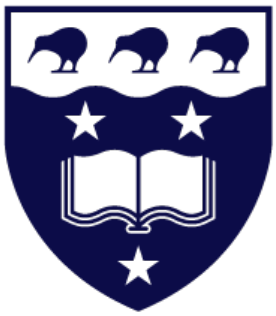
RESEARCH PROJECT REPORT

Vector Data Management in External Memory

Project Number: 120



Waipapa Taumata Rau
University of Auckland

Author: Oliver Lin
olin423@aucklanduni.ac.nz

Co-worker: James Sun
ysun755@aucklanduni.ac.nz

Supervisor: Dr. Miao Qiao

Department of Electrical, Computer, and Software Engineering
The University of Auckland

October 19, 2025

Abstract

Graph-based indexes have become the state-of-the-art for Approximate Nearest Neighbor Search (ANNS) in high-dimensional vector spaces. However, as datasets grow to billion-point scales, they often exceed main memory capacity, necessitating external memory solutions that store the index on Solid-State Drives (SSDs). This shift introduces a significant I/O bottleneck and highlights a fundamental trade-off between the offline cost of index construction and the online efficiency of queries. Pruning algorithms, which shape the graph's topology, are central to this trade-off.

This research investigates this problem by proposing and evaluating a novel pruning strategy called **Geometric Informed Pruning (GIP)**. Unlike the purely distance-based RobustPrune algorithm used in the state-of-the-art DiskANN system, GIP incorporates an additional angular constraint to preserve geometrically diverse neighbors during graph construction. The central hypothesis is that this dual-condition approach creates a more navigable graph topology, leading to more efficient search paths.

We conduct a rigorous empirical evaluation comparing GIP against the baseline RobustPrune on standard public benchmarks (Sift1M, Gist1M, Glove). Our results demonstrate that GIP consistently achieves a superior trade-off between Queries Per Second (QPS) and recall, particularly in the low-to-mid recall range. This query performance improvement is achieved without increasing the index size or graph density. However, this gain is counterbalanced by a notable increase in index construction time—up to 3x on high-dimensional data—due to the computational overhead of the angular calculations. We conclude that GIP is a valuable, albeit situational, alternative for applications where query throughput is paramount and the dataset is sufficiently static to justify the higher one-time construction cost.

STUDENT DECLARATION

I hereby declare that:

1. This report is the result of the final year project work carried out by my project partner (see cover page) and I under the guidance of our supervisor (see cover page) in the 2025 academic year at the Department of Electrical, Computer and Software Engineering, Faculty of Engineering, University of Auckland.
2. This report is not the outcome of work done previously.
3. This report is not the outcome of work done in collaboration, except that with a potential project sponsor (if any) as stated in the text.
4. This report is not the same as any report, thesis, conference article or journal paper, or any other publication or unpublished work in any format.

In the case of a continuing project, please state clearly what has been developed during the project and what was available from previous year(s):

_____ Not a continuing project _____

Signature: _____ **Date:** _____

Contents

Contents iii

Acknowledgements iv

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Aims and Objectives . . . 1
- 1.3 Scope of Research . . . 2
- 1.4 Report Structure . . . 2

2 Literature Review 3
- 2.1 Existing Methods of Indexing Vector Databases . . . 3
  - 2.1.1 Tree-Based Methods . . . 3
  - 2.1.2 Hashing-Based Methods . . . 3
  - 2.1.3 Graph-Based Methods . . . 3
- 2.2 Best Practices for Vector Indexing in External Memory . . . 4
  - 2.2.1 Core Challenges in External Memory Vector Indexing . . . 5
  - 2.2.2 DiskANN: A State-of-the-Art External Memory Index . . . 5
  - 2.2.3 Alternative External Memory Approaches . . . 5
- 2.3 Advanced Pruning Techniques in Proximity Graphs . . . 5
  - 2.3.1 Construction-Time Pruning . . . 6
  - 2.3.2 Query-Time Pruning . . . 7

3 Methodology 8
- 3.1 Experimental Environment . . . 8
- 3.2 Baseline Algorithm: DiskANN with RobustPrune . . . 8
- 3.3 Proposed Algorithm: Geometric Informed Pruning (GIP) . . . 8
- 3.4 Experimental Setup . . . 10
  - 3.4.1 Datasets . . . 10
  - 3.4.2 Index Construction and Querying Parameters . . . 11
  - 3.4.3 Evaluation Metrics . . . 11
  - 3.4.4 Experimental Procedure . . . 12

4 Results and Discussion 13
- 4.1 Query Performance: QPS vs. Recall . . . 13
- 4.2 Index Construction Time . . . 13
- 4.3 Summary of Trade-offs . . . 14

5 Conclusion 15

6 Future Work 15

Bibliography 17

A Additional Experiments 19

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Miao Qiao, for her invaluable guidance, critical feedback, and unwavering support throughout this research project. Her expertise was instrumental in shaping the direction of this work. I would also like to thank my project partner, James Sun, for the collaborative effort, insightful discussions, and shared commitment that were essential to completing this project.

Glossary of Terms

Term	Definition
ANNS	Approximate Nearest Neighbor Search.
Vamana	The specific type of proximity graph used in the DiskANN index.
Proximity Graph	A graph where nodes represent data points and edges connect "close" points.
Recall	A measure of search accuracy, representing the fraction of true nearest neighbors found.

