# Overcoming the Sync-Compute Dilemma in Parallel Graph-Based Vector Retrieval

Qiji Mo[1*], Zhiyuan Hua[1*], Zebin Yao[1], Lixiao Cui[1,†], Gang Wang[1], Xiaoguang Liu[1,†]
Zijing Wei[2], Xinyu Liu[2], Tianxiao Tang[2], Shaozhi Liu[2], Lin Qu[2]
[1] College of Computer Science, Nankai University, Tianjin, China
[2] Alibaba Group Holding Limited, Beijing, China
{moqj, huazy, yaozb, cuilx, wgzwp, liuxg}@nbjl.nankai.edu.cn
{feishi.wzj, lxy264173, tianxiao.ttx, shaozhi.liu}@alibaba-inc.com, xide.ql@taobao.com

***Abstract*—Approximate Nearest Neighbor Search (ANNS) is fundamental to modern applications such as Retrieval-augmented Generation. Among various ANNS algorithms, graph-based methods have become the state-of-the-art due to their excellent search efficiency. The core of graph-based methods is to perform a Best-First Search (BFiS) on the proximity graph. However, with the continuous growth in data scale and dimensionality, the single-threaded BFiS algorithm is becoming increasingly inadequate to meet performance demands.**

**In this paper, we first deconstruct the existing parallel graph-based ANNS algorithms (e.g., iQAN, Edge-wise). We find that they are constrained by the Bulk Synchronous Parallel model, which leads to a dilemma between reducing the synchronization overhead and reducing the computational overhead. To overcome this dilemma, we propose a decoupled search paradigm named ScatterSearch. In this paradigm, each thread independently conducts a full BFiS from a distinct entry point and maintains a private candidate queue, thereby eliminating explicit synchronization. Building on ScatterSearch, we further propose a leader-guided search pruning strategy to reduce computational overhead. It prunes search progress of some threads to avoid long-tail computation and redirects the pruned threads through a work-stealing mechanism. Comprehensive experiments on real-world datasets show that with 16 threads and at Recall@100=0.99, our method reduces latency by up to 41.46% and increases throughput by up to 70.82% compared to iQAN. Our method has been utilized on the Alibaba's taobao platform.**

***Index Terms*—approximate nearest neighbor search, vector search, intra-query parallelism.**

## I. Introduction

Nearest neighbor search from high-dimensional vector datasets has become a fundamental component of modern applications such as large-scale recommendation systems [1]–[3], vector databases [4]–[9], Retrieval-augmented Generation (RAG) [10]–[20] and Large Language Models (LLMs) inference [21], [22]. However, performing an exact search for nearest neighbors is often computationally prohibitive due to the *curse of dimensionality* [23]. To overcome this challenge, Approximate Nearest Neighbor search (ANNS) has emerged as an indispensable solution, trading a tiny loss of accuracy for a significant reduction in search latency. Over the years, a rich landscape of ANNS algorithms has been developed, which can be broadly classified into four families: hashing-based [24]–[30], partition-based [31]–[42], quantization-based [43]–[51] and graph-based methods [52]–[68]. Among these diverse approaches, graph-based methods have consistently demonstrated superior performance [69], [70]. The core of graph-based methods is to perform a Best-First Search (BFiS) on the proximity graph.

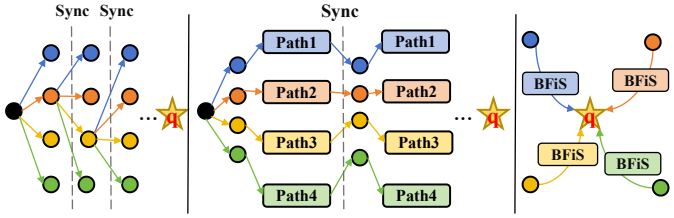



Fig. 1: The paradigms of Edge-wise parallelism, Path-wise parallelism and *ScatterSearch*. Each color represents a thread.

Although graph-based methods have achieved significant success in ANNS, ANNS systems face escalating performance challenges. This is because modern applications impose stringent, often millisecond-level latency constraints to ensure a responsive user experience. As the scale and dimensionality of data increase [71], [72], the single-threaded BFiS is increasingly unable to meet the requirement [73]. A lot of systems use inter-query parallelism to improve system throughput, but this does not help in reducing single-query latency.

To bridge this performance gap, intra-query parallelism [73], [74] has emerged as a critical optimization technique for single-machine multi-core systems. Intra-query parallelism leverages multiple threads to execute a single query collaboratively. Modern intra-query parallel graph-based ANNS methods target the parallelization of specific steps within the BFiS algorithm. Edge-wise Parallelism (Figure 1a) decomposes the neighbor expansion step of a **single vertex** into concurrent sub-tasks. At each step, each thread is assigned a subset of neighbors and calculates the distance between the vectors within its subset and the query vector. Path-wise Parallelism (e.g., iQAN [73]) selects **a batch of candidate vertices** from the global candidate queue instead of strictly extracting the single closest one. Then these vertices are assigned among

* The two authors contribute equally to this work.
† Corresponding authors.

threads. Each thread is responsible for exploring a path based on its assigned vertices, as shown in Figure 1b. After a period of exploration, all threads merge their local results into the global candidate queue. Repeat this periodic exploration until there are no unexpanded vertices in the global candidate queue.

In this paper, we first deconstruct existing parallel graph-based ANNS algorithms (e.g., iQAN, Edge-wise). We find that they all can be formulated within the Bulk Synchronous Parallel (BSP) model [75]. We then analyze the performance impact of two critical data structures in BFiS: the candidate queue and the visited list. Our analysis reveals that (1) existing algorithms face a dilemma between reducing the synchronization overhead of the global candidate queue and reducing the computational overhead; (2) maintaining a lock-free, shared visited list is crucial for eliminating duplicate computations with minimal overhead.

Based on our systematic analysis, our core idea is to design a new parallel search paradigm that overcomes the current dilemma, **reducing synchronization overhead and computational overhead simultaneously**.

We first propose *ScatterSearch*, a fundamentally decoupled search paradigm, which **eliminates explicit synchronization overhead**. Figure 1c illustrates the paradigm of *ScatterSearch*. *ScatterSearch* abandons the search paradigm guided by the global candidate queue. Instead, *ScatterSearch* allows each thread independently conducts a full BFiS from a randomly selected distinct entry point, maintaining private candidate queue and result set. During search, all threads share a lock-free visited list to effectively prevent duplicate vertex accesses.

Based on our *ScatterSearch* paradigm, we further propose a *leader-guided search pruning* strategy to **reduce the computational overhead**. Through extensive empirical analysis, we identify three intrinsic characteristics of parallel graph-based ANNS. (1) **Highly skewed thread contribution**: A small fraction of threads discover the majority of the final KNN; (2) **Contribution-Efficiency synergy**: Threads that contribute more to recall are also more efficient, finding more KNN results with fewer distance calculations; (3) **Early advantage dominance**: A thread's final recall contribution is largely determined in the early stage of the search. Leveraging these characteristics, our idea is to identify a **leader** thread at the early stage of the search and prune the search of other threads. We classify the non-leader threads as **follower** and **lagging** threads. Guided by the search progress of leader thread, we apply varying degrees of search pruning to other threads and redirect pruned threads using a work-stealing mechanism.

In summary, our main contributions are as follows:

- We present a systematic deconstruction of parallel graph-based ANNS. We find existing algorithms all face a dilemma between reducing the synchronization overhead and reducing the computational overhead.
- We design a decoupled search paradigm, named *ScatterSearch*, which eliminates the synchronization overhead. Building upon this, we further propose a *leader-guided search pruning* strategy, which prunes the search based on thread progress and redirects the pruned threads through a work-stealing mechanism, thereby reducing the computational overhead.
- We conduct comprehensive experiments on real-world datasets. The results show that our method outperforms the state-of-the-art method iQAN. With 16 threads and at Recall@100=0.99, our method reduces latency by up to 41.46% and increases throughput by up to 70.82% compared to iQAN. Our method has been utilized on the Alibaba's taobao platform.

## II. Preliminaries

In this section, we first formally define the problem of ANNS. We then introduce the graph-based algorithms and detail the Best-First Search (BFiS) algorithm, which is the foundational search algorithm for graph-based ANNS.

### A. Approximate Nearest Neighbor Search (ANNS)

Given a dataset $X = \{x_1, x_2, ..., x_n\}$ of $n$ points in a $d$-dimensional space $\mathbb{R}^d$, and a query point $q \in \mathbb{R}^d$, the goal of an **exact K-Nearest Neighbor (KNN) search** is to find a subset $X_K \subseteq X$ with $|X_K| = K$ such that:

$$\max_{x \in X_K} \delta(q, x) \leq \min_{x \in X \setminus X_K} \delta(q, x) \tag{1}$$

Where $\delta(x, q)$ denote the distance (e.g., Euclidean distance) between point $x$ and query $q$. However, performing an exact KNN search is often computationally prohibitive on large-scale datasets. To address this, Approximate Nearest Neighbor Search (ANNS) has been proposed to trade a small amount of accuracy for significant speed efficiency. ANNS aims to find a subset $X'_K \subseteq X$ with $|X'_K| = K$ such that:

$$\max_{x \in X'_K} \delta(q, x) \leq \alpha \cdot \min_{x \in X \setminus X'_K} \delta(q, x) \tag{2}$$

where $\alpha \geq 1$ is an approximation factor that quantifies the trade-off between accuracy and search efficiency. By allowing a deviation from the exact $K$ nearest neighbors, ANNS achieves significantly faster search speed.

**Recall@$K$:** ANNS algorithm quality is commonly evaluated by Recall@$K$, which measures the fraction of true nearest neighbors successfully identified by approximate search. Given a query $q$, let $X_K$ be the set of the actual $K$ nearest neighbors, and $X'_K$ be the set of $K$ neighbors returned by the ANNS algorithm. The Recall@$K$ is formally defined as Eq.(3):

$$\text{Recall@}K = \frac{|X'_K \cap X_K|}{K} \tag{3}$$

**Latency, NDC:** ANNS algorithms efficiency is usually measured by latency, defined as the time elapsed from issuing a query to receiving the results. Generally, the primary factor affecting latency is the Number of Distance Calculations (NDC) [37], [73], [76].

