuracy across four query datasets, as shown in Figure 5. Although the PQ-compressed method fails to meet the target recall defined in Section 6.1, it still achieves approximately 20% recall on all datasets. We evaluate downstream accuracy using these low-quality retrieved results. For our method, we set the target recall level to 90% for retrieving the top-3 relevant documents and use BM25 with its default configuration for keyword matching. As illustrated in Figure 5, our method consistently achieves higher downstream accuracy across datasets except GPQA. This is because GPQA is attributed to the datastore being somewhat out-of-distribution (OOD) for GPQA, which primarily consists of graduate-level questions that are not well-supported by the retrieved documents. Additionally, the accuracy improvement on HotpotQA is smaller compared to the first two datasets. This is because HotpotQA requires multi-hop reasoning, whereas our current setup only performs single-hop retrieval, which provides limited benefit to downstream accuracy.

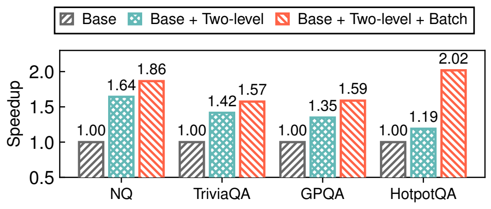


Figure 6. **[Ablation Study]:** Speedup achieved by different optimization techniques described in Section 4 when evaluated on four datasets to reach the same recall level on the A10 GPU. *Two-level* refers to the optimization in Section 4.1, while *Batch* corresponds to Section 4.2.

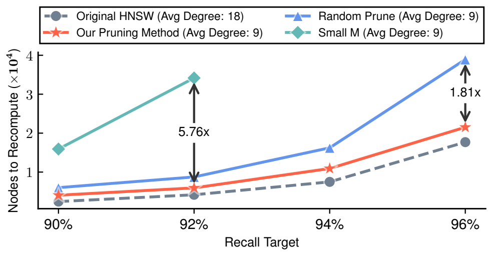


Figure 7. **[Ablation Study]:** Comparison of pruned graph quality against two heuristic methods and the upper bound. We vary the target recall and measure the number of nodes each method needs to recompute. The dashed gray line represents the original HNSW graph, which serves as the upper bound, with twice the storage (i.e., average degree) of the others.

6.3. Ablation Study

Ablation study on latency optimization technique. To evaluate LEANN's latency optimization techniques, we incrementally enable the components introduced in Section 4, using a fixed target recall across multiple datasets. The evaluation begins with a naive graph-based recomputation baseline. Incorporating the two-level hybrid distance computation strategy from Section 4.1 yields an average speedup of 1.40×, as it reduces the number of nodes requiring recomputation and allows for lightweight distance estimation without invoking the embedding server. Adding the dynamic batching technique further improves GPU utilization during recomputation, increasing the overall speedup to 1.76×. Among the datasets, HotpotQA benefits most from batching due to its need for a longer search queue to achieve the target recall, which enables more effective grouping of multi-hop requests.

Comparison with Alternative Graph Pruning Methods. We compare our graph pruning algorithm with two heuristic baselines and evaluate graph quality by measuring the number of embeddings that must be fetched to achieve a given recall target, as shown in Figure 7. In our setting, the end-to-end latency scales linearly with the number of embeddings that require recomputation, making this metric a strong proxy for retrieval latency. The two heuristic baselines are as follows: **(1) Random Prune**, which randomly removes 50% of the existing edges from the original graph; and **(2) Small M**, which directly constrains the maximum out-degree during graph construction, resulting in an average degree that is half that of the original graph.

We evaluate performance on the NQ dataset by varying the search queue length ef to determine the number of embeddings fetched at different recall targets. As shown in Figure 7, our pruning method introduced in Section 5 achieves performance comparable to the original unpruned graph, despite using only half the edges. It outperforms the Random Prune baseline by up to $1.18\times$ and the Small M baseline by up to $5.76\times$. We omit the Small M data points at 94% and 96% recall targets due to their poor performance.