Abbreviations

Abbreviation	Full Form
GIP	Geometric Informed Pruning
QPS	Queries Per Second
PQ	Product Quantization
SSD	Solid-State Drive
I/O	Input/Output
RAM	Random-Access Memory

1 Introduction

1.1 Background

In modern data-intensive applications, from large language models and e-commerce platforms to multimedia retrieval systems, high-dimensional vector embeddings have become a fundamental tool for representing complex data. The task of efficiently finding similar vectors—known as Approximate Nearest Neighbor Search (ANNS)—is a core and computationally demanding operation. Among the various indexing strategies developed to accelerate this task, graph-based methods have emerged as the state-of-the-art due to their superior query performance and accuracy [10]. These methods construct a proximity graph where vectors are nodes and edges connect "close" neighbors, allowing a query to be answered by traversing the graph.

However, as vector datasets grow to billions or even trillions of points, they often exceed the capacity of a machine's main memory (RAM). This necessitates the use of external memory solutions, where the index primarily resides on secondary storage like Solid-State Drives (SSDs). This shift from an in-memory to a disk-resident model introduces substantial challenges, chief among them the I/O bottleneck. Disk access is orders of magnitude slower than RAM access, making random I/O operations prohibitively expensive.

Pioneering systems like DiskANN [2] have been developed to address this, employing advanced techniques such as SSD-aware data layouts and strategic prefetching. Despite these advances, a fundamental trade-off persists between the offline cost of constructing a high-quality graph index and the online efficiency of performing queries on it. This project examines this key trade-off by focusing on pruning algorithms, which are central to shaping the graph's topology and, consequently, its search performance and navigability.

1.2 Aims and Objectives

The primary aim of this research is to conduct a rigorous investigation into the performance and theoretical underpinnings of graph-based ANNS algorithms within the external memory model. We seek to analyze, evaluate, and propose enhancements to state-of-the-art systems by focusing on the pivotal role of pruning strategies in balancing index construction cost against query efficiency. This research is guided by the following core objectives:

1. **In-depth Analysis of External Memory ANNS:** We will conduct a comprehensive analysis of DiskANN as a representative baseline for disk-resident vector search, examining its Vamana graph structure, SSD-aware layout, and the intricate interplay between its beam search algorithm and underlying I/O patterns.

2. **Investigation of Advanced Pruning Strategies:** A central objective is to explore novel pruning algorithms for graph construction. Our preliminary work

on a **Geometric Informed Pruning (GIP)** strategy suggested that combining distance-based and angular-based pruning can improve the navigability of the graph. This frames our core research challenge: *to design and evaluate a pruning technique that improves the graph topology for faster search without incurring excessive construction overhead or increasing graph density.*

3. **Rigorous Experimental Validation:** All proposed techniques will be empirically validated on large-scale public benchmarks. Performance will be measured against a comprehensive set of metrics, including query latency (QPS), recall, I/O efficiency, construction cost, and index footprint, to provide a holistic view of the trade-offs involved.

1.3 Scope of Research

To ensure the project remains focused and achievable within its timeframe, the following boundaries are defined:

- The project will exclusively focus on **graph-based ANNS algorithms** designed for or adapted to **external memory** (disk-resident) scenarios. Other indexing paradigms such as tree-based or hashing-based methods are discussed for context but not empirically evaluated.
- The primary focus will be on optimizing the **static index construction process** and its impact on subsequent query performance. Dynamic indexes, which support insertions and deletions, are considered out of scope.

1.4 Report Structure

The remainder of this report is organized as follows. **Section 2** presents a comprehensive literature review of vector indexing methods, from foundational tree-based and hashing-based approaches to modern graph-based techniques, with a special focus on external memory solutions and advanced pruning strategies. **Section 3** details the methodology for our experiments, including the algorithms, datasets, and evaluation metrics used for our comparative analysis. **Section 4** presents and discusses the results of our experiments, analyzing the performance trade-offs of the proposed method. Finally, **Section 5** and **Section 6** conclude the report, summarizing our key findings and suggesting promising directions for future work.

2 Literature Review

2.1 Existing Methods of Indexing Vector Databases

Vector database indexing methods aim to enable efficient similarity search in high-dimensional spaces, addressing the "curse of dimensionality" which renders traditional spatial indexes ineffective. These methods employ various strategies to organize vector data in ways that facilitate rapid retrieval of approximate nearest neighbors, balancing query performance against index quality, memory usage, and construction costs [1].

2.1.1 Tree-Based Methods

Tree-based indexing methods recursively partition the vector space to create hierarchical structures that enable logarithmic-time search [1, 4, 5]. Early examples like k-d trees are effective in low dimensions but suffer performance degradation as dimensionality increases. VP-Trees offer an improvement by partitioning the space based on distances from selected "vantage points," making them adaptable to general metric spaces [4].

A more modern and practical approach is Spotify's ANNOY (Approximate Nearest Neighbors Oh Yeah), which constructs a forest of random projection trees [5]. In each tree, the data is split by a random hyperplane at each node. By building multiple trees, ANNOY improves search robustness. Its design allows the index to be memory-mapped from disk, reducing RAM consumption and making it a viable option for medium-scale problems [1].

2.1.2 Hashing-Based Methods

Hashing-based methods map high-dimensional vectors to compact hash codes that preserve similarity, allowing for fast, bucket-based lookups [1]. The foundational technique in this category is Locality-Sensitive Hashing (LSH), which uses families of hash functions designed such that similar items are more likely to collide in the same bucket [6]. While theoretically grounded, LSH often requires numerous hash tables to achieve high recall, leading to significant memory overhead.