**max-t NDC, $dist_1$, $dist_k$**: For intra-query parallel ANNS, the query latency is determined by the last thread to finish its search. Therefore, the key factor influencing latency is the maximum NDC per thread, calculated by $\max_{i \in \{1,...,N_t\}}\{\text{NDC}_i\}$ (we abbreviate it as max-t NDC),

**Algorithm 1** Best-First Search (BFiS)

**Input:** Graph $G$, query $q$, entry point $ep$, number of neighbors to return $K$, search list size $L$
**Output:** Set of $K$ nearest neighbors $R$

1: Initialize $C \leftarrow \{ep\}$, $R \leftarrow \{ep\}$, $V_{visited} \leftarrow \{ep\}$
2: **while** $C \neq \emptyset$ **do**
3: $\quad c \leftarrow \arg\min_{c' \in C} dist(c', q)$
4: $\quad C \leftarrow C \setminus \{c\}$
5: $\quad dist_{max} \leftarrow \max_{x \in R} dist(x, q)$
6: $\quad$ **if** $dist(c, q) > dist_{max}$ **then** ▷ Converge Condition
7: $\quad\quad$ **break**
8: $\quad$ **for all** neighbor $n$ of $c$ in $G$ **do**
9: $\quad\quad$ **if** $n \notin V_{visited}$ **then**
10: $\quad\quad\quad V_{visited} \leftarrow V_{visited} \cup \{n\}$
11: $\quad\quad\quad$ **if** $|R| < K$ or $dist(n, q) < dist_{max}$ **then**
12: $\quad\quad\quad\quad C \leftarrow C \cup \{n\}$
13: $\quad\quad\quad\quad R \leftarrow R \cup \{n\}$
14: $\quad\quad\quad\quad$ **while** $|R| > L$ **do**
15: $\quad\quad\quad\quad\quad x_{worst} \leftarrow \arg\max_{x' \in R} dist(x', q)$
16: $\quad\quad\quad\quad\quad R \leftarrow R \setminus \{x_{worst}\}$
17: $\quad\quad\quad\quad dist_{max} \leftarrow \max_{x \in R} dist(x, q)$
18: $R \leftarrow$ select Top-$K$ elements from $R$
19: **return** $R$

where $N_t$ is the number of threads. Thread search progress can be measured by the distance from its first and $K$-th nearest local result to the query, denoted as $dist_1$ and $dist_k$.

### B. Graph-Based ANNS

In graph-based ANNS algorithms, the base dataset $X$ is modeled as a proximity graph $G = (V, E)$, where the vertices $V$ are the data points and edges $E$ connect points that are close to each other in the vector space. The search process is thus transformed into a graph search process, starting from one entry point and navigating the graph to find $K$ vertices that are closest to the query $q$.

**Best-First Search (BFiS)**: This is the foundational search algorithm for graph-based ANNS. As detailed in Algorithm 1, the BFiS algorithm navigates the graph by maintaining a candidate queue $C$ and a result set $R$. The size of the result set $R$ is bounded by $L$ to tradeoff search speed and quality. The search process is initiated from an entry point $ep$ (line 1). In each iteration, the algorithm extracts the candidate vertex $c$ from $C$ that is closest to the query $q$ (lines 3-4). The algorithm then explores the neighbors of $c$ on the graph. If a neighbor $n$ has not been previously visited and is closer to the query $q$ than the farthest vertex currently in the result set $R$, it is added to both the candidate queue $C$ and the result set $R$ (lines 9-16). The algorithm terminates when the best candidate $c$ is already farther from the query than any vertex in the result set $R$ (lines 6-7) or the candidate queue $C$ becomes empty.

**KNN region**: We define the KNN region as the minimal induced subgraph that connects the true K-nearest neighbors within the index structure.

### C. Parallel Execution Models

**Bulk Synchronous Parallel (BSP) Model [75]**: The BSP model organizes parallel processing into a sequence of "supersteps." Each superstep consists of three ordered phases:(1) concurrent local computation by multiple threads, (2) global communication to exchange data, and (3) a barrier synchronization until all threads finish the current superstep. In the context of intra-query parallel ANNS, existing algorithms typically follow this model to synchronize the global candidate queue. Our analysis reveals that the barrier synchronization in BSP creates a "Sync-Compute Dilemma." Our proposed ScatterSearch aims to break the constraints of this model. The detailed analysis is shown in Section III.

**Work-Stealing Mechanism**: Work-stealing mechanism is designed to achieve load balancing in parallel computing. It allow threads that have completed their assigned tasks attempt to "steal" work from threads that are still busy. In Section IV-B, we adopt this mechanism to redirect computational resources from pruned threads to promising search paths.

## III. Deconstructing Parallel Graph-based ANNS: Bottleneck Analysis

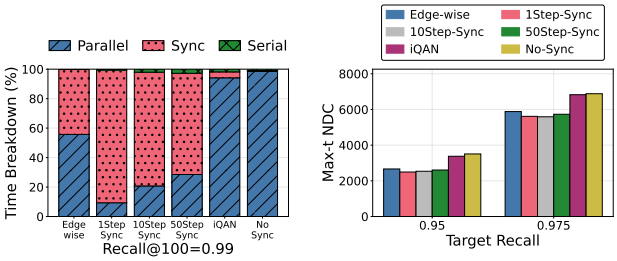
In this section, we first review existing parallel graph-based ANNS algorithms and identify two critical data structures that require careful management: the **global candidate queue** and the **visited list**. Then we analyze how the synchronization strategies for these two data structures impact on the performance of existing algorithms.

### A. Review of Existing Algorithms

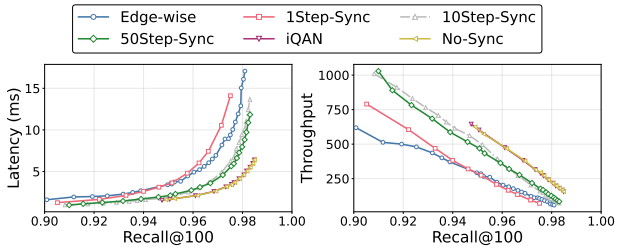
Existing algorithms primarily address the parallel graph-based ANNS task by partitioning BFiS algorithm across threads, and can be categorized into two main approaches based on how this partitioning is achieved:

- *Edge-wise parallelism* targets the neighbor exploration step. At each step, Edge-wise parallelism dequeues **the most promising vertex** from the global candidate queue, then partitions its neighbors evenly among worker threads for parallel distance calculation, as shown in Figure 1a.
- *Path-wise parallelism*, represented by iQAN [73], expands multiple vertices from the global candidate queue at once. At each step, **a batch of promising vertices** are dequeued from the global candidate queue and assigned among threads. Each thread is responsible for exploring a path based on its assigned vertices, as shown in Figure 1b.

Both Edge-wise and Path-wise parallelism can be formulated within the BSP model. The distinction lies in the task granularity of each superstep. For all these algorithms, two critical data structures have a decisive impact on performance: the global candidate queue and the visited list. The former steers the overall search direction, while the latter prevents duplicate vertex expansion across threads. Carefully managing their synchronization strategies is essential for high-performance parallel search.




(a) Execution time breakdown and Max-t NDC of different global candidate queue synchronization strategies.




(b) End-to-end latency comparison of different global candidate queue synchronization strategies.

Fig. 2: Analysis of synchronization strategy for global candidate queue in parallel graph-based ANNS.

### *B. Analysis of Synchronization Strategy for Global Candidate Queue*

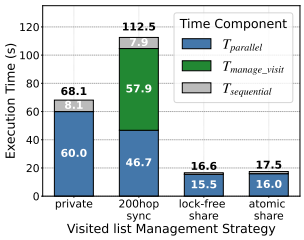
As described above, all existing algorithms can be formulated within the BSP model. After each superstep is finished, they need a synchronization for the global candidate queue before the next superstep can commence. In this section, we reveal the bottleneck of existing algorithms by analyzing the synchronization strategy of the global candidate queue.

For Edge-wise parallelism, its algorithm requires the global candidate queue to synchronize every step. For Path-wise parallelism, there are three synchronization strategies:

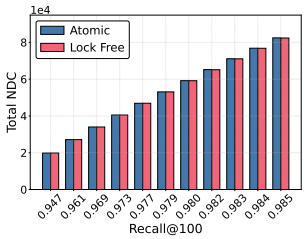
- *Fixed-period synchronization:* Synchronization occurs at regular intervals after a predefined number of search hops.
- *Dynamic synchronization:* iQAN determines synchronization timing through a heuristic approach that monitors the candidate queue update positions. It triggers synchronization when update positions fall below a predefined threshold.
- *No synchronization (We proposed) :* Add the neighbors of the entry point to the global candidate queue, and then assign them directly into worker threads for search. Each thread maintains its private candidates during search without any synchronization.

We conduct an in-depth analysis of these strategies on the LAION10M dataset. As shown in the left side of Figure 2a , we first perform an execution time breakdown for these synchronization strategies, dividing the time into three parts, which are parallel search time, synchronization time, serial time. We can see that the dynamic synchronization strategy of iQAN can lower the synchronization overhead than Fixed-period strategies and Edge-wise parallelism. This means that iQAN performs a low-frequency synchronization. We then analyze the performance of max-t NDC (defined in Section II), as shown in the right side of Figure 2a. We can observe that more frequent synchronization can lead to a lower max-t NDC. This illustrates an inherent trade-off: strategies that synchronize more frequently achieve a lower max-t NDC but incur heavier synchronization overhead, whereas less frequent synchronization reduces this cost at the expense of a higher max-t NDC.

This leads to a critical insight: **within the BSP model, minimizing synchronization overhead and reducing the NDC are conflicting objectives.** Existing algorithms are fundamentally constrained by this trade-off.




(a) The impact of different synchronization strategies for visited list on search performance.




(b) Total NDC of lock-free access and atomic protection strategy under same recalls.

Fig. 3: Analysis of visited list Management Strategies in Parallel ANNS.

When we compare the end-to-end query latency, as shown in Figure 2b, the end-to-end query latency of the Edge-wise and the Fixed-period synchronization strategies is significantly worse than No-Sync strategy, owing to their heavy synchronization overhead. We can also observe that, compared to the No-Sync strategy, dynamic synchronization of iQAN can slightly reduce the max-t NDC. But its synchronization overhead results in a final end-to-end latency comparable to the No-Sync strategy.

This reveals that the end-to-end latency of parallel graph-based ANNS is determined by both synchronization time and max-t NDC. Therefore, optimizing them simultaneously is necessary to achieve a better performance.

### *C. Analysis of Management Strategy for the Visited List*

In this section, we analyze how to manage the visited list in parallel graph-based ANNS. The purpose of the visited list is to prevent duplicate vertex visits during search. Existing algorithms generally classify the management strategies for the visited list into three categories:

- fully private: Each thread maintains its own visited list. There is no interaction between threads.
- periodically synchronized: Each thread maintains its own visited list and merge at fixed intervals.
- fully shared: All threads access a global visited list.