Degree Distribution in Pruned Graphs. To better understand the effectiveness of our pruning method, we analyze the out-degree distribution of the original graph, our method, Random Prune, and Small M. As discussed in Section 5, our design explicitly aims to preserve high degree "hub" nodes in the graph. As shown in Figure 8, our method successfully retains a substantial number of high degree nodes, whereas the other two baselines fail to do so. This underscores the critical role of hub nodes in supporting efficient graph-based vector search, a finding that aligns with insights from prior work (Ren et al., 2020; Munyampirwa et al., 2024; Manohar et al., 2024).

Relaxing disk constraint. As discussed in Section 3, when disk storage constraints are relaxed, LEANN can materialize embeddings of high degree nodes to reduce recomputation overhead. For instance, as shown in Figure 9, storing only 10% of the original embeddings results in a $1.47\times$ speedup, with a cache hit rate of up to 41.9%. This high cache hit rate arises from the skewed access pattern characteristic of graph-based traversal . However, the observed speedup does not fully align with the hit rate due to the non-negligible loading overhead introduced by SSDs with limited bandwidth.

7. Related Work

General Vector Search. Vector search typically relies on IVF (Lempitsky, 2012), which clusters vectors and probes selected subsets, or proximity graphs (Malkov & Yashunin, 2018), which link similar vectors for traversal. Graph-based methods such as HNSW (Malkov & Yashunin, 2018), NSG (Fu et al., 2019), Vamana (Subramanya et al., 2019), and others (Chen et al., 2021; Fu et al., 2021; Munoz et al., 2019) generally achieve better accuracy-efficiency tradeoffs. Some work reduces graph size via learned neighbor selection (Zhang et al., 2020; Baranchuk & Babenko, 2019), though it is often impractical due to training overhead.

Resource-Constrained Vector Search. Reducing the memory footprint of vector search has attracted significant attention. Disk-based systems like DiskANN (Subramanya et al., 2019) store both vectors and graph structures on disk, using in-memory compressed codes for navigation. Starling (Wang et al., 2024) improves I/O for disk-resident graphs, while FusionANNS (Tian et al., 2025) leverages SSD, CPU, and GPU for cost-effective search. AiSAQ (Tatsuno et al., 2024) further reduces DRAM usage by keeping compressed codes on disk. EdgeRAG (Seemakhupt et al., 2024) avoids storing embeddings via online generation with an IVF index but suffers from large cluster storage and high recomputation cost at scale. Compression techniques like PQ (Jégou et al., 2011) and RabitQ (Gao & Long, 2024) reduce storage but often degrade accuracy under tight budgets. In contrast, LEANN combines on-the-fly recomputation with a graph index, using degree-aware pruning and optimized traversal for edge devices.

Vector Search Applications on Edge Devices. On-device vector search enables privacy-preserving, low-latency, and offline capabilities across various applications. On-device RAG systems ground language models in personal document collections while maintaining data privacy (Ryan et al., 2024; Wang & Chau, 2024; Lee et al., 2024; Zerhoudi & Granitzer, 2024). Personalized recommendation systems (Yin et al., 2024) match user profiles against item embeddings locally, while content-based search within image and video libraries leverages efficient on-device vision models (Ren et al., 2023). These applications motivate the design of LEANN to enable efficient and low storage overhead vector search on edge devices.

8. Conclusion

Similarity search over high-dimensional embeddings underpins many generative AI applications such as retrieval-augmented generation (RAG). However, enabling such capabilities on personal devices remains challenging due to the substantial storage required for storing embeddings and rich vector index metadata. In this paper, we present LEANN, a storage-efficient neural retrieval system that leverages *graph-based recomputation*. By combining a *two-level search algorithm* with *batch execution*, LEANN achieves efficient query processing without storing the full embedding set. Furthermore, we introduce a *high degree preserving pruning* strategy to reduce graph storage overhead while maintaining accuracy. Together, these techniques enable LEANN to operate with less than 5% of the original data size – achieving a $50\times$ storage reduction compared to existing methods – while maintaining fast and accurate retrieval.

References

Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2023.

AWS. Amazon EC2 G5 instance. https://aws.amazon.com/ec2/instance-types/g5, 2025a.

AWS. Amazon EC2 G5 instance. https://aws.amazon.com/ec2/instance-types/mac/, 2025b.