A highly influential extension is Product Quantization (PQ) [8]. PQ decomposes a high-dimensional vector space into a Cartesian product of lower-dimensional subspaces and quantizes each subspace independently. This creates a compact codebook that drastically reduces the memory footprint of vectors. This compression makes PQ particularly well-suited for external memory systems, where minimizing data transfer size is critical for mitigating I/O costs [1, 8].

2.1.3 Graph-Based Methods

Graph-based indexing methods represent vectors as nodes in a proximity graph and have become the state-of-the-art for high-dimensional similarity search due to their outstanding empirical performance [10]. The core idea is to build a graph where a greedy search

traversal can efficiently navigate from a starting point to the query's nearest neighbors. The design of these graphs is often inspired by one of three foundational concepts. The k-Nearest Neighbor Graph (KNNG) connects each node to its k closest neighbors, but its exact construction is computationally expensive. The Relative Neighborhood Graph (RNG) refines a graph by pruning an edge (u, v) if a third node w exists that is closer to both u and v than they are to each other, enhancing connectivity while limiting node degree [12, 16].

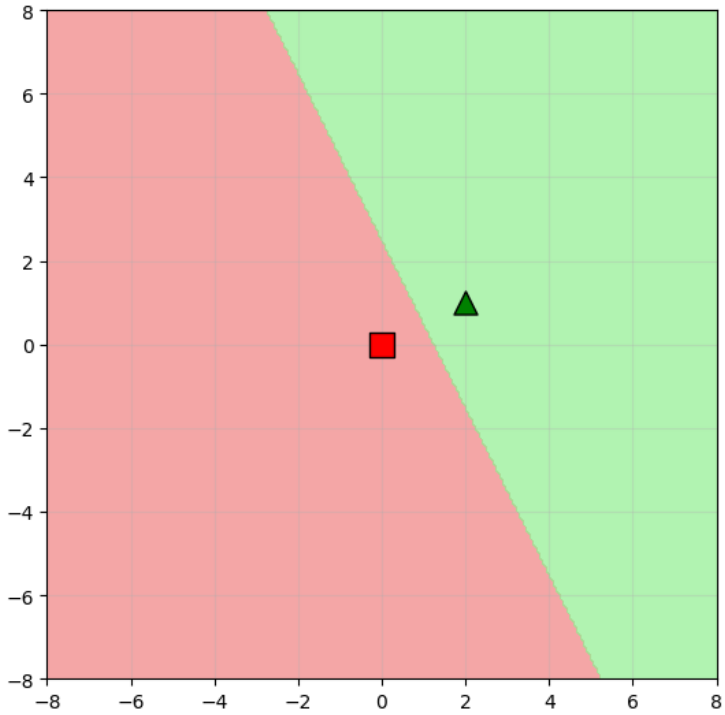


Figure 1: Illustration of the RNG-based pruning rule. An edge to node v is pruned from source node u if v falls within the lune (green shaded area) defined by existing neighbor w.

The Navigable Small World (NSWG) graph builds connections incrementally, adding both short-range and long-range links to ensure that greedy routing can efficiently discover short paths between any two nodes, mimicking the "small-world" phenomenon [3].

Modern algorithms build upon these principles. HNSW (Hierarchical Navigable Small World) organizes nodes into a hierarchical structure of increasingly sparse layers. A search starts at the sparsest top layer to quickly find the approximate region of the query, then progressively refines the search in the denser lower layers [3]. Another key refinement is the NSG (Navigable Spreading-out Graph) algorithm, which optimizes graph connectivity by using an RNG-based heuristic to ensure an even distribution of edges and enhance path diversity [9].

2.2 Best Practices for Vector Indexing in External Memory

When vector datasets grow too large for RAM, external memory solutions become essential [2]. These approaches must overcome the fundamental performance gap between

main memory and secondary storage, including limited bandwidth, high latency, and block-based access patterns [1, 2].

2.2.1 Core Challenges in External Memory Vector Indexing

The primary obstacle for disk-resident indexes is the I/O bottleneck. Secondary storage offers significantly lower bandwidth and higher latency than RAM, making each disk read a costly operation [2]. Random access patterns, which are common in graph traversal, are particularly detrimental to performance. Therefore, effective external memory algorithms must be designed to maximize sequential I/O and minimize the total number of disk accesses. Furthermore, intelligent caching and prefetching are critical for hiding latency and improving throughput [2, 11].

2.2.2 DiskANN: A State-of-the-Art External Memory Index

DiskANN is a leading graph-based index designed specifically for external memory, introducing key innovations for efficient disk-based similarity search [2, 10]. At its core, DiskANN employs a navigable proximity graph called Vamana, which is constructed using a specialized pruning mechanism known as RobustPrune to maintain a balance between connectivity and sparsity [2].

A central innovation in DiskANN is its locality-optimized node layout. Before writing the graph to disk, it reorders the nodes so that neighbors in the graph are placed in contiguous disk locations. This transforms the random access pattern of a graph traversal into a more sequential one, drastically improving I/O efficiency. This is complemented by a beam search algorithm with strategic prefetching of node neighborhoods, further reducing the impact of disk latency on search performance [2].

2.2.3 Alternative External Memory Approaches

While graph-based methods like DiskANN are prominent, other techniques have emerged. SPANN (Set of Points Approximate Nearest Neighbor) uses a partition-based approach where the dataset is divided into clusters [10]. The cluster centroids are kept in memory, forming a coarse-grained index, while the full vectors are stored on disk. A search first identifies relevant clusters using the in-memory centroids and then retrieves only the necessary points from disk, reducing I/O operations. More recently, Starling combines an in-memory navigation graph with a reordered on-disk graph to improve data locality and reduce disk access [11]. ScaDANN focuses on scalable construction, using techniques like ‘overlapping block-level insertion‘ to integrate graph building and merging into a single, more efficient operation, reducing I/O overhead during the build phase [15].