The visited list is an unordered structure (typically implemented as a bitset). We explore two implementations for the

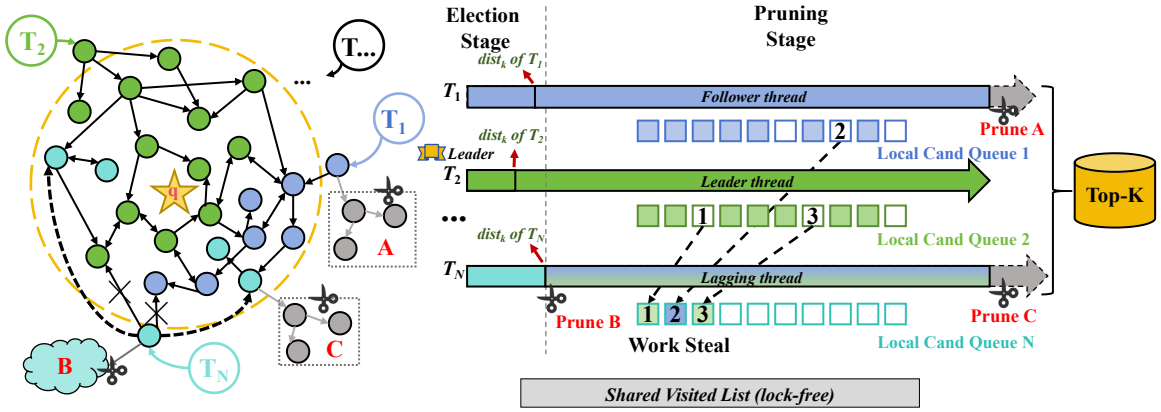



Fig. 4: The overview of our method. The left side of the figure shows N threads searching for a single query. The yellow dashed line represents the KNN region (defined in Section II) of the query. The right side of the figure shows the search progress of three thread, where $\{T_1, T_2, T_N\}$ represent the {follower, leader, lagging} thread. We use the $dist_k$ (defined in Section II) of each thread to elect $T_2$ as the leader thread. The long tail computation (area A) of $T_1$ is pruned, the useless computation (area B) and long tail computation (area C) of $T_N$ are pruned, and $T_N$ steals work from $T_1$ and $T_2$.

fully shared strategy: (1) a **lock-free** version that accesses the global visited list without concurrency control, permitting potential simultaneous access by multiple threads; and (2) an **atomic protection** version that utilizes atomic operations to guarantee that no vertex is visited more than once. To understand the impact of different access patterns, we conduct an execution time breakdown analysis across 10,000 query at the same recall rate (Recall@100=0.95) on the LAION10M [77] dataset. Figure 3a shows that the fully private strategy requires 59.98 seconds of parallel execution time. Each thread uses a private visited list, resulting in substantial duplicate vertex explorations and computational waste. For periodically synchronized strategy, we set the synchronization period to 200 hops, which is a very loose synchronization. It can reduce parallel execution time to 46.73 seconds but incurs substantial visited list management overhead (57.95 seconds). In contrast, fully shared strategy demonstrate superior efficiency. We observed that the lock-free version achieves only 15.46 seconds of parallel execution time, while the atomic protection version shows slightly lower performance (15.97 seconds). We further observe that the NDC of lock-free version and atomic protection version are nearly identical, as shown in Figure 3b. For instance, when Recall@100 is 0.947, the average NDC is 19853.5 for lock-free and 19829.7 for atomic protection. It means that the lock-free version has very few duplicate accesses, resulting in better performance by eliminating the atomic operation overhead.

Based on the above analysis, **a fully shared visited list with lock-free access** is a prerequisite for high-performance parallel search. It eliminates vast duplicate NDC while achieving zero access control overhead.

### D. Summary

We summarize our analysis as follows:

1) Existing parallel graph-based ANNS algorithms are formulated within the BSP model, requiring synchronization after each superstep.
2) Within the BSP model, existing algorithms are fundamentally constrained by the trade-off between minimizing synchronization overhead and reducing NDC.
3) A lock-free, fully shared visited list eliminates vast duplicate NDC with negligible synchronization costs.

This inspires us to use the lock-free, shared visited list in parallel graph-based ANNS. More importantly, this motivates us to design a new parallel search paradigm that overcomes the current dilemma of reducing synchronization overhead versus lowering computational overhead.

TABLE I: Characteristics of Parallel Search Algorithms

| Method | EP | Granularity | Queue | Coordination |
|---|---|---|---|---|
| Edge-wise | Single | Single Vertex | Shared | Fixed Sync. |
| iQAN | Single | Vertex Batch | Shared | Adaptive Sync. |
| ScatterSearch | Multiple | Full BFiS | Private | Prune, Work-Steal |

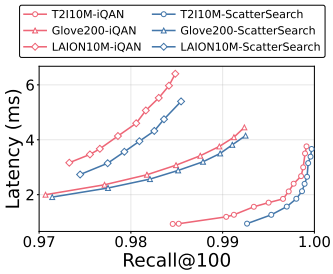
## IV. Design

We formulate the end-to-end query latency in parallel graph-based ANNS as Eq. (4), where $N_t$ is the number of threads:

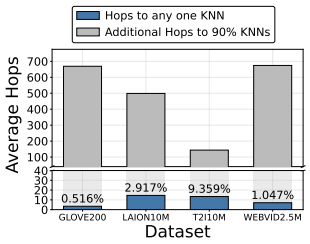
$$T_{\text{end\_to\_end}} = \max_{i \in \{1,\ldots,N_t\}} \{T_{i,\text{compute}}\} + T_{sync} \tag{4}$$

$\max_{i \in \{1,\ldots,N_t\}} \{T_{i,\text{compute}}\}$ represents the computation time of the thread with the longest computation time, $T_{sync}$ represents the time spent on synchronization. Therefore, the design goal of a parallel graph-based ANNS algorithm is **lower synchronization overhead** and **lower max-t NDC** (defined in Section II).

In this section, we first propose *ScatterSearch*, a decoupled intra-query parallel ANNS paradigm. *ScatterSearch* lets each thread manages its own local candidate queue from its own search entry point. During search, all threads share the visited list in a lock-free manner. So the $T_{sync}$ is eliminated by *ScatterSearch*. Based on *ScatterSearch*, we further propose

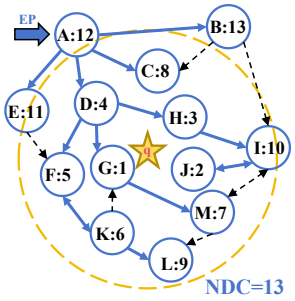



(a) The recall-latency curves of *ScatterSearch* and iQAN on three datasets.

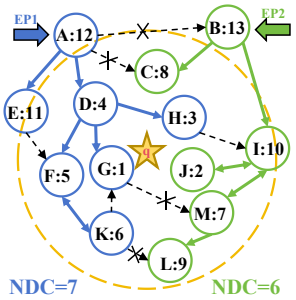



(b) The minimal cost of search before entering the KNN region.

Fig. 5: The efficiency of *ScatterSearch*.




(a) The NDC of single-thread search is 13.




(b) The NDC of two-thread *ScatterSearch* are 7 and 6, respectively.

Fig. 6: A toy example of ANNS. *ScatterSearch* lowers the per-thread NDC. The number in the vertex represents the distance to the query. Black edges are the original graph; colored edges are thread-specific paths; crossed edges $(u, v)$ mean $v$ has been visited by another thread.

a *leader-guided search pruning* strategy to reduce the max-t NDC, thereby reducing the $\max_{i\in\{1,\ldots,N_t\}}\{T_{i,\text{compute}}\}$ directly. We divide threads into three categories based on their search progress: "**leader**", "**follower**", and "**lagging**". Then, we adopt different pruning and work-stealing strategies on each thread based on its category. To highlight the fundamental structural differences between *ScatterSearch* and existing methods, we summarize the comparisons in Table I. The overview of our method is shown in Figure 4.

### A. ScatterSearch

As illustrated in Section III, reducing the synchronization overhead can effectively lower query latency. We first propose *ScatterSearch* (Figure 1c), a fundamentally decoupled parallel paradigm to eliminate the synchronization overhead. *ScatterSearch* abandons the conventional search coordinated by the global candidate queue. Instead, each thread operates as an independent BFiS agent, executing the full search process (Algorithm 1) from a distinct entry point. Each thread maintains its own private candidate queue and result set. According to our analysis of the visited list in Section III, we let all threads share a lock-free visited list to effectively prevent duplicate vertex accesses during search.

We evaluate *ScatterSearch* on Glove200 (1M scale) [78], LAION10M [77], and T2I10M [79] datasets with 8 threads. As shown in Figure 5a, the decoupled and multi-source search paradigm already enables *ScatterSearch* to achieve better search efficiency than iQAN.

We next discuss two questions: (1) Since each thread independently executes BFiS without synchronization, why can *ScatterSearch* achieve speedup compared to serial search? (2) Does the multi-source search in *ScatterSearch* lead to excessive redundancy in early exploration?

To address these two questions, following the previous work [80], we divide the search into two stages: (1) the phase before reaching the KNN region (i.e., locating any one KNN), and (2) the phase of refining KNN results within the region (e.g., achieving 90% recall of the Top-100 neighbors). Experiments across four datasets (Figure 5b) show that the first stage accounts for less than 10% of total visits. Most of the search time is spent refining results within the KNN region.

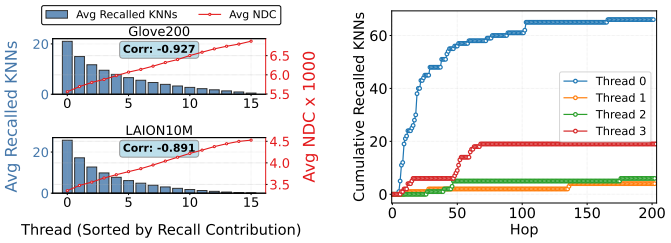
Based on the above experimental results, we first address the first question. In parallel graph-based ANNS, multiple threads may repeatedly visit the same vertex. **By using a lock-free shared visited list, this situation can be effectively avoided.** Therefore, the process of refining KNN results within the KNN region can be well-partitioned across multiple threads. Figure 6 is a toy example that shows the single-thread search (left) can be well partitioned by two-thread *ScatterSearch* (right). As shown, the single-thread search incurs a NDC of 13, whereas the NDC of the two-thread *ScatterSearch* are 7 and 6, respectively.

The result in Figure 5b also directly addresses the second question, showing that the first phase typically accounts for less than 10% of the total time, indicating that the overhead of initial exploration is negligible. Moreover, modern graph-based ANN indices [52], [54], [67] often provide dozens of outgoing edges for each vertex, which enables multiple paths to reach the KNN region for a given query [80]. So the multi-source search of *ScatterSearch* can increases the probability of discovering high-quality search paths.

### B. Leader-Guided Search Pruning

In this section, we introduce how to reduce $\max_{i\in\{1,\ldots,N_t\}}\{T_{i,\text{compute}}\}$ through our *leader-guided search pruning* strategy.

We conduct an in-depth, thread-level analysis of parallel graph-based ANNS, as illustrated in Figure 7a. It reveals two key intrinsic characteristics: (1) **Highly skewed thread contribution:** Thread contribution exhibits a Pareto-like imbalance, where a small subset of threads discovers a large fraction of the final KNN results. Across experiments on the Glove200 and LAION10M datasets, the top **3** of **16** threads contribute about **50%** of the final KNNs. (2) **Contribution-Efficiency synergy:** Threads that contribute more to recall are also more efficient, finding more KNN while using less computation. Specifically, we identify a strong negative correlation between the recall contribution of a thread and the corresponding NDC.




