

RETHINKING MULTIPLE-QUERY OPTIMIZATION FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

Anonymous authors

Paper under double-blind review

ABSTRACT

Approximate nearest neighbor search (ANNS) over vector databases is a fundamental operation for many modern applications, where rapid processing of queries is of critical importance. In traditional database systems, which face the same requirement, multiple-query optimization (MQO) has been extensively studied to address this challenge. Although MQO is a general technique that exploits shared computation to process a set of queries more efficiently than evaluating each query in isolation, no analogous algorithmic strategy has yet been proposed for ANNS. To this end, we present a novel algorithmic MQO framework tailored to ANNS. The framework is universally applicable to graph-based ANNS methods, delivering significant speedups while keeping both the underlying index and the search algorithm intact. Specifically, we construct a minimum spanning tree over the query vectors and initialize each query’s search entry using the nearest neighbor returned by its parent in the tree, thereby revealing and exploiting opportunities for computation reuse. We empirically validate our framework across multiple ANNS methods and datasets, demonstrating its feasibility and effectiveness.

1 INTRODUCTION

Given a database $\mathcal{D}$ of vectors, the end goal of approximate nearest neighbor search (ANNS) is to retrieve, for each query vector, its closest neighbor in $\mathcal{D}$ in as little time as possible. This problem serves as a fundamental building block for many modern ML applications, from retrieval-augmented generation (Lewis et al., 2020; Borgeaud et al., 2021) and recommendation systems (Bachrach et al., 2014; Feng et al., 2022) to computer vision tasks (Aiger et al., 2023; Jang & Cho, 2020) and content generation and summarization (Baud & Aussem, 2023). To improve performance, much prior work has focused on devising advanced index structures to accelerate individual query processing (Bentley, 1975; Gionis et al., 1999; Wang et al., 2021; Weber et al., 1998; Malkov & Yashunin, 2018).

In traditional relational databases, which share a similar need for efficient query processing, multiple-query optimization (MQO) introduces a complementary dimension of optimization, offering a different perspective on the same challenge. MQO (Dokeroglu et al., 2014; Bayir et al., 2006; Marcus et al., 2019) identifies and reuses overlapping computations among multiple queries so that collective execution outperforms isolated processing. For relational databases, queries can be decomposed via relational algebra, enabling MQO algorithms to capture shared tasks and avoid redundant work.

Since MQO is a powerful algorithmic concept that can be applied generally, it is natural to ask whether its principles could accelerate query processing in ANNS. Indeed, a batched setting—where multiple queries are submitted to the index simultaneously—commonly arises in ANNS and is supported by many prior works (Jayaram Subramanya et al., 2019; Azizi et al., 2023; Liu et al., 2023; Zhao et al., 2020; Groh et al., 2023; Yu et al., 2022; Ootomo et al., 2024), yet they do not explicitly explore inter-query relationships to uncover reuse opportunities and exploit overlapping computations. To the best of our knowledge, an MQO framework tailored to ANNS has not yet been proposed. This difficulty stems from the fact that ANNS queries are vectors of real numbers, lacking the algebraic structure that makes shared computation explicit in relational settings.

Fortunately, MQO becomes feasible for ANNS once one exploits the inherent structure and details of modern ANNS algorithms. Modern ANNS solutions construct a proximity graph by connecting nearby data points and then perform a greedy search. Starting from an entry point selected according to a predefined policy, the search hops to the neighbor closest to the query until convergence. Because

choosing an entry point near the query reduces the number of hops and hence the search time, initializing the search closer to the query accelerates retrieval. When multiple queries are processed as a batch, we therefore propose using, for each query, the nearest neighbor result of any previously processed query that lies close in the embedding space as its entry point. Through this strategy, we enable indirect reuse of computation by supplying high-quality entry points.

The remaining challenge is to determine a processing strategy that maximizes proximity between each entry point and its corresponding query. We address this by organizing the query batch Q into a minimum spanning tree (MST) based on pairwise distances, which yields a connected structure spanning all queries with minimum total edge length. Each query’s search is then initialized at the nearest neighbor result of its parent in the MST, accelerating the overall batch search. We also present a formal theoretical analysis, deriving explicit bounds on the expected search latency to establish provable performance guarantees that underpin our MQO framework.