2.3 Advanced Pruning Techniques in Proximity Graphs

Pruning is an essential operation in graph-based ANNS that shapes the graph's topology to create a "navigable small-world" network. The goal is to build a sparse graph where a

greedy search can still efficiently find short paths to a query's nearest neighbors. Research has developed specialized techniques that fall into two main categories: construction-time pruning, which optimizes the graph's static topology, and query-time pruning, which makes the dynamic search traversal more efficient [12, 13].

2.3.1 Construction-Time Pruning

Construction-time pruning aims to build a graph that is sparse yet robustly connected. A foundational example is the RobustPrune algorithm used in DiskANN's Vamana graph [2]. This strategy is governed by a hyperparameter, α, which controls the trade-off between sparsity and connectivity. For a node u, its selected neighbor w, and a candidate neighbor v, the edge to v is pruned if $\alpha \cdot d(w, v) \leq d(u, v)$. This heuristic prunes a candidate v if an existing neighbor w already provides a sufficiently short path to it. A larger α creates a sparser graph, while a smaller α preserves more edges, potentially improving search quality at the cost of a larger index [2].

While effective, this distance-only approach may inadvertently discard neighbors that are geometrically diverse, even if they are slightly further away. This can lead to sub-optimal topologies where the search algorithm has fewer distinct paths to explore. Recent work by Yang et al. [12] addresses this limitation by introducing **α-pruning**, a strategy that enhances the standard RNG check with an angular constraint. An edge (u, v) is pruned by a neighbor w only if it satisfies both the distance condition and an angular condition, specifically that the angle $\angle uwv$ exceeds a threshold α [12]. Standard RNG pruning is equivalent to an α-pruning with $\alpha = 60°$; by setting $\alpha > 60°$, the pruning becomes less aggressive, preserving neighbors that are more angularly diverse. This creates a more robust graph topology that can improve search performance.

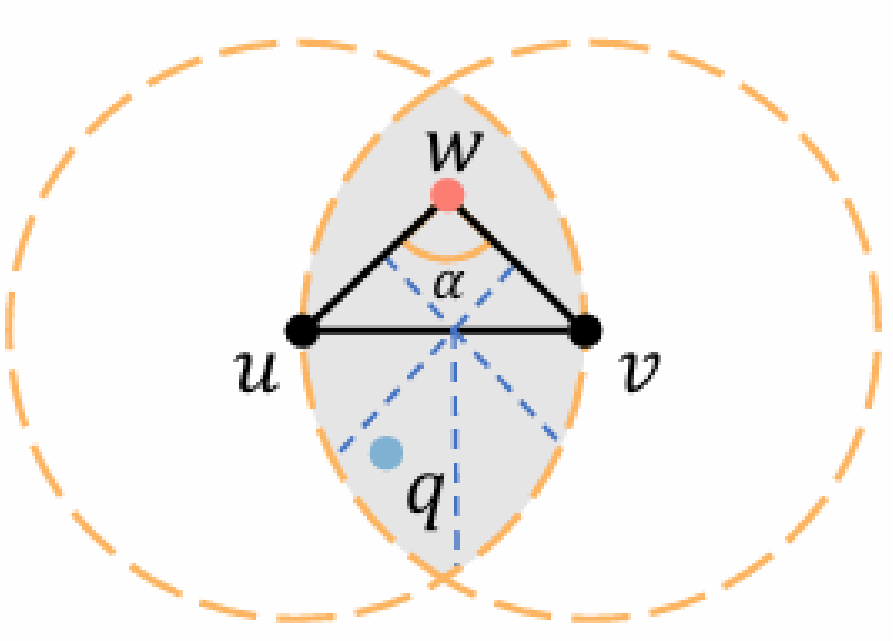


Figure 2: A conceptual diagram of the α-pruning strategy. **Source: Yang et al. [12].**

2.3.2 Query-Time Pruning

Separate from optimizing the graph's static structure, query-time pruning aims to reduce computationally expensive distance calculations during the search traversal [13]. The TRIM operation is a notable example that uses a vector's Product Quantization (PQ) representation to compute a relaxed lower bound based on the triangle inequality [13]. This allows for dynamic pruning of candidates during search without computing their full-precision distance to the query, thereby reducing computations and potential I/O costs.

The success of techniques like TRIM, which work around the limitations of a static graph, highlights a potential research gap. If a graph's construction could be made more topologically aware by integrating multiple geometric criteria—such as both distance and angle—the need for complex query-time pruning might be reduced. An inherently more navigable graph could lead to more direct search paths, which is particularly beneficial in disk-based systems where minimizing random access is paramount.

3 Methodology

This research conducts a comparative analysis of pruning strategies for graph-based ANNS in an external memory context. The methodology is designed to rigorously evaluate a novel pruning technique against a state-of-the-art baseline, isolating the impact of the pruning heuristic on both index construction cost and query performance.

3.1 Experimental Environment

All experiments were conducted on a dedicated server with the following specifications to ensure consistency and reproducibility:

- **CPU:** 2 x Intel(R) Xeon(R) Silver 4210R processors (total of 40 logical cores).
- **Memory:** 256 GB of DRAM.
- **Operating System:** CentOS 7.9.2009.

This setup provides sufficient resources to ensure that performance measurements primarily reflect the algorithms' intrinsic efficiency rather than system constraints, particularly during the computationally intensive index construction phase.