(a) Highly skewed thread contribution & Contribution-Efficiency synergy.

(b) Cumulative recalled KNNs by each thread. The Early Advantage Dominance.

Fig. 7: Three intrinsic characteristics in parallel ANNS.

The Pearson correlation coefficient is **-0.927** on Glove200 and **-0.891** on LAION10M. The smaller value indicate higher correlation.

Intuitively, if a few "successful" threads cover the KNN region, it will to some extent cause some other "failed" threads to hover outside the KNN region. These "failed" threads not only contribute little to the final result, but their NDCs are likely become the primary source of the max-t NDC. If we can identify the "successful" threads, then we can prune other threads to avoid excessive ineffective distance calculations. We further find the third intrinsic characteristics. **Early Advantage Dominance:** The final recall contribution of a thread is largely determined in the early stage of the search. As shown in the Figure 7b, taking a query as an example, we record the cumulative number of KNN recalls of 4 threads as the search hop changed. Result shows that threads which locate the KNN in the early stage of the search overwhelmingly dominate the final recall contribution. We further validate this early-stage impact across multiple datasets. Specifically, we measure the Pearson correlation coefficient between each thread's early and final recall contributions, with the results presented in Table II. Smaller value indicate higher correlation.

TABLE II: Pearson correlation coefficient between each thread's early (e.g., at 100th hop) and final recall contributions.

| | Dataset Name | | | |
|---|---|---|---|---|
| | **Glove200** | **LAION10M** | **Webvid2.5M** | **Mainsearch11M** |
| Corr | -0.7716 | -0.8517 | -0.9381 | -0.7850 |

Based on these three intrinsic characteristics observed above, we propose a *leader-guided search pruning* strategy. **The core idea is to elect a leader thread in the early stage of search and apply different levels of pruning to the search of other threads.** We divide the search into an election stage and a pruning stage. In the election stage, all threads start their search. A **leader thread** is elected by comparing the relative search progress of each thread, and the non-leader threads are subsequently divided into **follower threads** and **lagging threads**. Then, during the pruning stage, utilize the progress of the leader thread to prune and redirect the work of other threads. What we want to emphasize is that the entire process does not require any explicit synchronization. The details of the algorithm are as follow:

**Algorithm 2** The election stage of *ScatterSearch*

**Input:** Graph $G$, query $q$, number of neighbors to return $K$, number of threads $N_t$, election hops $H_{\text{elec}}$, shared visited list $V_{visited}$, atomic shared threshold $\epsilon$, number of threads that finish election $Cnt_{\text{elec}}$

**Output:** Status: $type_i \in \{\text{leader}, \text{follower}, \text{lagging}\}$, intermediate result $R_i$

1: $i \leftarrow$ get_thread_id()
2: $ep_i \leftarrow$ random entry point from $G$
3: $C_i \leftarrow \{ep_i\}$, $R_i \leftarrow \{ep_i\}$
4: **for** $hop \leftarrow 1$ to $H_{\text{elec}}$ **do**
5: $\quad p \leftarrow \text{pop\_best}(C_i)$
6: $\quad$ Expand neighbors of $p$, update $C_i$, $R_i$ and $V_{visited}$
7: $dist_k \leftarrow R_i[K-1].distance$
8: **if** $dist_k < \epsilon$ **then**
9: $\quad \epsilon \leftarrow dist_k$, $ID_{\text{leader}} \leftarrow i$
10: $Cnt_{\text{elec}} \leftarrow Cnt_{\text{elec}} + 1$
11: Continue Search until $Cnt_{\text{elec}} == N_t$
12: $dist_1 \leftarrow R_i[0].distance$
13: **if** $dist_1 > \epsilon$ **then**
14: $\quad type_i = \text{lagging}$
15: **else if** $i == ID_{\text{leader}}$ **then**
16: $\quad type_i = \text{leader}$
17: **else**
18: $\quad type_i = \text{follower}$
19: **return** $R_i, type_i$

*1) Election Stage:* As shown in Algorithm 2, in the election stage, all threads begin from their respective entry points and perform a fixed-step exploration (lines 1-6). After the fixed-step exploration, each thread records its $dist_k$ (defined in Section II) (line 7). They asynchronously elect the thread with the minimum $dist_k$ as the **leader thread**, and the $dist_k$ of the leader thread is stored in a global atomic variable $\epsilon$ (lines 8-9). The threads then continue searching forward without stopping to wait for others (lines 10-11). After all threads complete the election, we compare the $dist_1$ (also defined in Section II) of each thread against $\epsilon$ to classify the non-leader threads into two categories. If a thread's $dist_1 \leq \epsilon$, it means the search of this thread is able to keep pace with the leader thread, and it is marked as **follower thread**. Conversely, if a thread's $dist_1 > \epsilon$, it implies its search progress is far behind the leader thread, and its contribution to the final KNN results is limited; these threads are marked as **lagging thread** (lines 12-18). For example, as shown in Figure 4, $T_2$ is marked as the leader due to its optimal search progress. The search of $T_1$ is able to keep pace with the leader thread and is marked as a follower. In contrast, $T_N$'s search progress is far behind the leader thread, and is marked as lagging.

*2) Pruning Stage:* As shown in Algorithm 3, in the pruning stage, we apply distinct management strategies to the leader, follower, and lagging threads, respectively. The **leader thread**

performs standard BFiS on its local candidate queue $C_i$, expanding the nearest unvisited vertex until the search naturally converges (lines 1–4). For the rest two types of threads, we set the following management strategies:

**For follower threads, we adopt long tail pruning strategy.** These threads continue searching and listen for global termination signals (lines 7-9). When the leader thread completes the search, it sets $Flag_{\text{stop}} = \text{true}$, signaling that the high-quality neighborhood of the query has been fully explored. At this point, further exploration by other threads is unlikely to yield significant improvements. And this would incur substantial ineffective distance calculations. Therefore, follower threads will prune their search to avoid long tail computations. For example, in the Figure 4, follower thread $T_1$ travels to the search ineffective area **A** after the leader thread $T_2$ search ends, and this long tail search will be pruned.

**For lagging threads, we adopt early pruning and work-stealing mechanism.** These threads are far behind other threads. When they try to enter the KNN region, other threads have already fully explored the KNN region. This will cause lagging threads to hover outside the KNN region, resulting in a waste of computing resources. Therefore, we directly prune the search of lagging threads. Instead of remaining idle, lagging threads randomly select frontier vertices from the candidate queues of the leader thread and follower threads as high-quality starting vertices. Then lagging threads will continue searching for this query from these high-quality starting vertices (lines 14). This work-stealing mechanism effectively redirects computing resources from lagging areas and refocuses them on "hotspots" that have been validated by other threads. By doing so, we can fully utilize computing resources without affecting end-to-end query latency. On the basis of implementing the above strategies, lagging threads are also constrained by global termination signals to avoid long tail computations (line 15). For example, as shown in the Figure 4, Lagging thread $T_N$'s search is is far behind the leader and follower threads. It fails to enter the KNN region and travels to search useless area **B**. So its search will be early pruned. Then $T_N$ randomly select frontier vertices from $T_1$ and $T_2$ and reinitialize its search. After leader thread ($T_2$)'s search converges, its long tail search in area **C** is pruned.

The work-stealing mechanism is also adopted by some follower threads. If a follower thread finishes searching before the leader thread, it also restarts its search from high-quality vertices, sampled from candidate queues of the leader and other active follower threads. The thread continues search until the global termination signal is received (lines 10-12).

Finally, after the all threads terminates, the Top-K results from all threads are merged into the final KNN result set.

### C. Discussion

A natural question arises: if some threads contribute little to final recall, why not simply use fewer threads from the beginning? The answer is that due to the high complexity of proximity graph search, it is difficult to predict a priori which threads will successfully reach the KNN region. Reducing the number of starting threads will lower the number of effective threads. As shown in Figure 7b, with only 4 threads, nearly half exhibit minimal contribution. Consequently, decreasing thread count forces each remaining thread to perform significantly deeper searches, which means an increase in end-to-end latency. For instance, on the LAION10M dataset, achieving a Recall@100=0.98 requires average search hops of 205.68, 405.13 and 804.86 for 16, 8 and 4 threads, respectively.

**Algorithm 3** The pruning stage of *ScatterSearch*

**Input:** Thread id $i$, private candidate queue $C_i$, result set $R_i$, shared visited list $V_{visited}$, shared $Flag_{\text{stop}}$, candidate queue $\{C_j\}$ of leader and follower threads, status: $type_i \in \{\text{leader}, \text{follower}, \text{lagging}\}$
**Output:** Updated $R_i$

1: **if** $type_i ==$ leader **then**
2:  **while** $C_i \neq \emptyset$ **do**
3:  $p \leftarrow \text{pop\_min}(C_i, q)$
4:  Expand $p$, update $R_i$, $C_i$, and $V_{visited}$
5:  $Flag_{\text{stop}} \leftarrow \text{true}$ ▷ Trigger global termination
6: **else if** $type_i ==$ follower **then**
7:  **while** $C_i \neq \emptyset$ **and** $Flag_{\text{stop}} ==$ false **do**
8:  $p \leftarrow \text{pop\_min}(C_i, q)$
9:  Expand $p$, update $R_i$, $C_i$
10:  **if** $Flag_{\text{stop}} ==$ false **then**
11:  $C_i \leftarrow C_i \cup \text{sample}(\bigcup_{j:\, \text{type}_j \neq \text{lagging}} C_j)$
12:  Search until $Flag_{\text{stop}}$ set
13: **else** ▷ $type_i =$ lagging
14:  $C_i \leftarrow C_i \cup \text{sample}(\bigcup_{j:\, \text{type}_j \neq \text{lagging}} C_j)$
15:  Search until $Flag_{\text{stop}}$ set
16: **return** $R_i$

### D. Implementation Details

In ANNS, a conventional BFiS algorithm requires that the search list size $L$, be greater than or equal to the number of nearest neighbors to return, $K$, to ensure correct convergence of the algorithm. However, in parallel ANNS, it may be necessary to set $L$ to a lower value to achieve lower latency (e.g., $L$=80, $K$=100). In such case, directly maintaining the size of the $R$ equal to $L$ is infeasible, as this would prevent some true KNNs from being added to $R$ after visited by this thread, making them also inaccessible to other threads and thus never retrievable. Therefore, we make a minor adjustment on Algorithm 1. When $L < K$, we fix the size of $R$ to be $K$ instead of $L$. We then modify the search convergence condition from $dist(c, q) > dist_{max}$ to $dist(c, q) > dist_L$, where $dist_L = L\text{-th\_largest}_{x \in R}\{dist(x, q)\}$. Taking $N_t$=8, $L$=80, $K$=100 as an example, Recall@100 can increase from 0.946 to 0.955 on MainSearch11M dataset with our optimization.