Furthermore, we introduce a rapid query preprocessing method for scenarios where the full batch is only revealed at runtime and end-to-end latency must be minimized. Constructing the exact MST in such cases is prohibitive, as it requires $\mathcal{O}(|Q|^2)$ pairwise-distance computations. Instead, we partition Q into multiple groups and employ an auxiliary ANNS index to build an approximate MST within each group, reducing computational burden while preserving high-quality processing strategy.

Experimental results show that our framework achieves up to a $1.84\times$ speedup in the search process and a fully end-to-end geometric mean speedup of $1.22\times$. Importantly, these gains are obtained without altering any underlying index structure or search algorithm, and the framework applies effectively to arbitrary graph-based indexes. Consequently, it offers a valuable, readily integrable technique for accelerating diverse ANNS applications without modifying their core structure.

2 PRELIMINARIES

2.1 APPROXIMATE NEAREST NEIGHBOR SEARCH

Given a dataset $\mathcal{D}$ of high-dimensional vectors, the nearest neighbor search problem seeks to construct a data structure that, for any query vector q , returns its closest neighbor in $\mathcal{D}$ with minimal latency. However, as the dimensionality of $\mathcal{D}$ increases, exact search methods suffer from the *curse of dimensionality* and effectively degrade to linear-time scanning (Indyk & Motwani, 1998; Marimont & Shapiro, 1979; Clarkson, 1994). Consequently, the combination of large dataset sizes and high dimensionality motivates the approximate nearest neighbor search (ANNS), in which one trades the accuracy for speed. In this context, accuracy is measured by recall, defined as the fraction of queries for which the true nearest neighbor is correctly identified.

Existing approximate nearest neighbor search (ANNS) techniques are classified into four groups: tree-based (Bentley, 1975; Beckmann et al., 1990), quantization-based (Jegou et al., 2010; Gao & Long, 2024), hashing-based (Gionis et al., 1999; Liu et al., 2014), and graph-based methods (Jayaram Subramanya et al., 2019; Malkov & Yashunin, 2018; Harwood & Drummond, 2016). Multiple researches experimentally report that graph-based approaches outperform others by achieving higher recall and lower latency (Wang et al., 2021; Malkov & Yashunin, 2018; Fu et al., 2019).

2.2 GRAPH-BASED APPROXIMATE NEAREST NEIGHBOR SEARCH

Graph-based ANNS methods (Malkov & Yashunin, 2018; Fu et al., 2019; Jayaram Subramanya et al., 2019; Harwood & Drummond, 2016; Li et al., 2019) construct a *graph index* in the form of a proximity graph $G = (V, E)$ over the dataset $\mathcal{D}$, where each vector $v \in \mathcal{D}$ corresponds to a node in V . The edge set E is determined with respect to the distance metric ρ and the underlying geometric structure of $\mathcal{D}$.

Although numerous algorithms exist for constructing G , most share a common search rou-

Algorithm 1 GREEDYBEAMSEARCH(ep, q, w, G)

```

1: beam  $\leftarrow [ep]$ , visited  $\leftarrow \emptyset$ 
2: while beam  $\not\subseteq$  visited do
3:    $v \leftarrow \underset{x \in \text{beam} \setminus \text{visited}}{\text{argmin}} \rho(x, q)$ 
4:   visited  $\leftarrow$  visited  $\cup \{v\}$ 
5:   beam  $\leftarrow$  beam  $\cup \mathcal{N}(v)$ 
6:   sort beam increasingly by  $\rho(\cdot, q)$ 
7:   beam  $\leftarrow$  top- $w$  entries of beam
8: end while
9: return beam

```

time. Given a graph index G and an entry point ep , a simple greedy search locates a query q by iteratively moving from the current node v to a neighbor $x \in \mathcal{N}(v)$ that minimizes $\rho(x, q)$. The search terminates once no neighbor is closer to q . Graph-based methods extend this to a *greedy beam search*, as formalized in Algorithm 1. This variant maintains a beam of size w tracking the w closest nodes encountered, with strict greedy search recovered when $w = 1$. The search concludes when every node in the beam has already been visited.