3.2 Baseline Algorithm: DiskANN with RobustPrune

The baseline for this study is the Vamana graph construction algorithm used in DiskANN, which employs a distance-based pruning strategy known as **RobustPrune**. During index construction, for each node u, a set of candidate neighbors is identified. The RobustPrune algorithm then refines this set to select the final outgoing edges for u.

The pruning decision is governed by a single geometric condition controlled by the hyperparameter $\alpha \geq 1$. For a source node u, an already selected neighbor w, and a candidate neighbor v, the algorithm prunes v if it does not offer a significantly shorter path, as defined by the inequality:

$$\alpha \cdot d(w, v) \leq d(u, v)$$

Here, $d(w, v)$ represents the Euclidean distance between vectors w and v. This mechanism effectively removes redundant edges where an existing neighbor (w) already provides a good route to another candidate point (v). This distance-only approach is highly effective but may not always preserve the most geometrically diverse set of neighbors, potentially leading to a graph where search paths are suboptimal.

3.3 Proposed Algorithm: Geometric Informed Pruning (GIP)

Building upon the baseline, this research introduces and evaluates a novel strategy called **Geometric Informed Pruning (GIP)**. The central hypothesis of GIP is that a more

robust and navigable graph topology can be achieved by considering not only distance but also the angular diversity of neighbors during pruning. A graph with well-distributed, non-collinear edges is presumed to offer more distinct and efficient pathways for a greedy search algorithm.

GIP enhances RobustPrune by incorporating a second condition: a minimum angle constraint, θ. A candidate neighbor v is pruned only if it satisfies both the distance condition from RobustPrune and this new angular condition. The combined logic is as follows: A candidate v is pruned if:

$$(\alpha \cdot d(w, v) \leq d(u, v)) \quad \text{AND} \quad (\angle wuv > \theta)$$

where $\angle wuv$ is the angle between the vectors $\vec{uw}$ and $\vec{uv}$.

The formal algorithm for the graph construction process incorporating GIP is detailed in Algorithm 1, and a visual breakdown of the pruning logic is provided in Figure 3.

Algorithm 1 GIP(p, V, α, θ, R)

Data: Graph G, point $p \in P$, candidate set V, distance threshold $\alpha \geq 1$, angle threshold θ, degree bound R
Result: G is modified by setting at most R new out-neighbors for p
begin
$V \leftarrow (V \cup N_{\text{out}}(p)) \setminus \{p\}$;
$N_{\text{out}}(p) \leftarrow \emptyset$;
while $V \neq \emptyset$ **do**
    $p' \leftarrow \arg\min_{p' \in V} d(p, p')$;
    $N_{\text{out}}(p) \leftarrow N_{\text{out}}(p) \cup \{p'\}$;
    if $|N_{\text{out}}(p)| = R$ **then**
        break;
    end if
    for $p^* \in V$ **do**
        if $\alpha \cdot d(p', p^*) \leq d(p, p^*)$ **and** $\angle p'pp^* > \theta$ **then**
            remove p^* from V;
        end if
    end for
end while

Proof of Pruning Relaxation

This dual-condition mechanism is a formal **relaxation** of the original RobustPrune algorithm. Let C_D be DiskANN's distance condition and C_A be GIP's angular condition. DiskANN prunes a candidate node v if C_D is true. GIP prunes v only if the composite condition $(C_D \wedge C_A)$ is true. By the properties of logical conjunction, for $(C_D \wedge C_A)$ to be true, C_D must necessarily be true. Therefore, the set of nodes pruned by GIP is a subset of those pruned by DiskANN ($P_{GIP} \subseteq P_{DiskANN}$). This proves that GIP is a less aggressive pruning strategy, as it deliberately preserves candidates that DiskANN