In our implementation, we utilize the **bitset** to implement the visited list. To mitigate the overhead of memory allocation for the bitset at query time, we maintain a **visited list pool** initialized before the query service starts. This pool contains a set of pre-allocated bitsets. When a query arrives, an idle

bitset is fetched from the pool. Once the query is completed, the bitset is simply reset and returned to the pool for reuse.

## V. Experiments

### A. Experimental Setup

**Datasets.** We use 7 datasets with sizes ranging from 1 to 100 million vectors. Notably, the MainSearch11M dataset [81] is derived from real-world product search scenarios on Alibaba's Taobao platform, featuring a realistic data distribution. The detailed statistics of these datasets are shown in Table III, where $|X|$ represents the number of base vectors and $|Q|$ represents the number of query vectors.

TABLE III: Statistics of the datasets.

| Dataset | $\|X\|$ | $\|Q\|$ | Dim | Metric |
|---|---|---|---|---|
| Glove200 [78] | 1M | 10K | 200 | Cosine |
| MainSearch11M [81] | 11.2M | 50K | 256 | Inner Product |
| WebVid2.5M [82] | 2.5M | 10K | 512 | Cosine |
| LAION10M [77] | 10M | 10K | 512 | Cosine |
| Text-to-Image10M [79] | 10M | 10K | 200 | Inner Product |
| DEEP100M [83] | 100M | 10K | 96 | Inner Product |
| SIFT100M [84] | 100M | 10K | 128 | L2 |

**Algorithms and Parameter Setting.** Our experiments are primarily conducted on NSG [54]. To demonstrate the generality of our method, we also conduct experiments on HNSW [52]. For NSG construction, we set the parameter configuration to $L$=2000, $R$=64, and $C$=2000 across all datasets. For HNSW construction, we set the parameter configuration to $M$=32, $efc$=1000. These configurations yield high-quality graphs. It is worth noting that our method is generic and can be applied to any graph-based ANNS index. Our baseline methods include the (1) serial search algorithm, (2) Edge-wise parallel algorithm, and (3) iQAN, which is the current state-of-the-art algorithm. For iQAN[1], we conduct a grid search for parameter tuning and adopt the configuration that yields the best performance. In comparison to the serial BFiS algorithm, our method introduces one additional parameter: the election hop $H_{\text{elec}}$. We set it to 50. We control the trade-off between search latency and recall for all algorithms by varying the search list size $L$ during search. A larger $L$ leads to higher recall at the cost of increased query latency.

**Metrics.** Following the previous works [52], [54], [73], [80], we use the average Recall@$K$ metric to evaluate the accuracy of search results. To measure search efficiency, we report latency, throughput (query-per-second), and max-t NDC. The first two are end-to-end performance metrics, while max-t NDC is a machine-independent metric that reflects the main factor affecting query latency. These metrics are also defined in Section II. All comparisons are conducted at the same recall level, achieved by adjusting the search list size $L$. Under this comparison protocol, higher throughput, lower latency, and smaller max-t NDC indicate better algorithmic performance.

**Environment.** Our method targets intra-query parallelism within a single-machine, shared-memory environment. All experiments are conducted on a server equipped with two Intel(R) Xeon(R) Gold 5220 CPUs (2.20GHz), providing a total of 36 physical cores (72 threads with hyper-threading). The server is configured with 256 GB of DDR4 memory (@2666MT/s). We use CentOS 7.9 with linux kernel version 5.5. All code is compiled using g++ 13.2.0. The AVX2 instruction is used to accelerate vector distance calculation for all methods. Our code is available at https://github.com/mqqqqj/ScatterSearch.

[1]https://github.com/johnpzh/iQAN_AE

### B. Overall Performance

We conduct experiments with 8 and 16 threads to compare the end-to-end query latency of serial search, Edge-wise parallelism, iQAN, and our method, as illustrated in Figure 8 . The results show that our algorithm exhibits significant advantages. Under 8 threads, when Recall@100 is 0.99, the latency can be reduced by 21.23%-48.84% compared to iQAN. Under 16 threads, when Recall@100 is 0.98, our method achieves latency reduction by 9.86%-43.99% compared to iQAN. When Recall@100 is 0.99, the latency is reduced by 14.22%-41.46%. The primary reason of our performance advantage is the search pruning strategy, which effectively reduces the max-t NDC and therefore lowers the latency. Additionally, our *ScatterSearch* paradigm avoids synchronization overhead, which also contributes to the latency reduction. Compared to serial search and the Edge-wise parallel method, our algorithm demonstrates a more significant speedup effect. In comparison with serial search, when Recall@100 is 0.98, our method decreases query latency by 65.01%-86.22% with 8 threads, and by 66.01%-89.41% with 16 threads. We observe that the end-to-end latency of the Edge-wise parallelism is similar to serial search in most cases. This is because its extremely heavy synchronization overhead offsets the benefits of parallel search. Notably, our method achieves the most significant speedup on the Mainsearch11M dataset. This is because Mainsearch11M is sourced from the real search scenarios on Alibaba's Taobao platform with realistic, complex data distribution. The complex nature of this dataset exacerbates the imbalance of searches between different threads, and our pruning strategy can effectively prune the extra computation.

To demonstrate the generality of our method, we compare iQAN and our method on the HNSW index [52]. As shown in Table IV, our method consistently outperforms iQAN.

TABLE IV: Performance comparison using HNSW with 8 threads at Recall@100=0.99.

| Dataset | iQAN (ms) | Ours (ms) | Reduction (%) |
|---|---|---|---|
| Glove200 | 3.99 | 3.13 | 21.46 |
| MainSearch11M | 0.94 | 0.72 | 23.41 |
| LAION10M | 5.69 | 4.85 | 14.74 |
| WebVid2.5M | 7.88 | 6.15 | 21.95 |

### C. Scalability

In this section, we evaluate the scalability of our method and iQAN. We measure the speedup on Glove200 and T2I10M datasets using 2 to 32 threads. The experimental results are

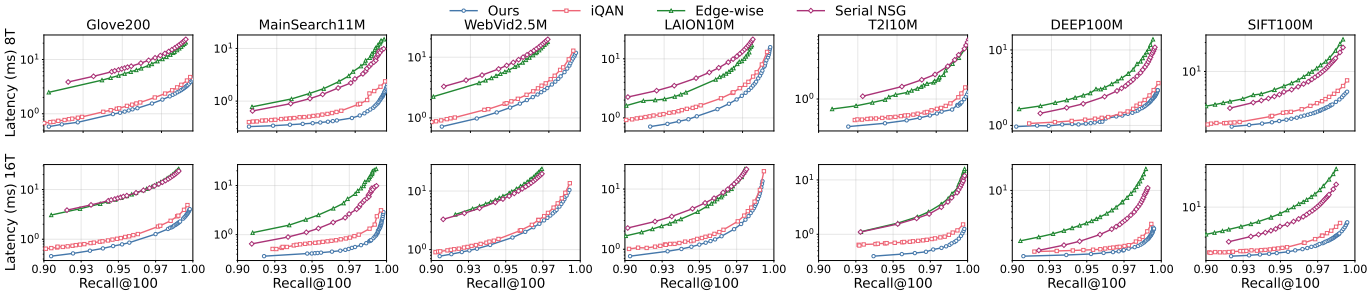



Fig. 8: Overall performance comparison. The first row shows the latency comparison under 8 threads, and the second row shows the latency comparison under 16 threads among iQAN, Edge-wise, Serial NSG and our method.

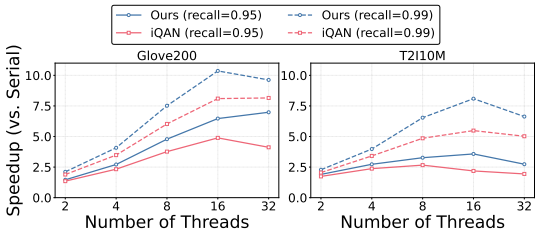



Fig. 9: The speedup ratio of our method and iQAN under different numbers of threads.

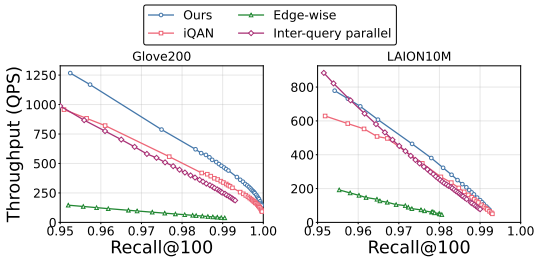



Fig. 10: Throughput under 16 threads.

presented in Figure 9. Taking serial search latency as the baseline, our method outperforms iQAN in terms of speedup. On the Glove200 dataset (Recall@100=0.99), our method achieves speedups of 7.51x and 10.36x with 8 and 16 threads, respectively, while iQAN achieves 6.03x and 8.11x speedups, respectively. On the T2I10M dataset, our speedup is 6.54x (8 threads) and 8.09x (16 threads), also surpassing iQAN. These results demonstrate that our method has better thread scalability than iQAN. Our approach achieves near-linear speedup with up to 8 threads. However, due to the limited divisibility of the task, our approach experiences diminishing returns with 32 threads, where adding more threads results in unnecessary computation. We also observed that our approach achieves greater speedup at higher recall rates. This is because higher recall rates require more computation, which is more effectively accelerated by parallelization.

### D. Throughput

We evaluate the throughput with 16 threads, comparing our method against iQAN, Edge-wise, and inter-query parallel search (Figure 10) on 2 datasets. The results indicate that our method significantly outperforms iQAN, improving throughput by 32.40%-33.20% at Recall@100=0.95 and 26.68% to 42.00% at Recall@100=0.99. The advantage over Edge-wise is even more pronounced due to its heavy synchronization overhead. Notably, at high recall targets, our method achieves higher throughput than inter-query parallel search while maintaining significantly lower query latency.

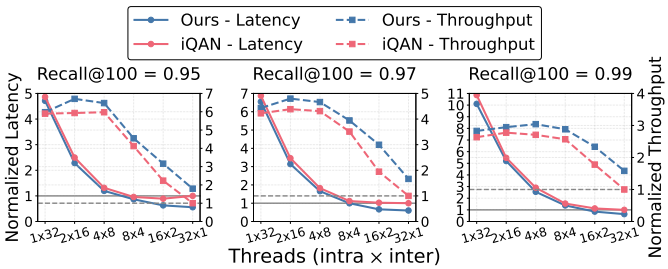



Fig. 11: Trade-off between query latency and throughput under different $intra \times inter$ thread configurations. The dashed line represents throughput, and the solid line represents latency.

### E. Throughput-Latency Trade-off

We investigate the throughput-latency trade-off by partitioning a fixed total number of threads. We define intra-query parallelism (intra) as the number of threads collaborating on a single query, and inter-query parallelism (inter) as the number of queries processed concurrently. Thus, in a 4×8 configuration with 32 total threads, 4 threads serve each query while the system handles 8 queries simultaneously.