Fast convergence of Algorithm 1 depends critically on the choice of $\mathcal{N}(v)$, how the graph index is built. A naive approach is letting $\mathcal{N}(v)$ to consist of the M closest nodes of v . Recent works (Malkov & Yashunin, 2018; Fu et al., 2019; Jayaram Subramanya et al., 2019; Harwood & Drummond, 2016) refine this by enforcing diversification through pruning. They try to construct $\mathcal{N}(v)$ such that it consists of M nodes that are not only close to v but are also well separated from one another. Further details on the pruning methodology can be found on Appendix B. In practice, Algorithm 1 is invoked for each $v \in V$ to generate a candidate set, and pruning is then applied to finalize $\mathcal{N}(v)$.

2.3 MULTIPLE-QUERY OPTIMIZATION

For many computational tasks, processing a batch of inputs together can bring an extra dimension of optimization and higher throughput. In traditional databases, this challenge is known as the multiple-query optimization (MQO) (Sellis, 1988). MQO seeks an execution strategy that combines relational queries by exploiting shared intermediate results and eliminating redundant computation.

We illustrate MQO with a minimal example. Consider a table with attributes a and n , denoting users’ age and name, respectively, and two queries X and Y . Query X retrieves the names of individuals over 20 years old, while query Y counts the number of users older than 45. Query X is evaluated by applying a selection $\sigma_{a>20}$ followed by a projection Π_n , whereas query Y applies $\sigma_{a>45}$ and then an aggregation to count the results. Since $\sigma_{a>20}$ yields a superset of $\sigma_{a>45}$, the intermediate result of X can be reused for Y , reducing computation. A more detailed example is provided in Appendix C.

This example highlights the relational domain’s substantial overlapping computations and ample opportunities for reuse. A rich body of work has consequently devised MQO algorithms that construct processing strategies for batches to exploit such reuse (Dokeroglu et al., 2014; Bayir et al., 2006; Trummer & Koch, 2017; Marcus et al., 2019). By contrast, the algorithmic potential of MQO in ANNS remains largely underexplored. In this paper, we bridge this gap by identifying and exploiting MQO opportunities to achieve more efficient query processing in ANNS.

3 ENABLING MULTIPLE-QUERY OPTIMIZATION IN ANNS

To enable *multiple-query optimization*, two key prerequisites must be satisfied:

1. **Identification of reuse opportunities.** Analyze incoming queries to uncover common intermediate computations, enabling computational reuse and efficiency.
2. **Preprocessing algorithm for query batches.** Reorder or transform queries to materialize the identified *reuse potential* and fully exploit it throughout execution.

The fundamental reason MQO is feasible in traditional databases is that the first requirement, identification of reuse opportunities, is readily satisfied. Since relations consist of tuples with fixed schemas and finite attributes, factoring each query into a relational-algebra tree reveals identical or subsumable operations that naturally emerge across queries. This structural characteristic allows overlapped computations to be identified easily through syntactic or semantic analysis of the queries.

By contrast, in approximate nearest neighbor search, a query is simply a vector of real numbers with no additional structure. Consequently, potential overlaps across distinct queries remain hidden, making the discovery of reusable computations nontrivial.

3.1 MAIN IDEA: MQO FRAMEWORK FOR ANNS

The key to revealing reuse opportunities lies in leveraging the distinctive properties and design details of modern ANNS algorithms. A crucial observation is that modern ANNS solutions predominantly rely on graph-based methods, wherein a greedy beam search (Algorithm 1) is initiated from an