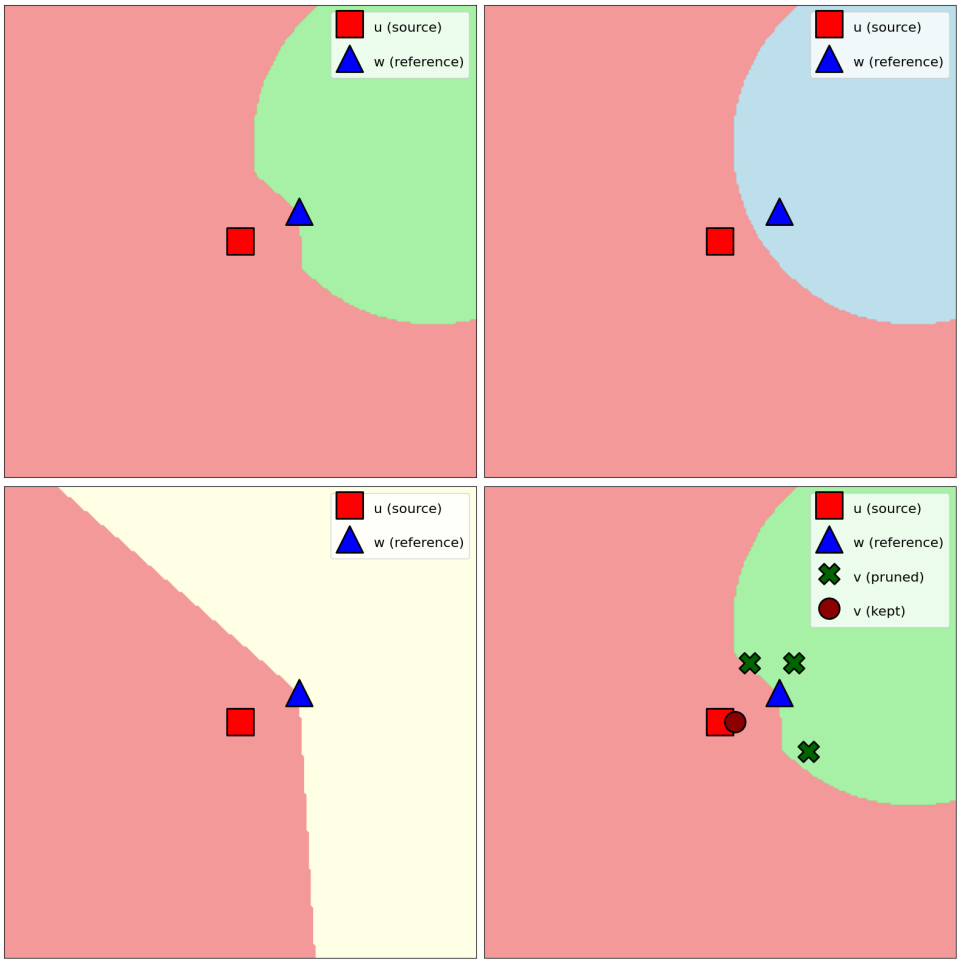


Figure 3: Visualization of the GIP algorithm. The final pruning zone (green) is the intersection of the distance-based condition (blue) and the angle-based condition (yellow), leading to a more selective pruning process that preserves angularly diverse neighbors.

would discard if they offer significant angular diversity (i.e., if C_D is true but C_A is false). The intended outcome is a graph with an improved topology at the same density, as the maximum degree is fixed.

3.4 Experimental Setup

3.4.1 Datasets

The experiments were performed on three widely-used public benchmarks: Sift1M, Gist1M [17], and Glove [18]. These were chosen to evaluate the algorithms across different data modalities (image vs. text embeddings) and dimensionalities. Table 1 summarizes their properties.

Table 1: Datasets used for experimental evaluation.

Dataset	Size	#Queries	Dim.	Type
Sift1M	1,000,000	10,000	128	Image
Gist1M	1,000,000	1,000	960	Image
Glove	1,183,514	10,000	100	Text

3.4.2 Index Construction and Querying Parameters

The DiskANN `apps/build_disk_index` program was modified to incorporate GIP and used for index construction. Any parameters not explicitly mentioned retain their default values.

- `-R (--max_degree):` Fixed at 64. Controls the maximum out-degree for each node.
- `--build_PQ_bytes:` Set to 64. Specifies the bytes for Product Quantization.
- `-B (--search_DRAM_budget):` Constrained to 1 GB. RAM allocated for search.
- `-M (--build_DRAM_budget):` Set to 16 GB. RAM budget for construction.

Subsequently, the standard DiskANN `apps/search_disk_index` program was used for querying.

- `-L (--Lsearch):` Varied across a range of values (10, 20, 30, 40, 50, 100). This parameter controls the search list size (beam size). A larger L generally leads to higher recall at the cost of increased latency and I/O.
- `--num_nodes_to_cache:` Fixed at 10000. Nodes cached in memory during search.
- `-W (--beamwidth):` Fixed at 2. The beam width used during greedy search.

3.4.3 Evaluation Metrics

The performance of each algorithm was evaluated based on a combination of efficiency and effectiveness metrics.

- **Index Building Time:** Measured in seconds, this is the total wall-clock time required to construct the index. This metric represents the one-time offline cost.
- **Queries Per Second (QPS):** This metric measures query throughput. A higher QPS is better.
- **Recall@10:** This metric measures search accuracy. It is the fraction of queries for which the true top-10 nearest neighbors intersect with the top 10 results returned by the algorithm.

3.4.4 Experimental Procedure

The evaluation process involved the following steps for each of the three datasets:

1. **Index Construction:** Two indexes were built: one using the baseline Robust-Prune and another using the proposed GIP algorithm, with the parameters specified above. The total wall-clock time for each build was recorded.

2. **Querying:** The `apps/search_disk_index` program was used to run the corresponding query set against each constructed index. This step was repeated for different values of the search list size parameter `-L` to generate a performance curve.

3. **Data Analysis:** For each run, performance metrics (QPS, latency, I/Os) were recorded. Recall@10 was calculated by comparing the results against the provided ground truth files. The results were then plotted to visualize the trade-off between QPS and Recall.

4 Results and Discussion

The experimental evaluation was designed to compare the proposed Geometric Informed Pruning (GIP) strategy against the baseline RobustPrune algorithm used in DiskANN. The analysis focuses on the trade-offs between online query performance (QPS vs. Recall) and the offline index construction cost. To ensure a fair comparison, the maximum degree of all indexes was fixed at 64. The final average degree of all graphs was also 64, meaning the GIP and RobustPrune graphs had the same number of edges. This isolates the impact of edge *quality* on search performance, rather than graph density.

4.1 Query Performance: QPS vs. Recall

The primary measure of search performance is the trade-off between query throughput (QPS) and search accuracy (Recall). Figure 4 illustrates this relationship for both algorithms across the Sift1M, Gist1M, and Glove datasets.

On both the **Sift1M** and **Glove** datasets, GIP consistently demonstrates a superior QPS-to-Recall trade-off. For any given level of recall below approximately 0.98, the GIP-constructed index delivers a higher QPS than the baseline. This indicates that GIP's topology is more efficient for greedy search. By preserving angularly diverse neighbors, GIP creates a graph with more effective navigation paths, allowing the search algorithm to locate nearest neighbors with fewer I/O operations and distance computations. However, this performance advantage diminishes at very high recall levels (¿0.98), where the search becomes more exhaustive and the subtle topological advantages are less impactful.