Baranchuk, D. and Babenko, A. Towards similarity graphs constructed by deep reinforcement learning. *arXiv preprint arXiv:1911.12122*, 2019.

Cai, D., Wang, S., Peng, C., et al. Recall: Empowering multimodal embedding for edge devices. arXiv:2409.15342, 2024.

Castro, P. Announcing cost-effective rag at scale with azure ai search. https://techcommunity.microsoft.com/blog/azure-ai-services-blog/announcing-cost-effective-rag-at-scale-with-azure-ai-search/4104961, 2024.

Chen, Q., Zhao, B., Wang, H., Li, M., Liu, C., Li, Z., Yang, M., and Wang, J. Spann: Highly-efficient billion-scale approximate nearest neighbor search. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021.

Choo, D., Grunau, C., Portmann, J., and Rozhon, V. k-means++: few more steps yield constant approximation. In *International Conference on Machine Learning*, pp. 1909–1917. PMLR, 2020.

Computer, T. RedPajama: An open source recipe to reproduce LLaMA training dataset. https://github.com/togethercomputer/RedPajama-Data, 2023. Accessed: May 10, 2025.

Contributors, K. Ktransformers: A flexible framework for experiencing cutting-edge llm inference optimizations. https://github.com/kvcache-ai/ktransformers, 2025. Accessed: 2025-05-14.

Craswell, N., Mitra, B., Yilmaz, E., Campos, D., and Voorhees, E. M. Overview of the trec 2019 deep learning track. *arXiv preprint arXiv:2003.07820*, 2020.

Craswell, N., Mitra, B., Yilmaz, E., Campos, D., and Lin, J. Ms marco: Benchmarking ranking models in the large-data regime. In *proceedings of the 44th International ACM SIGIR conference on research and development in information retrieval*, pp. 1566–1576, 2021.

Douze, M. Indexing 1t vectors. https://github.com/facebookresearch/faiss/wiki/Indexing-1T-vectors, 2020.

Douze, M., Sablayrolles, A., and Jégou, H. Link and code: Fast indexing with graphs and compact regression codes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3646–3654, 2018.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. The faiss library, 2025. URL https://arxiv.org/abs/2401.08281.

Fu, C., Xiang, C., Wang, C., and Cai, D. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, January 2019. ISSN 2150-8097. doi: 10.14778/3303753.3303754. URL https://doi.org/10.14778/3303753.3303754.

Fu, C., Wang, C., and Cai, D. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility, 2021. URL https://arxiv.org/abs/1907.06146.

Gao, J. and Long, C. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proc. ACM Manag. Data*, 1 (2), June 2023. doi: 10.1145/3589282. URL https://doi.org/10.1145/3589282.

Gao, J. and Long, C. RabitQ: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. In *Proceedings of the ACM on Management of Data (SIGMOD '24)*, volume 2, 2024.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pp. 604–613, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919629. doi: 10.1145/276698.276876. URL https://doi.org/10.1145/276698.276876.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Jégou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. doi: 10.1109/TPAMI.2010.57.

Karpukhin, V., Oguz, B., Min, S., Lewis, P. S., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pp. 6769–6781, 2020.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Devlin, J., Lee, K., Toutanova, K., Jones, L., Kelcey, M., Chang, M.-W., Dai, A. M., Uszkoreit, J., Le, Q., and Petrov, S. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lee, C., Prahlad, D., Kim, D., and Kim, H. Work-in-progress: On-device retrieval augmented generation with knowledge graphs for personalized large language models. In *2024 International Conference on Embedded Software (EMSOFT)*, pp. 1–1, 2024. doi: 10.1109/EMSOFT60242.2024.00006.

Lempitsky, V. The inverted multi-index. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pp. 3069–3076, USA, 2012. IEEE Computer Society. ISBN 9781467312264.

Li, M., Lin, Y., Zhang, Z., Cai, T., Li, X., Guo, J., Xie, E., Meng, C., Zhu, J.-Y., and Han, S. Svdqunat: Absorbing outliers by low-rank components for 4-bit diffusion models. *arXiv preprint arXiv:2411.05007*, 2024.

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., and Lin, X. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2019.