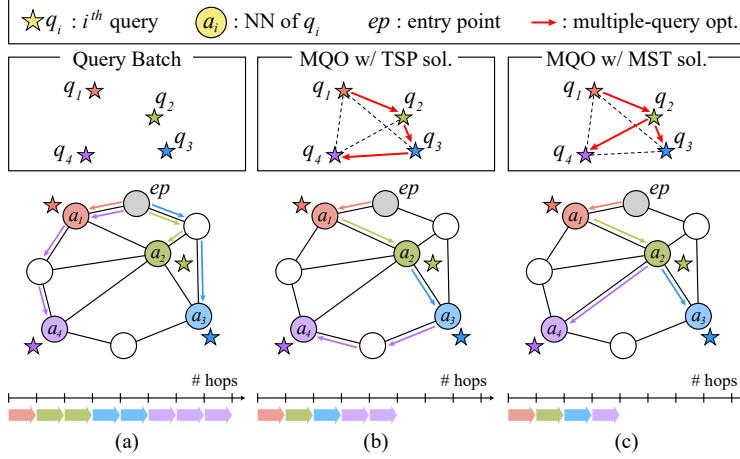


Figure 1: Overall idea for multiple-query optimization in ANNS.

entry point ep determined by a predefined policy that is independent of the query. For instance, HNSW (Malkov & Yashunin, 2018) uses the topmost node in its hierarchical graph, while NSG (Fu et al., 2019) and Vamana (Jayaram Subramanya et al., 2019) use a medoid of $\mathcal{D}$ regardless of the input query. However, since proximity graph tends to connect nearby points, the choice of entry point critically influences performance. Intuitively, selecting an entry point close to the query reduces the number of hops required for search, thereby minimizing search latency.

We propose replacing the predefined entry point with one tailored to each query. While this is infeasible for an isolated query, it becomes possible when processing a batch. Consider two successive queries $q_1, q_2 \in Q$ with $\rho(q_1, q_2)$ small. Searching for q_1 returns its nearest neighbor a_1 such that $\rho(a_1, q_1)$ is small. Given that ρ is a proper metric,

$$\rho(a_1, q_2) \leq \rho(a_1, q_1) + \rho(q_1, q_2),$$

guaranteeing a_1 to lie near q_2 . Initializing the search for q_2 at a_1 therefore provides a high-quality entry point for q_2 , enabling indirect reuse of computation without modifying the underlying index.

To realize and exploit such reuse potential, we must order the query batch $Q = \{q_1, \dots, q_{|Q|}\}$ so that successive queries remain close. Formally, we seek a permutation $Q_p = (q'_1, \dots, q'_{|Q|})$ minimizing

$$\sum_{i=2}^{|Q|} \rho(q'_{i-1}, q'_i),$$

which is precisely the open traveling salesman problem (TSP) on Q .

However, our mechanism does not require a single path. Once query q_i yields its nearest neighbor a_i , that same a_i can serve as the entry point for all subsequent queries. Thus it suffices to find any connected structure spanning the entire set Q . Such structure supports traversal from an arbitrary starting query, where the discovery sequence of oriented edges defines a processing strategy. Our objective is now to choose a set of query pairs $Q_p \subseteq \{(q, q') \mid q, q' \in Q\}$ spanning Q that minimizes

$$\sum_{\{q, q'\} \in Q_p} \rho(q, q'),$$

which is exactly the classical minimum spanning tree (MST) objective. Since every hamiltonian path is a spanning tree, MST-based preprocessing is superior over TSP-based formulation. Thus, we adopt the MST approach as our final strategy. We additionally include experimental comparisons with alternative query planning strategies, with results presented in Appendix D.

Figure 1 summarizes our discussion. In (a), the standard ANNS search always begins from the predefined entry point ep , irrespective of the query. In (b), we apply MQO via TSP: solving the TSP on Q yields a path $Q_p = \{q_1, q_2, q_3, q_4\}$, so that search for q_1 starts from ep and each subsequent q_i

initializes at the nearest neighbor a_{i-1} of the preceding query. In (c), we discard path constraint and utilize a tree structure: solving the MST on Q gives the edge set $Q_p = \{(q_1, q_2), (q_2, q_3), (q_2, q_4)\}$, allowing queries to use the best available entry point. For instance, the search for q_4 starts from a_2 instead of a_3 . This improved approach minimizes inter-query distances and, consequently, the total number of hops and node visits across the batch, ultimately reducing the overall latency.

3.2 THEORETICAL GUARANTEE

At a high level, our MQO framework posits that the distance L between the entry point and a query node directly dictates the search latency. This dependence was intuitively assumed in our earlier discussion on Section 3.1 without formal justification. Theorem 1 supplies the missing link. We prove that the *expected* running time grows at least linearly with L , so any preprocessing that systematically shortens L is guaranteed to reduce the search cost.