The results on the **Gist1M** dataset show a markedly different trend. While GIP still offers a marginal benefit at very low recall, its performance advantage tapers off much more quickly. This is a direct consequence of the "curse of dimensionality," as Gist1M has a significantly higher dimensionality (960) than Sift1M (128) or Glove (100). In such high-dimensional spaces, vector distances tend to concentrate (i.e., the distances between most pairs of points become almost indistinguishable), and vectors become nearly orthogonal. This diminishes the meaningfulness of both distance and angular separation, reducing the effectiveness of GIP's angle-based pruning condition and making it difficult for any greedy search to perform efficiently.

4.2 Index Construction Time

While GIP provides a clear benefit in query performance on lower-dimensional data, this comes at the cost of increased index construction time. Figure 5 shows the build times for both algorithms across the three datasets.

The additional angular calculations required by GIP introduce a significant computational overhead. The construction time for the GIP index was approximately **1.3x slower** on Sift1M and **1.1x slower** on Glove. This overhead was most pronounced on the high-dimensional Gist1M dataset, where the GIP index took nearly **3x longer** to



(a) Sift1M (k=10) (b) Gist1M (k=10) (c) Glove (k=10)

Figure 4: Comparative query performance (QPS) vs. Recall@10 for GIP and RobustPrune.

build than the baseline. This is expected, as the computational complexity of calculating the angle between two vectors is linear with respect to their dimensionality. The higher the dimension, the more computationally expensive each pruning decision in GIP becomes.

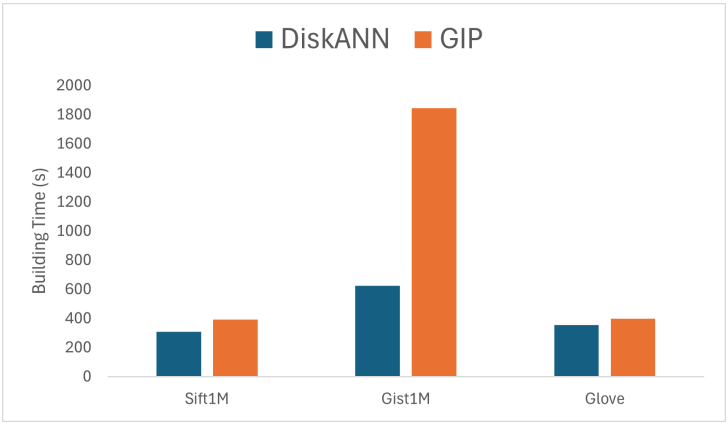


Figure 5: Index construction times for GIP and RobustPrune across datasets.

4.3 Summary of Trade-offs

The results highlight a fundamental and clear trade-off. GIP leverages a more complex, geometrically-aware heuristic to build a higher-quality graph, which in turn supports faster queries. This investment is paid during the offline index construction phase. For applications involving relatively static datasets where query latency and throughput are critical, the higher upfront cost of GIP is likely justified by the subsequent gains in search performance. Conversely, for dynamic environments requiring frequent index rebuilds or for extremely high-dimensional data where GIP's benefits are marginal, the faster construction of the standard RobustPrune remains the preferable approach.

5 Conclusion

This research set out to investigate and enhance pruning strategies for graph-based ANNS algorithms in an external memory context. By introducing and evaluating the Geometric Informed Pruning (GIP) method, which uniquely combines distance-based and angular-based criteria, we have demonstrated that it is possible to construct a graph with a superior topology for similarity search without increasing its size or density.

Our primary finding is that GIP's more sophisticated pruning heuristic leads to a more efficient graph structure for querying, especially on datasets of low to moderate dimensionality. The GIP-enabled index demonstrated a notable improvement in Queries Per Second (QPS) for any given recall target compared to the baseline. This confirms our central hypothesis that by preserving geometrically diverse neighbors, GIP enables more efficient and direct graph traversal during search.

However, this query performance gain is counterbalanced by a significant increase in the offline index construction cost, which is exacerbated by high dimensionality. This highlights the fundamental trade-off between index quality and construction efficiency that was a central theme of this investigation. The choice between GIP and a simpler heuristic like RobustPrune is therefore application-dependent.

In conclusion, the GIP strategy presents a valuable, albeit situational, alternative to existing pruning methods. It is most suitable for production systems where query throughput is the primary concern, a moderate level of recall is acceptable, and the underlying vector dataset is relatively static, making the one-time index construction cost a worthwhile investment for long-term query performance gains.

6 Future Work

Building upon the findings of this project, several promising avenues for future research have been identified to further advance the field of external memory ANNS.

- **Optimization of GIP Construction:** The primary drawback of GIP is its high construction cost. Future work should focus on optimizing this phase. This could involve developing more efficient methods for calculating or approximating angular relationships, perhaps using vector quantization techniques to estimate angles from compressed codes, or applying the dual-condition pruning strategy more selectively (e.g., only for nodes in denser regions of the graph) to reduce the number of expensive computations.
- **Scalability to Billion-Scale Datasets:** While this study provided valuable insights on million-scale benchmarks, validating these findings on billion-scale datasets is an essential next step. Such a study would confirm whether the observed trade-offs in query performance and construction cost hold at the massive scale for which external memory systems are primarily designed.