Li, Z., Zhang, X., Zhang, Y., Long, D., Xie, P., and Zhang, M. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

Lin, J., Nogueira, R., and Yates, A. *Pretrained transformers for text ranking: Bert and beyond*. Springer Nature, 2022.

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*, 42(4):824–836, 2018.

Manohar, M. D., Shen, Z., Blelloch, G., Dhulipala, L., Gu, Y., Simhadri, H. V., and Sun, Y. 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*, pp. 270–285, 2024.

Microsoft Learn. Vector index size and staying under limits, 2025. URL https://learn.microsoft.com/en-us/azure/search/vector-search-index-size?utm_source=chatgpt.com&tabs=portal-vector-quota.

Munoz, J. V., Gonçalves, M. A., Dias, Z., and Torres, R. d. S. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition*, 96:106970, 2019.

Munyampirwa, B., Lakshman, V., and Coleman, B. Down with the hierarchy: The'h'in hnsw stands for" hubs". *arXiv preprint arXiv:2412.01940*, 2024.

NVIDIA. NVIDIA A10 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/products/a10-gpu/, 2025.

ObjectBox Ltd. Edge AI: The era of on-device ai. https://objectbox.io/on-device-vector-databases-and-edge-ai/, 2024. Accessed May 2025.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Rekabsaz, N., Lesota, O., Schedl, M., Brassey, J., and Eickhoff, C. Tripclick: the log files of a large health web search engine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2507–2513, 2021.

Ren, J., Zhang, M., and Li, D. Hm-ann: efficient billion-point nearest neighbor search on heterogeneous memory. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Ren, J., Tulyakov, S., Peng, K.-C., Wang, Z., and Shi, H. Efficient neural networks: From algorithm design to practical mobile deployments. CVPR 2023 Tutorial, 2023. https://snap-research.github.io/efficient-nn-tutorial/.

Research, F. A. Guidelines to choose an index. https://github.com/facebookresearch/faiss/wiki/Guidelines-to-choose-an-index/

28074dc0ddc733f84b06fa4d99b3f6e2ef65613d#if-below-1m-vectors-ivfx, 2025. Accessed: 2025-05-10.

Ryan, M. J., Xu, D., Nivera, C., and Campos, D. EnronQA: Towards personalized RAG over private documents. *arXiv preprint arXiv:2505.00263*, 2024.

Schuhmann, C., Vencu, R., Beaumont, R., Kaczmarczyk, R., Mullis, C., Katta, A., Coombes, T., Jitsev, J., and Komatsuzaki, A. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs. *arXiv preprint arXiv:2111.02114*, 2021.

Seemakhupt, K., Liu, S., and Khan, S. Edgerag: Online-indexed rag for edge devices. *arXiv preprint arXiv:2412.21023*, 2024.

Severo, D., Ottaviano, G., Muckley, M., Ullrich, K., and Douze, M. Lossless compression of vector ids for approximate nearest neighbor search. *arXiv preprint arXiv:2501.10479*, 2025.

Shao, R., He, J., Asai, A., Shi, W., Dettmers, T., Min, S., Zettlemoyer, L., and Koh, P. W. W. Scaling retrieval-based language models with a trillion-token datastore. *Advances in Neural Information Processing Systems*, 37: 91260–91299, 2024.

Shen, M., Umar, M., Maeng, K., Suh, G. E., and Gupta, U. Towards understanding systems trade-offs in retrieval-augmented generation model inference, 2024. URL https://arxiv.org/abs/2412.11854.

Subramanya, S. J., Devvrit, Kadekodi, R., Krishaswamy, R., and Simhadri, H. V. *DiskANN: fast accurate billion-point nearest neighbor search on a single node*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Tatsuno, K., Miyashita, D., Ikeda, T., Ishiyama, K., Sumiyoshi, K., and Deguchi, J. AiSAQ: all-in-storage ANNS with product quantization for DRAM-free information retrieval. *arXiv preprint arXiv:2404.06004*, 2024. URL https://arxiv.org/abs/2404.06004.

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)*, pp. 171–185, Santa Clara, CA, February 2025. USENIX Association. ISBN 978-1-939133-45-8. URL https://www.usenix.org/conference/fast25/presentation/tian-bing.