Figure 11 illustrates the throughput-latency trade-off of our method and iQAN on the Glove200 dataset with 32 threads. To facilitate a clear comparison, all results are normalized to the performance of iQAN's 32×1 configuration, which serves as the baseline. The results show our method (blue lines) has a advantage over iQAN (red lines) in both latency and throughput. For instance, with a 16×2 configuration, our method increases throughput by up to 54.05% and reduces latency by as much as 35.24%. Furthermore, the results reveal that query latency decreases as number of intra-query thread increases. When the number of intra-query thread is small, the speedup is nearly linear. As the thread count increases, the speedup diminishes because the search task is not infinitely parallelizable. The query throughput shows a first plateaus and then declines trend as number of intra-query thread increases. The initial

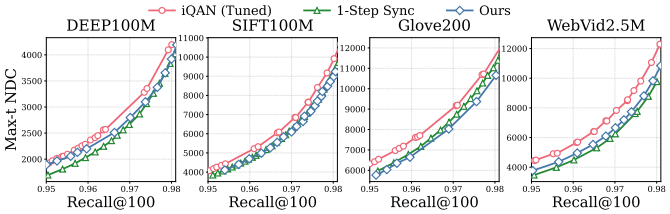



Fig. 12: The max-t NDC of iQAN, 1-Step Sync strategy and our method.

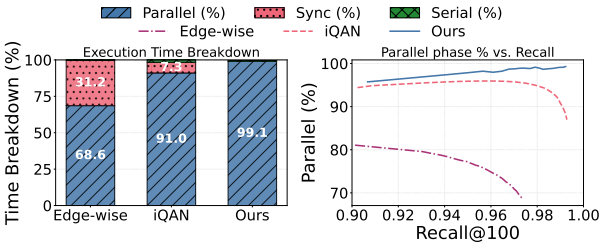



Fig. 13: The execution time breakdown of Edge-wise, iQAN and our method.

plateau occurs because the near-linear speedup achieved with a small intra-query thread count. As the number of intra-query thread becomes too large, the speedup diminishes, causing the query throughput to decrease. In practice, users can adjust the number of intra-query and inter-query thread according to their specific requirements for latency and throughput.

### F. Reduction of max-t NDC

We demonstrate the effectiveness of our method in reducing max-t NDC. As shown in Figure 12, we compare the max-t NDC of iQAN, 1-Step Sync strategy, and our method using 8 threads. Specifically, the 1-Step Sync strategy serves as the ideal lower bound of Max-t NDC under the Path-wise parallelism paradigm, as it enforces strict synchronization at every step to ensure minimal redundant computations. Compared to the state-of-the-art iQAN, our method consistently reduces the Max-t NDC. For instance, on the SIFT100M and WebVid2.5M datasets, we achieve a reduction of 8.98% and 14.47%, respectively. More importantly, the Max-t NDC curve of our method closely aligns with that of the 1-Step Sync strategy. This indicates that our method achieves near-optimal NDC reduction, approaching the lower bound of Path-wise parallelism in an asynchronous manner, and avoids the prohibitive synchronization overheads inherent to 1-Step Sync.

### G. Execution Time Breakdown Analysis

We conduct an execution time breakdown analysis on the WebVid2.5M dataset, partitioning execution time into Parallel, Synchronization, and Serial Phases (Figure 13). As shown in Figure 13 (left), at Recall@100=0.99, Edge-wise and iQAN spend significant time (31.2% and 7.3%, respectively) on synchronization. In contrast, our method eliminates explicit synchronization, dedicating over 99% of time to the Parallel Phase. Notably, the Serial Phase (result merging) is negligible. Using an $O(KlogN_t)$ multi-way merge, the result merging

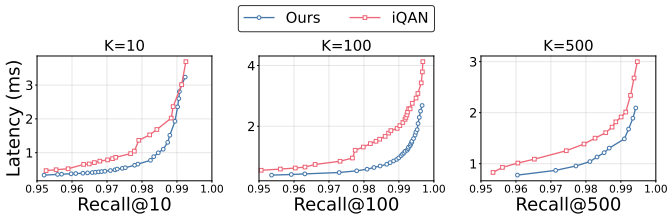



Fig. 14: Latency-Recall@$K$ performance comparison for $K = \{10, 100, 500\}$ under 8 threads.

takes only 0.004 ms for $K$=100 on MainSearch11M (approx. 0.3% of total latency), confirming that our design does not shift the bottleneck to the serial phase. Figure 13 (right) further shows that our parallel efficiency improves with higher recall targets, whereas Edge-wise and iQAN degrade due to increased synchronization overhead.

### H. Different Top-K

We evaluate the adaptability of our method across different Top-K values. As illustrated in Figure 14, we measure the query latency on MainSearch11M dataset with different $K$ under 8 threads. When Recall@$K$ is 0.95, compared with iQAN, our algorithm achieves 28.5%/34.1%/23.4% latency reduction when $K$ is 10/100/500, respectively. For a high recall target (i.e., Recall@$K$=0.99), our algorithm reduces latency by 25.2%/49.2%/22.5% when $K$ is 10/100/500, respectively. These results show that our method consistently outperforms iQAN across all evaluated $K$ values, maintaining superior performance even with large retrieval sets (e.g., $K$ = 500).

### I. Parameter Sensitivity Analysis

We evaluate the impact of the election hop $H_{elec}$, which determines the timing of the leader election. The results are shown in Figure 15. Theoretically, the reasonable range for $H_{elec}$ is $[0, L]$, as the BFiS process with candidate queue size $L$ usually nears convergence within $L$ hops. At small $H_{elec}$ values (e.g., $H_{elec}$=5, 10), the search performance is suboptimal. This is because the graph traversal typically experiences an initial volatile phase with drastically fluctuating candidate queues, followed by a stable refinement phase [80], [85]. Premature elections during the first phase (i.e., small $H_{elec}$) risk misidentifying the leader due to transient states. As $H_{elec}$ increases, the search performance stabilizes. If $H_{elec}$ exceeds $L$ (e.g., $H_{elec}$=1000), the search may complete before the election is even triggered. In this case, our algorithm degenerates into the No-Sync strategy (represented by the black line in Figure 15), losing the efficiency benefits of pruning. Empirically, we find that $H_{elec}$=50 serves as a robust setting across datasets, ensuring a reliable leader identification at the stable refinement phase.

### J. Performance on Low-Quality Indices

Real-world scenarios often prioritize faster index construction at the cost of reduced graph quality. We investigate the robustness of our method on indices built with relaxed parameters ($L$=200, $R$=32, $C$=200). As shown in Figure 16,

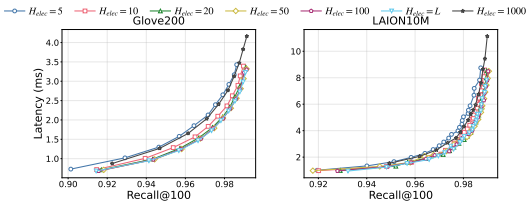



Fig. 15: Recall-Latency performance under different $H_{elec}$.

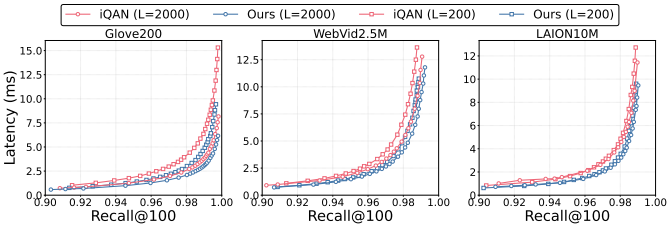



Fig. 16: The performance comparison on indices with relaxed construction parameters.

our method maintains robust performance advantages on low-quality indices (square markers) compared to high-quality ones ($L$=2000, circular markers). While lower graph quality inevitably increases query latency for both methods (by 10%-44%), our method consistently outperforms the baseline. Specifically, at Recall@100=0.98, our method reduces latency by 26.7%-31.7% compared to iQAN.

### *K. Ablation Study*

In this section, we conduct the ablation study to demonstrate the effectiveness of our proposed components, as shown in Figure 17. We first introduce a decoupled search paradigm, *ScatterSearch*, which eliminates the synchronization overhead. Experimental results show that *ScatterSearch* reduces query latency to 93% of that of iQAN. We then apply our search pruning strategy on *ScatterSearch*, further reducing query latency to 80% of the iQAN by eliminating long-tail computations. Finally, to fully utilize the computational resources of the pruned threads, we employ a work-stealing mechanism, bringing the final query latency down to 77% of the iQAN.

## VI. Related Work

ANNS algorithms can be broadly categorized into four main families. Hashing-based methods [24]–[30] employ hash functions designed to map similar data points to the same bucket with high probability. These methods demonstrate its effectiveness on small-scale datasets. Partition-based methods partition the data space into a multi dimensional tree structure [31]–[35] or clusters [36]–[42]. Quantization-based methods [43]–[51] compress the vectors into short codewords to improve computation efficiency. Graph based methods [52]–[68] construct a proximity graph, where vertices represent data points, and uses heuristic algorithm to establish edges between vertices, thereby providing high search efficiency while maintaining sparsity.

There has been a lot of work on parallel BFiS. Classic parallel BFiS algorithms typically target exact path planning

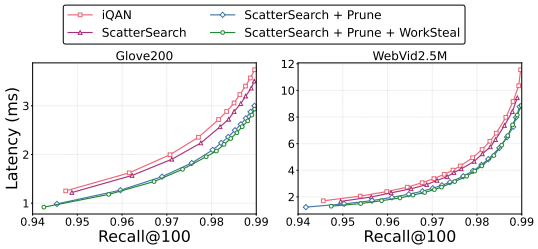



Fig. 17: Ablation study with 8 threads.

within implicit state spaces. In these scenarios, strict duplicate detection is mandatory to guarantee path optimality and prevent the exponential explosion of redundant states. Algorithms like Hash Distributed A* (HDA*) [86] and Parallel Best-NBlock-First (PBNF) [87] utilize complex hashing or partitioning mechanisms to achieve this. However, these strategies are not well-suited for Graph-based ANNS due to its nanosecond-level computation granularity and localized search patterns. Specifically, applying HDA* is inefficient because the overhead of hashing and transmitting nodes between threads far exceeds the cost of a simple vector distance calculation. Similarly, PBNF is suboptimal because an ANNS query typically visits only a tiny fraction of the graph (e.g., fewer than 10,000 vertices); enforcing parallel search across independently partitioned subgraphs results in a large number of useless accesses. In the field of parallel ANNS, the state-of-the-art algorithm iQAN [73] adopts partially similar concepts, such as the concurrent expansion of multiple nodes in a global candidate queue, and optimizes for latency by employing a dynamic synchronization strategy that relaxes the synchronization frequency.

Recently, GPU-based parallel ANNS methods [85], [88]–[91] leverage the GPU's massive computational power to achieve higher throughput than CPU-based ANNS. However, limited by the GPU memory capacity, a CPU–GPU co-processing paradigm is typically required to serve large-scale datasets effectively [92], [93].