- **Adaptation for Dynamic Datasets:** The current GIP implementation is designed for static indexes. A significant research challenge is to adapt the principles of GIP for dynamic environments where vectors are frequently inserted, updated, or deleted. This would require developing novel algorithms that can maintain the graph's geometric diversity incrementally without requiring costly full rebuilds.
- **Automated Hyperparameter Tuning:** This work relied on fixed, empirically chosen values for the GIP hyperparameters α and θ. Future research could explore methods for automatically tuning these parameters based on the statistical properties of the dataset (e.g., its intrinsic dimensionality or cluster structure). This could lead to a more robust and adaptive algorithm that performs optimally across a wider range of data distributions.

Bibliography

[1] Zhang, Y., Ma, Z., Chen, Y., Lin, J., & Zhang, W. (2024). Survey of vector database management system. *The VLDB Journal, 33*, 1591–1615.

[2] Subramanya, S. J., Devvrit, F., Simhadri, H. V., Krishnaswamy, R., & Kadekodi, R. (2019). DiskANN: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems, 32*.

[3] Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*.

[4] Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 311–321.

[5] Spotify. (2013). *ANNOY: Approximate Nearest Neighbors Oh Yeah*. Retrieved from https://github.com/spotify/annoy

[6] Andoni, A., & Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM, 51*(1), 117–122.

[7] Weiss, Y., Torralba, A., & Fergus, R. (2008). Spectral hashing. *Advances in Neural Information Processing Systems (NeurIPS)*, 1753–1760.

[8] Jégou, H., Douze, M., & Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 33*(1), 117–128.

[9] Fu, C., Xiang, C., Wang, C., & Cai, D. (2019). Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment, 12*(5), 461–474.

[10] Chen, Q., Zhao, B., Wang, H., Li, M., Liu, C., Li, Z., Yang, M., & Wang, J. (2021). SPANN: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems, 34*, 5199–5212.

[11] Wang, M., Xu, W., Yi, X., Wu, S., Peng, Z., Ke, X., Gao, Y., Xu, X., Guo, R., & Xie, C. (2024). Starling: An I/O-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment. *Proceedings of the ACM on Management of Data, 2*(1), Article 14, 1–27.

[12] Yang, S., Xie, J., Liu, Y., Yu, J. X., Gao, X., Wang, Q., Peng, Y., & Cui, J. (2025). Revisiting the Index Construction of Proximity Graph-Based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2410.01231*.

[13] Song, Y., Zhang, P., Gao, C., Yao, B., Wang,K., Wu, Z., & Qu, L. (2025). TRIM: Accelerating High-Dimensional Vector Similarity Search with Enhanced Triangle-Inequality-Based Pruning. *arXiv preprint arXiv:2508.17828*.

[14] Wang, J., Wang, J., Zeng, G., Tu, Z., Gan, R., & Li, S. (2012). Scalable k-NN graph construction for visual descriptors. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 1106-1113.

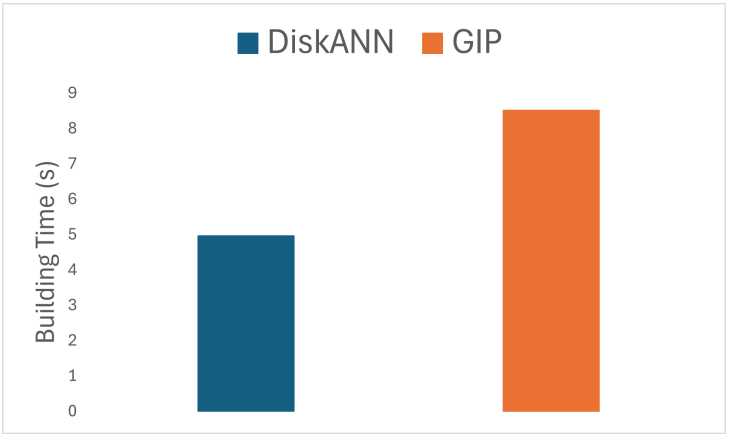
[15] Gu, H., & Kim, M. (2025). ScaDANN: A Scalable Disk-Based Graph Indexing Method for ANN. *Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference*.

[16] Toussaint, G. T. (1980). The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, *12*(4), 261–268.

[17] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Datasets for approximate nearest neighbor search. *http://corpus-texmex.irisa.fr/*, 2010.

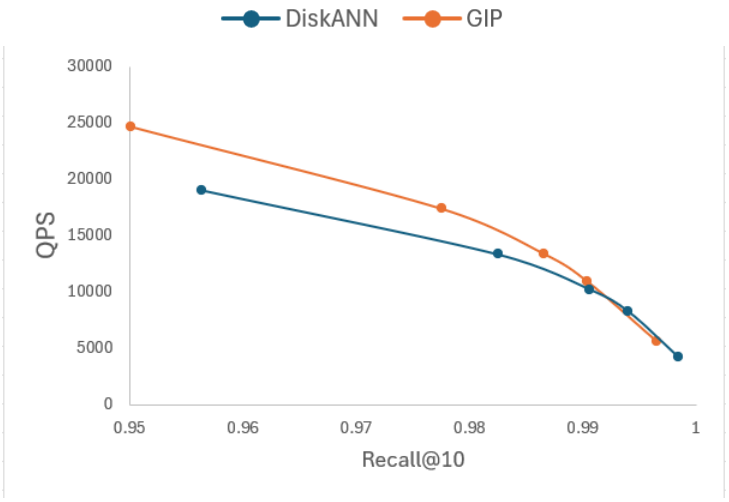
[18] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

A Additional Experiments



Sift100K

Figure 6: Build time for Sift100K dataset.



Sift100K

Figure 7: Performance time for Sift100K dataset.

These additional experiments served as the preliminary basis for verifying the validity of the methodology before scaling up to million-point datasets.