Totino, V. Phone storage: How much do you really need?, 2025a. URL https://www.optimum.com/articles/mobile/choosing-phone-storage-amount-needs-guide.

Totino, V. Phone storage: How much do you really need?, 2025b. URL https://www.optimum.com/articles/mobile/choosing-phone-storage-amount-needs-guide. Accessed May 15, 2025.

Wang, M., Xu, W., Yi, X., Wu, S., Peng, Z., Ke, X., Gao, Y., Xu, X., Guo, R., and Xie, C. Starling: An i/o-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment. In *Proceedings of the ACM on Management of Data (SIGMOD '24)*, volume 2, 2024. doi: 10.1145/3639269.3652200.

Wang, P., Wang, C., Lin, X., Zhang, W., and He, Q. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, 2021. doi: 10.14778/3476249.3476258.

Wang, Z. J. and Chau, D. H. Mememo: On-device retrieval augmentation for private and personalized text generation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2765–2770, 2024.

Xue, Z., Song, Y., et al. Powerinfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*, 2024.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Yin, H., Chen, T., Qu, L., and Cui, B. On-device recommender systems: A comprehensive survey. *arXiv preprint arXiv:2401.11441*, 2024.

Yu, W., Liao, N., Luo, S., and Liu, J. Ragdoll: Efficient offloading-based online rag system on a single gpu. *arXiv preprint arXiv:2504.15302*, 2025.

Zamani, H., Trippas, J. R., Dalton, J., Radlinski, F., et al. Conversational information seeking. *Foundations and Trends® in Information Retrieval*, 17(3-4):244–456, 2023.

Zerhoudi, S. and Granitzer, M. Personarag: Enhancing retrieval-augmented generation systems with user-centric agents. *arXiv preprint arXiv:2407.09394*, 2024.

Zhang, M., Wang, W., and He, Y. Learning to anneal and prune proximity graphs for similarity search. In *International Conference on Learning Representations (ICLR)*, 2020. Available at https://openreview.net/forum?id=HJlXC3EtwB.

Zhang, Y., Pan, P., Zheng, Y., Zhao, K., Zhang, Y., Ren, X., and Jin, R. Visual search at alibaba. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 993–1001, 2018.

Zhu, J., Patel, L., Zaharia, M., and Popa, R. A. Compass: Encrypted semantic search with high accuracy. Cryptology ePrint Archive, Paper 2024/1255, 2024. URL https://eprint.iacr.org/2024/1255.

Zilliz AI FAQ. How much memory overhead is typically introduced by indexes like hnsw or ivf?, 2025. Accessed May 2025.

A. Appendix

A.1. Best-First Search

Algorithm 1 Best-First Search on Graph-based Index

```
1: Input: Graph $G$ with entry node $p$, query $x_q$, result size $k$, queue size $ef$ ($k \leq ef$)
2: Output: Top-$k$ approximate neighbors $R$
3: Initialize $C \leftarrow \{p\}$, $R \leftarrow \{p\}$, $V \leftarrow \{p\}$
4: while $C \neq \emptyset$ and $\min(C.\text{dist}) \leq \max(R.\text{dist})$ do
5:     $c \leftarrow$ node in $C$ with smallest distance to $x_q$
6:     Remove $c$ from $C$
7:     for each neighbor $n$ of $c$ do
8:         if $n \notin V$ then
9:             Extract Embedding $x_n$
10:            Compute $d = \text{Dist}(x_q, x_n)$
11:            Add $n$ to $V$, add $n$ to $C$ and $R$ with distance $d$
12:    if $|R| > ef$ then
13:        Keep only the $ef$ closest nodes in $R$
14: return top $k$ closest nodes in $R$
```

Graph-based indexes converge quickly to the nearest neighbors for two main reasons: (1) Graph structures connect vectors to their approximate neighbors (identified during construction). These connections typically link semantically similar vectors, creating pathways that allow the search to navigate towards relevant regions efficiently. Consequently, neighbors of a vector similar to the query are also likely to be similar. (2) The graph implicitly yields a much finer-grained partitioning of the vector space compared to IVF, enabling the search to examine significantly fewer candidates from the entire database (Gao & Long, 2023; Li et al., 2019; Malkov & Yashunin, 2018; Indyk & Motwani, 1998).