## VII. Conclusion

In this paper, we deconstruct existing parallel graph-based ANNS algorithm and find that they all face a dilemma between reducing the synchronization overhead and reducing the computational overhead. To overcome this dilemma, we introduce *ScatterSearch*, a decoupled search paradigm to eliminate the synchronization overhead. Building upon this, we further propose a *leader-guided search pruning* strategy to reduce the computational overhead. Our comprehensive experiments on real-world datasets demonstrate the superior performance of our method.

## VIII. Acknowledgement

We thank the anonymous reviewers for their valuable feedback and suggestions. This research is supported by the National Natural Science Foundation of China (grant numbers, 62272252, 62272253), and the Fundamental Research Funds for Central Universities.

## IX. AI-Generated Content Acknowledgement

During the preparation of this work, the authors used Google Gemini 2.5 Pro to polish grammar and improve language. However, the tool was not used to generate algorithm, code, figures and tables. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

## References


[1] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, ser. RecSys '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 191–198. [Online]. Available: https://doi.org/10.1145/2959100.2959190

[2] P. Nigam, Y. Song, V. Mohan, V. Lakshman, W. A. Ding, A. Shingavi, C. H. Teo, H. Gu, and B. Yin, "Semantic product search," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2876–2885. [Online]. Available: https://doi.org/10.1145/3292500.3330759

[3] A. Pal, C. Eksombatchai, Y. Zhou, B. Zhao, C. Rosenberg, and J. Leskovec, "Pinnersage: Multi-modal user embedding framework for recommendations at pinterest," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2311–2320. [Online]. Available: https://doi.org/10.1145/3394486.3403280

[4] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu *et al.*, "Milvus: A purpose-built vector data management system," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2614–2627.

[5] R. Guo, X. Luan, L. Xiang, X. Yan, X. Yi, J. Luo, Q. Cheng, W. Xu, J. Luo, F. Liu *et al.*, "Manu: a cloud native vector database management system," *Proceedings of the VLDB Endowment*, vol. 15, no. 12, pp. 3548–3561, 2022.

[6] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," *IEEE Transactions on Big Data*, pp. 1–17, 2025.

[7] C. Wei, B. Wu, S. Wang, R. Lou, C. Zhan, F. Li, and Y. Cai, "Analyticdb-v: a hybrid analytical engine towards query fusion for structured and unstructured data," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 3152–3165, Aug. 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415541

[8] Y. Deng, Z. You, L. Xiang, Q. Li, P. Yuan, Z. Hong, Y. Zheng, W. Li, R. Li, H. Liu, K. Mouratidis, M. L. Yiu, H. Li, Q. Shen, R. Mao, and B. Tang, "Alayadb: The data foundation for efficient and effective long-context llm inference," 2025. [Online]. Available: https://arxiv.org/abs/2504.10326

[9] J. Pound, F. Chabert, A. Bhushan, A. Goswami, A. Pacaci, and S. R. Chowdhury, "Micronn: An on-device disk-resident updatable vector database," in *Companion of the 2025 International Conference on Management of Data*, ser. SIGMOD/PODS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 608–621. [Online]. Available: https://doi.org/10.1145/3722212.3724444

[10] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," 2025. [Online]. Available: https://arxiv.org/abs/2404.16130

[11] B. J. Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su, "Hipporag: neurobiologically inspired long-term memory for large language models," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24. Red Hook, NY, USA: Curran Associates Inc., 2025.

[12] Z. Wang, A. Liu, H. Lin, J. Li, X. Ma, and Y. Liang, "Rat: Retrieval augmented thoughts elicit context-aware reasoning in long-horizon generation," 2024. [Online]. Available: https://arxiv.org/abs/2403.05313

[13] J. Jin, Y. Zhu, Z. Dou, G. Dong, X. Yang, C. Zhang, T. Zhao, Z. Yang, and J.-R. Wen, "Flashrag: A modular toolkit for efficient retrieval-augmented generation research," in *Companion Proceedings of the ACM on Web Conference 2025*, ser. WWW '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 737–740. [Online]. Available: https://doi.org/10.1145/3701716.3715313

[14] Z. Guo, X. Ren, L. Xu, J. Zhang, and C. Huang, "Rag-anything: All-in-one rag framework," 2025. [Online]. Available: https://arxiv.org/abs/2510.12323

[15] W. Wu, H. Wang, B. Li, P. Huang, X. Zhao, and L. Liang, "Multirag: A knowledge-guided framework for mitigating hallucination in multi-source retrieval augmented generation," in *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 2025, pp. 3070–3083. [Online]. Available: https://doi.org/10.1109/ICDE65448.2025.00230

[16] X. Li, Q. Cai, Y. Shu, C. Guo, and B. Yang, "AID-SQL: adaptive in-context learning of text-to-sql with difficulty-aware instruction and retrieval-augmented generation," in *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 2025, pp. 3945–3957. [Online]. Available: https://doi.org/10.1109/ICDE65448.2025.00294

[17] B. Jin, H. Zeng, Z. Yue, J. Yoon, S. Arik, D. Wang, H. Zamani, and J. Han, "Search-r1: Training llms to reason and leverage search engines with reinforcement learning," *arXiv preprint arXiv:2503.09516*, 2025.

[18] T. Yang, Z. Yao, B. Jin, L. Cui, Y. Li, G. Wang, and X. Liu, "Demystifying and enhancing the efficiency of large language model based search agents," 2025. [Online]. Available: https://arxiv.org/abs/2505.12065

[19] J. Liang, G. Su, H. Lin, Y. Wu, R. Zhao, and Z. Li, "Reasoning rag via system 1 or system 2: A survey on reasoning agentic retrieval-augmented generation for industry challenges," 2025. [Online]. Available: https://arxiv.org/abs/2506.10408

[20] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, "A survey on rag meeting llms: Towards retrieval-augmented large language models," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 6491–6501. [Online]. Available: https://doi.org/10.1145/3637528.3671470

[21] Y. Chen, J. Zhang, B. Lu, Q. Zhang, C. Zhang, J. Luo, D. Liu, H. Jiang, Q. Chen, J. Liu, B. Ding, X. Yan, J. Jiang, C. Chen, M. Zhang, Y. Yang, F. Yang, and M. Yang, "Retroinfer: A vector-storage approach for scalable long-context llm inference," 2025. [Online]. Available: https://arxiv.org/abs/2505.02922

[22] D. Liu, M. Chen, B. Lu, H. Jiang, Z. Han, Q. Zhang, Q. Chen, C. Zhang, B. Ding, K. Zhang, C. Chen, F. Yang, Y. Yang, and L. Qiu, "Retrievalattention: Accelerating long-context llm inference via vector retrieval," 2024. [Online]. Available: https://arxiv.org/abs/2409.10516

[23] N. Altman and M. Krzywinski, "The curse(s) of dimensionality," *Nature Methods*, vol. 15, no. 6, pp. 399–400, Jun 2018. [Online]. Available: https://doi.org/10.1038/s41592-018-0019-x

[24] Y. Tian, X. Zhao, and X. Zhou, "DB-LSH: locality-sensitive hashing with query-based dynamic bucketing," in *ICDE*. IEEE, 2022, pp. 2250–2262.

[25] X. Zhao, Z. Chen, K. Huang, R. Zhang, B. Zheng, and X. Zhou, "Efficient approximate maximum inner product search over sparse vectors," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 3961–3974.

[26] Y. Wang, H. Ma, and D. Z. Wang, "Lider: an efficient high-dimensional learned index for large-scale dense passage retrieval," *Proc. VLDB Endow.*, vol. 16, no. 2, p. 154–166, Oct. 2022. [Online]. Available: https://doi.org/10.14778/3565816.3565819

[27] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 604–613. [Online]. Available: https://doi.org/10.1145/276698.276876

[28] J. He, W. Liu, and S.-F. Chang, "Scalable similarity search with optimized kernel hashing," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1129–1138. [Online]. Available: https://doi.org/10.1145/1835804.1835946

[29] A. Joly and O. Buisson, "Random maximum margin hashing," in *CVPR 2011*, 2011, pp. 873–880.

[30] J. Wang, W. Liu, A. X. Sun, and Y.-G. Jiang, "Learning hash codes with listwise supervision," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 3032–3039.

[31] H. Hu, J. Qiu, H. Wang, B. Liang, and S. Zou, "Dids: Double indices and double summarizations for fast similarity search," *Proc. VLDB Endow.*, vol. 17, no. 9, p. 2198–2211, May 2024. [Online]. Available: https://doi.org/10.14778/3665844.3665851

[32] K. Lampropoulos, F. Zardbani, N. Mamoulis, and P. Karras, "Adaptive indexing in high-dimensional metric spaces," *Proc. VLDB Endow.*, vol. 16, no. 10, p. 2525–2537, Jun. 2023. [Online]. Available: https://doi.org/10.14778/3603581.3603592

[33] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, Sep. 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[34] L. Cayton, "Fast nearest neighbor retrieval for bregman divergences," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 112–119. [Online]. Available: https://doi.org/10.1145/1390156.1390171

[35] E. Bernhardsson, "ANNOY library," https://github.com/spotify/annoy, accessed: 2025-03-19.

[36] Q. Xu, J. Yang, F. Zhang, J. Pan, K. Chen, Y. Shen, A. C. Zhou, and X. Du, "Tribase: A vector data query engine for reliable and lossless pruning compression using triangle inequalities," *Proc. ACM Manag. Data*, vol. 3, no. 1, Feb. 2025. [Online]. Available: https://doi.org/10.1145/3709743

[37] X. Zeng, L. Deng, P. Chen, X. Chen, H. Su, and K. Zheng, "Lira: A learning-based query-aware partition framework for large-scale ann search," in *Proceedings of the ACM on Web Conference 2025*, ser. WWW '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 2729–2741. [Online]. Available: https://doi.org/10.1145/3696410.3714633

[38] P. Zhang, B. Yao, C. Gao, B. Wu, X. He, F. Li, Y. Lu, C. Zhan, and F. Tang, "Learning-based query optimization for multi-probe approximate nearest neighbor search," *The VLDB Journal*, vol. 32, no. 3, p. 623–645, Sep. 2022. [Online]. Available: https://doi.org/10.1007/s00778-022-00762-0

[39] Y. Fu, C. Chen, X. Chen, W.-F. Wong, and B. He, "Optimizing the number of clusters for billion-scale quantization-based nearest neighbor search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 11, pp. 6786–6800, 2024.

[40] P. Sun, D. Simcha, D. Dopson, R. Guo, and S. Kumar, "Soar: Improved indexing for approximate nearest neighbor search," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 3189–3204. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/0973524e02a712af33325d0688ae6f49-Paper-Conference.pdf

[41] A. Babenko and V. Lempitsky, "The inverted multi-index," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 6, pp. 1247–1260, 2015.