Setup. Let the dataset $\mathcal{D} = \{v_i\}_{i=1}^N$ be i.i.d. draws from a distribution on $\mathbb{R}^d$ with density f supported on a region $\mathcal{W} \subset \mathbb{R}^d$ with nonempty interior. Assume f admits a strictly positive lower bound $f_{\min}$ on $\mathcal{W}$. Construct a proximity graph G by connecting each vector to its M nearest neighbors. For simplicity, assume a strict greedy search ($w = 1$) and the Euclidean metric $\rho(x, y) = \|x - y\|_2$. Further assume a dense regime where $\ln N \gg \ln d$ and $N \gg M$.

Theorem 1 (Average running time, lower bound). *Let $ep, q \in \mathcal{D}$ be the entry point and the query vector, respectively, and let $L = \rho(ep, q)$ be the distance between them. Suppose the greedy search process starting from ep converges to q . Then there exists $c = c(d, f_{\min}) > 0$ independent of L such that the expected running time $\mathbb{E}[\mathcal{T}(ep, q)]$ satisfies*

$$\mathbb{E}[\mathcal{T}(ep, q)] \geq \left[c M \left(\frac{N}{M \ln N} \right)^{\frac{1}{d}} \right] L \quad \left(i.e., \mathbb{E}[\mathcal{T}(ep, q)] = \Omega(L) \right)$$

The main strategy of the proof is to bound the expected reduction in distance achieved by a single greedy hop. We choose a radius R large enough that, with high probability, all M nearest neighbors of the current node lie within the ball of radius R . The core insight is that, for this typical case, the greedy step decreases the distance by at most R while on the rare complementary event that some neighbor lies outside this ball, the step may cover a larger gap, but no more than the total distance L . By balancing these two regimes, we obtain a tight upper bound on the expected progress per hop. Summing this bound over the entire path to the query then yields a matching lower bound on the expected number of hops and on the overall search complexity. The full proof is given in Appendix A.

While Theorem 1 establishes a direct link between the distance L and the search cost, it does not quantify how much L can be reduced by our MST-based preprocessing. Theoretical studies on Euclidean MSTs show that, for $|Q|$ points sampled uniformly from the unit hypercube, the total MST weight scales as $\mathcal{O}(|Q|^{(d-1)/d})$ (Steele & Snyder, 1989). Hence, by ordering the queries according to the MST, our MQO framework guarantees that the average inter-query distance shrinks at rate $\mathcal{O}(|Q|^{-1/d})$, leading to progressively larger per-query latency reductions as $|Q|$ grows.

3.3 RAPID QUERY PREPROCESSING

For batched query scenarios in which queries accumulate incrementally over a time window or are known *a priori*, preprocessing can be performed during the accumulation period. By contrast, in scenarios where the full batch is only revealed at runtime, one must minimize the end-to-end runtime, including any preprocessing overhead. In such cases, constructing the MST is prohibitively expensive, as computing all pairwise distances incurs $\mathcal{O}(|Q|^2)$ time. Since the subsequent search scales linearly with $|Q|$, the quadratic term soon dominates, precluding the use of our MQO framework.

Breaking the quadratic barrier starts from noting that constructing a *single* spanning tree is unnecessary; instead, a *spanning forest* suffices. Beginning from the entry point governed by the predefined policy, each tree root can be entered exactly once, after which the traversal proceeds within the tree according to the method proposed in Section 3.1. Thus, with K trees, the predefined entry point is utilized K times rather than only once.