A.2. Two-Level-Search

Algorithm 2 Two-Level Search

```
1: Input: query $q$, entry point $p$, re-ranking ratio $a$, result size $k$, search queue length $ef$
2: Output: $k$ closest neighbors to $q$
3: $visited \leftarrow \{p\}$; $AQ \leftarrow \emptyset$; $EQ \leftarrow \{p\}$; $R \leftarrow \{p\}$
4: while $EQ \neq \emptyset$ do
5:     $v \leftarrow$ extract closest element from $EQ$ to $q$
6:     $f \leftarrow$ get furthest element from $R$ to $q$
7:     if $distance(v, q) > distance(f, q)$ then
8:         break
9:     for each $n \in$ neighbors($v$) do
10:        if $n \notin visited$ then
11:            $visited \leftarrow visited \cup \{n\}$
12:            Calculate approximate distance $d_{approx}(n, q)$
13:            $AQ \leftarrow AQ \cup \{n\}$
14:    $M \leftarrow$ extract top $a\%$ from $AQ$ that are not in $EQ$
15:    for each $m \in M$ do
16:        Compute exact distance $d_{exact}(m, q)$
17:        $EQ \leftarrow EQ \cup \{m\}$; $R \leftarrow R \cup \{m\}$
18:        if $|R| > ef$ then
19:            Remove furthest element from $R$ to $q$
20: return top $k$ elements from $R$
```

Regarding the generalizability of this method, our approach can readily incorporate alternative lightweight approximation techniques beyond quantization. Methods such as distillation models or link and code representations (Douze et al., 2018) can be substituted, provided they maintain computational efficiency even at the cost of some accuracy. This flexibility makes our approach adaptable to various computational constraints and application scenarios while preserving the core two-level search paradigm.

A.3. High Degree Preserving Graph Pruning

Algorithm 3 High Degree Preserving Graph Pruning

```
1: Input: Original graph $G$ with the set of vertices $V$, candidate list size $ef$, connection number threshold $M$ for high
   degree nodes and $m$ for other nodes, where $m < M$, percentage of high degree nodes $a$
2: Output: Pruned graph $G_1$
3: $\forall v \in V : D[v] \leftarrow$ degree of $v$ of $G$, $G_1 \leftarrow$ empty graph
4: $V^* \leftarrow$ nodes with the top $a\%$ highest (out) degree in $D$
5: for $v \in V$ do
6:     $W \leftarrow$ search($v$, $ef$)                              ▷ Refer to Algorithm 1
7:     if $v \in V^*$ then
8:         $M_0 \leftarrow M$
9:     else
10:        $M_0 \leftarrow m$
11:    Select $M_0$ neighbors from $W$ using the original heuristic
12:    Add bidirectional edges between $v$ and its neighbors to $G_1$
13:    Shrink edges if $\exists q \in$ neighbor and $D_{out}(q) > M$
```

Note that this algorithm does not require knowledge about the query distribution. Hence, it can scale efficiently to large datasets, providing a simple yet effective mechanism to balance index size and search performance.

A.4. Comparison of Degree Distributions Across Pruning Methods

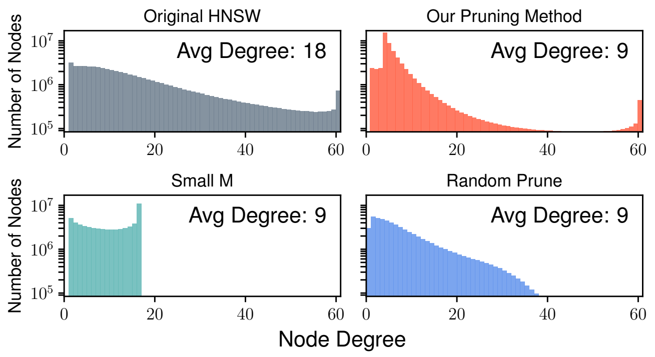


Figure 8. **[Ablation Study]:** Comparison of (out-)degree distributions between the original graph, our pruning method, and two heuristic baselines. Similar to Figure 7, the gray curve represents the original HNSW graph, which has twice the size of the others. Only our pruning method successfully preserves the high-degree nodes.