[42] D. Baranchuk, A. Babenko, and Y. Malkov, "Revisiting the inverted indices for billion-scale approximate nearest neighbors," in *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part XII*. Berlin, Heidelberg: Springer-Verlag, 2018, p. 209–224. [Online]. Available: https://doi.org/10.1007/978-3-030-01258-8_13

[43] J. Gao and C. Long, "Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search," *Proc. ACM Manag. Data*, vol. 2, no. 3, May 2024. [Online]. Available: https://doi.org/10.1145/3654970

[44] J. Gao, Y. Gou, Y. Xu, Y. Yang, C. Long, and R. C.-W. Wong, "Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search," *Proc. ACM Manag. Data*, vol. 3, no. 3, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3725413

[45] L. Deng, P. Chen, X. Zeng, T. Wang, Y. Zhao, and K. Zheng, "Efficient data-aware distance comparison operations for high-dimensional approximate nearest neighbor search," 2024. [Online]. Available: https://arxiv.org/abs/2411.17229

[46] F. André, A.-M. Kermarrec, and N. Le Scouarnec, "Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan," *Proc. VLDB Endow.*, vol. 9, no. 4, p. 288–299, Dec. 2015. [Online]. Available: https://doi.org/10.14778/2856318.2856324

[47] F. Andre, A.-M. Kermarrec, and N. Le Scouarnec, "Quicker adc: Unlocking the hidden potential of product quantization with simd," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, p. 1666–1677, May 2021. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2019.2952606

[48] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[49] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 861–864.

[50] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 744–755, 2014.

[51] Q. Yue, X. Xu, Y. Wang, Y. Tao, and X. Luo, "Routing-guided learned product quantization for graph-based approximate nearest neighbor search," in *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 2024, pp. 4870–4883. [Online]. Available: https://doi.org/10.1109/ICDE60146.2024.00370

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

[53] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437913001300

[54] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *Proc. VLDB Endow.*, vol. 12, no. 5, p. 461–474, Jan. 2019. [Online]. Available: https://doi.org/10.14778/3303753.3303754

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

[56] Y. Gou, J. Gao, Y. Xu, and C. Long, "Symphonyqg: Towards symphonious integration of quantization and graph for approximate nearest neighbor search," *Proc. ACM Manag. Data*, vol. 3, no. 1, Feb. 2025. [Online]. Available: https://doi.org/10.1145/3709730

[57] Y. Fu, C. Chen, Y. Chen, W. Wong, and B. He, "Vista: Vector indexing and search for large-scale imbalanced datasets," in *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 2025, pp. 543–556. [Online]. Available: https://doi.org/10.1109/ICDE65448.2025.00047

[58] H. Wang, W. Wu, C. Luo, A. Bian, C. Meng, Y. Wu, and J. Sun, "Boosting accuracy and efficiency for vector retrieval with local scaling graph," in *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 2025, pp. 336–348. [Online]. Available: https://doi.org/10.1109/ICDE65448.2025.00032

[59] M. Chen, K. Zhang, Z. He, Y. Jing, and X. S. Wang, "Roargraph: A projected bipartite graph for efficient cross-modal approximate nearest neighbor search," *Proc. VLDB Endow.*, vol. 17, no. 11, p. 2735–2749, Jul. 2024. [Online]. Available: https://doi.org/10.14778/3681954.3681959

[60] Y. Peng, B. Choi, T. N. Chan, J. Yang, and J. Xu, "Efficient approximate nearest neighbor search in multi-dimensional databases," *Proc. ACM Manag. Data*, vol. 1, no. 1, May 2023. [Online]. Available: https://doi.org/10.1145/3588908

[61] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1475–1488, 2020.

[62] J. Vargas Muñoz, M. A. Gonçalves, Z. Dias, and R. da S. Torres, "Hierarchical clustering-based graphs for large scale approximate nearest neighbor search," *Pattern Recogn.*, vol. 96, no. C, Dec. 2019. [Online]. Available: https://doi.org/10.1016/j.patcog.2019.106970

[63] B. Harwood and T. Drummond, "Fanng: Fast approximate nearest neighbour graphs," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5713–5722.

[64] P. Chen, W.-C. Chang, J.-Y. Jiang, H.-F. Yu, I. Dhillon, and C.-J. Hsieh, "Finger: Fast inference for graph-based approximate nearest neighbor search," in *Proceedings of the ACM Web Conference 2023*, ser. WWW '23. New York, NY, USA: Association for

Computing Machinery, 2023, p. 3225–3235. [Online]. Available: https://doi.org/10.1145/3543507.3583318

[65] N. Hezel, U. K. Barthel, K. Schall, and K. Jung, "An exploration graph with continuous refinement for efficient multimedia retrieval," p. 657–665, 2024.

[66] K. Lu, M. Kudo, C. Xiao, and Y. Ishikawa, "Hvs: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search," *Proc. VLDB Endow.*, vol. 15, no. 2, p. 246–258, Oct. 2021. [Online]. Available: https://doi.org/10.14778/3489496.3489506

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

[68] M. Wang, W. Xu, X. Yi, S. Wu, Z. Peng, X. Ke, Y. Gao, X. Xu, R. Guo, and C. Xie, "Starling: An i/o-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment," *Proc. ACM Manag. Data*, vol. 2, no. 1, Mar. 2024. [Online]. Available: https://doi.org/10.1145/3639269

[69] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *Inf. Syst.*, vol. 87, no. C, Jan. 2020. [Online]. Available: https://doi.org/10.1016/j.is.2019.02.006

[70] M. Wang, X. Xu, Q. Yue, and Y. Wang, "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search," *Proc. VLDB Endow.*, vol. 14, no. 11, p. 1964–1978, Jul. 2021. [Online]. Available: https://doi.org/10.14778/3476249.3476255

[71] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," in *Findings of the Association for Computational Linguistics: ACL 2024*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 2318–2335. [Online]. Available: https://aclanthology.org/2024.findings-acl.137/

[72] J. Lee, F. Chen, S. Dua, D. Cer, M. Shanbhogue, I. Naim, G. H. Ábrego, Z. Li, K. Chen, H. S. Vera, X. Ren, S. Zhang, D. Salz, M. Boratko, J. Han, B. Chen, S. Huang, V. Rao, P. Suganthan, F. Han, A. Doumanoglou, N. Gupta, F. Moiseev, C. Yip, A. Jain, S. Baumgartner, S. Shahi, F. P. Gomez, S. Mariserla, M. Choi, P. Shah, S. Goenka, K. Chen, Y. Xia, K. Chen, S. M. K. Duddu, Y. Chen, T. Walker, W. Zhou, R. Ghiya, Z. Gleicher, K. Gill, Z. Dong, M. Seyedhosseini, Y. Sung, R. Hoffmann, and T. Duerig, "Gemini embedding: Generalizable embeddings from gemini," 2025. [Online]. Available: https://arxiv.org/abs/2503.07891

[73] Z. Peng, M. Zhang, K. Li, R. Jin, and B. Ren, "iqan: Fast and accurate vector search with efficient intra-query parallelism on multi-core architectures," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 313–328. [Online]. Available: https://doi.org/10.1145/3572848.3577527

[74] J. Luo, M. Zhang, K. Chen, X. Liao, Y. Shan, J. Jiang, and Y. Wu, "Efficient graph-based approximate nearest neighbor search achieving: Low latency without throughput loss," 2025. [Online]. Available: https://arxiv.org/abs/2504.20461

[75] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, p. 103–111, Aug. 1990. [Online]. Available: https://doi.org/10.1145/79173.79181

[76] C. Li, M. Zhang, D. G. Andersen, and Y. He, "Improving approximate nearest neighbor search through learned adaptive early termination," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2539–2554. [Online]. Available: https://doi.org/10.1145/3318464.3380600

[77] C. Schuhmann, R. Vencu, R. Beaumont, R. Kaczmarczyk, C. Mullis, A. Katta, T. Coombes, J. Jitsev, and A. Komatsuzaki, "Laion-400m: Open dataset of clip-filtered 400 million image-text pairs," 2021. [Online]. Available: https://arxiv.org/abs/2111.02114

[78] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[79] A. Babenko and D. Baranchuk, "Text-to-image dataset for billion-scale similarity search," 2021, accessed on October 20, 2025. [Online]. Available: https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search

[80] H. Guo and Y. Lu, "Achieving low-latency graph-based vector search via aligning best-first search algorithm with ssd," in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. Boston, MA: USENIX Association, 2025, pp. 171–186.

[81] Z. Hua, Q. Mo, Z. Yao, L. Cui, X. Liu, G. Wang, Z. Wei, X. Liu, T. Tang, S. Liu, and L. Qu, "Dynamically detect and fix hardness for efficient approximate nearest neighbor search," *Proc. ACM Manag. Data*, vol. 3, no. 6, Dec. 2025. [Online]. Available: https://doi.org/10.1145/3769783

[82] M. Bain, A. Nagrani, G. Varol, and A. Zisserman, "Frozen in time: A joint video and image encoder for end-to-end retrieval," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 1708–1718.

[83] A. B. Yandex and V. Lempitsky, "Efficient indexing of billion-scale datasets of deep descriptors," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2055–2063.

[84] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 861–864.

[85] Y. Chen, L. Cui, Z. Yao, H. Zhou, G. Wang, and X. Liu, "Algas: A low-latency gpu-based approximate nearest neighbor search system," in *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2025, pp. 199–209.

[86] A. Kishimoto, A. Fukunaga, and A. Botea, "Scalable, parallel best-first search for optimal sequential planning," 01 2009.

[87] E. Burns, S. Lemons, W. Ruml, and R. Zhou, "Parallel best-first search: Optimal and suboptimal solutions," 08 2010.

[88] H. Ootomo, A. Naruse, C. Nolet, R. Wang, T. Feher, and Y. Wang, "Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 4236–4247.

[89] F. Groh, L. Ruppert, P. Wieschollek, and H. P. A. Lensch, "Ggnn: Graph-based gpu nearest neighbor search," *IEEE Transactions on Big Data*, vol. 9, no. 1, pp. 267–279, 2023.

[90] Y. Yu, D. Wen, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "Gpu-accelerated proximity graph approximate nearest neighbor search and construction," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 552–564.

[91] K. V., S. Khan, S. Singh, H. V. Simhadri, and J. Vedurada, "Bang: Billion-scale approximate nearest neighbor search using a single gpu," 2025. [Online]. Available: https://arxiv.org/abs/2401.11324

[92] Z. Zhang, F. Liu, G. Huang, X. Liu, and X. Jin, "Fast vector query processing for large datasets beyond GPU memory with reordered pipelining," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 23–40. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/zhang-zili-pipelining

[93] B. Tian, H. Liu, Y. Tang, S. Xiao, Z. Duan, X. Liao, H. Jin, X. Zhang, J. Zhu, and Y. Zhang, "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)*. Santa Clara, CA: USENIX Association, Feb. 2025, pp. 171–185. [Online]. Available: https://www.usenix.org/conference/fast25/presentation/tian-bing