A.5. Latency and Cache Behavior under Storage Constraints

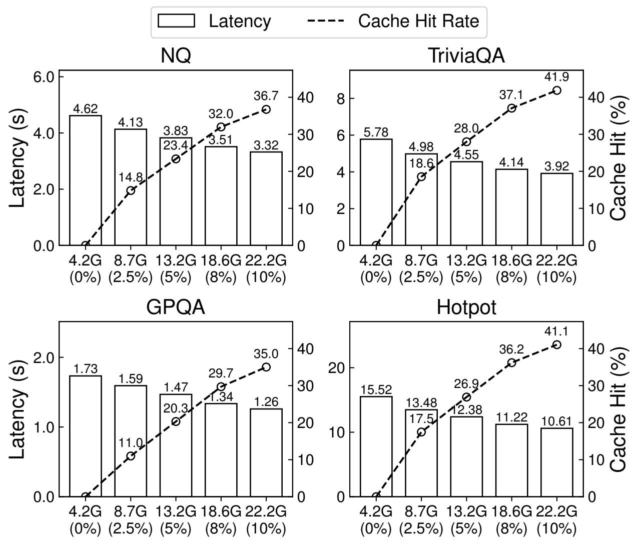


Figure 9. **[Ablation Study]:** Latency and cache hit rate comparison under varying storage constraints across four datasets. The x-axis indicates total storage size (graph size plus cached embeddings) and the corresponding percentage of cached embeddings.

A.6. Graph-based recomputation breakdown.

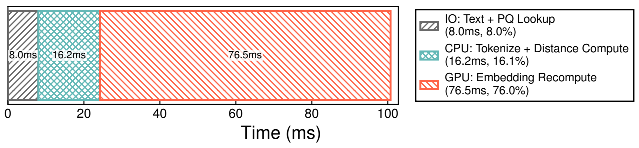
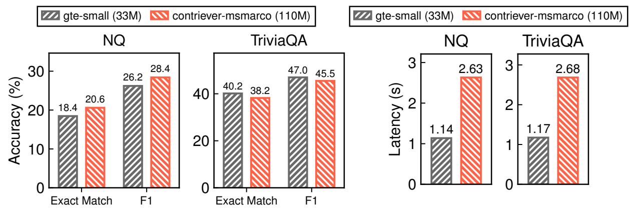


Figure 10. **[Ablation Study]:** Latency breakdown of a batch of requests in graph-based recomputation.

Figure 10 breaks down the time cost of a single batch in graph-based recomputation into three stages, categorized by the primary system resource used. Each batch aggregates multiple hops of recomputation, as described in Section 4.2. First, LEANN performs PQ lookups to select promising nodes, then retrieves and tokenizes the corresponding raw text. The tokenized inputs are sent to the embedding server. Finally, LEANN performs embedding recomputation and distance calculation. Although embedding recomputation is the primary bottleneck in LEANN, the three stages, spanning I/O, CPU, and GPU resources, can potentially be overlapped to improve overall efficiency. We leave this optimization to future work.

A.7. Using different embedding model sizes.



(a) Accuracy (b) Latency

Figure 11. **[Ablation Study]:** Latency on the A10 GPU and accuracy of a smaller embedding model evaluated on a 2M-chunk datastore, using a fixed search queue length of `ef=50`. The smaller embedding model significantly reduces latency without causing a substantial drop in downstream accuracy.

Since the primary bottleneck of our system lies in the recomputation process, as shown in Figure 10, we further explore the potential for latency reduction by adopting a smaller embedding model. Specifically, we replace the original *contriever* model (110M parameters) used in Section 6.1 with the lightweight *GTE-small* model (Li et al., 2023), which has only 34M parameters. We evaluate performance on a smaller 2M document datastore using a fixed search queue length of `ef=50`, as shown in Figure 11. The results show that *GTE-small* achieves a $2.3\times$ speedup while maintaining competitive downstream task accuracy, within 2% of the contriever baseline. This demonstrates the potential of LEANN to further reduce search latency via graph-based recomputation, while preserving the near-zero storage overhead characteristic of semantic search on edge devices.