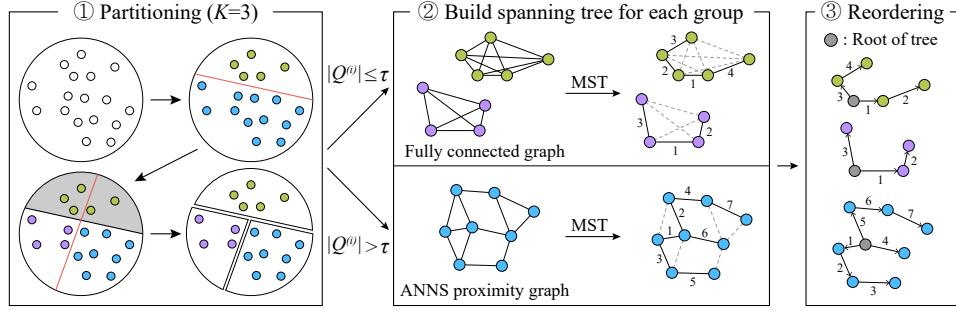


Figure 2: Overview of rapid query preprocessing scheme.

The overall procedure of our rapid preprocessing method is illustrated in Figure 2. ① We partition Q into K groups, $\{Q^{(1)} \dots Q^{(K)}\}$, via a recursive random-hyperplane strategy. At each iteration, we sample a random vector in $\mathbb{R}^d$ as the normal vector of the hyperplane. Then we project the queries in the largest group onto the vector, and partition that group into two subsets according to the sign of projection. The time required to build the spanning forest, where each tree is the MST of its corresponding group, is proportional to $\sum_{i=1}^K |Q^{(i)}|^2$. By always bisecting the largest group, we minimize $\max_i |Q^{(i)}|$, thereby effectively reducing the overall preprocessing cost.

After partitioning, ② we build a spanning tree *within* each group. We utilize an auxiliary ANNS to further tackle the quadratic dependency. If the group size $|Q^{(i)}|$ is sufficiently small, computing the exact MST for the group of queries does not incur intolerable overhead. On the other hand, if it is above a certain threshold τ , we first build a lightweight ANNS proximity graph over the group, symmetrize it to form an undirected graph, and then compute the MST on top of the ANNS graph. This confines the MST computation to a much smaller edge set, alleviating quadratic complexity.

Another important aspect worth noting is the ③ reordering step. We apply Prim’s algorithm (Prim, 1957) to each group to compute its MST, which produces oriented edges in discovery order. This orientation allows us to use the nearest neighbor result of the parent query as the entry point for the child query. Although one could process queries directly in Prim’s sequence, we instead reorder the forest to maximize locality for better cache utilization. For each tree, we select a root at random and perform a depth-first traversal, recording edges in the order they are first encountered. This *flattening* procedure ensures that consecutive edges correspond to spatially adjacent regions in the tree. While the total edge distance remains unchanged, the reordering mitigates cache inefficiencies caused by Prim’s arbitrary discovery order, thereby improving locality and reducing runtime.

3.4 OVERALL ALGORITHM

Algorithm 2 MQO-ANNS(Q, G, ep, w, K, τ)

```

1:  $A \leftarrow \text{EMPTYLIST}(|Q|)$ 
2: if needRapidPreprocessing then
3:    $F \leftarrow \text{MAKEFOREST}(Q, K, \tau)$ 
4: else
5:    $F \leftarrow \{\text{MST}(Q)\}$ 
6: end if
7:  $Q_p \leftarrow \text{REORDER}(F)$ 
8: for each  $(q_p, q_c)$  in  $Q_p$  do
9:    $s \leftarrow \begin{cases} ep & \text{if } q_c \text{ is root,} \\ A[p] & \text{otherwise} \end{cases}$ 
10:   $A[c] \leftarrow \text{GREEDYBEAMSEARCH}(s, q_c, w, G)$ 
11: end for
12: return  $A$ 

```

Algorithm 3 REORDER(F)

```

Require: forest  $F = \{T_1, \dots, T_K\}$ 
Ensure: list  $Q_p$  of (parent, child) pairs in
          DFS order with parent =  $\emptyset$  for each root
1:  $Q_p \leftarrow []$ 
2: for each tree  $T$  in  $F$  do
3:    $q_{\text{root}} \leftarrow \text{RANDSELECT}(T)$ 
4:    $\text{APPEND}(Q_p, (\emptyset, q_{\text{root}}))$ 
5:    $E \leftarrow \text{DFS}(T, q_{\text{root}})$ 
6:   for each edge  $(q_{\text{parent}}, q_{\text{child}})$  in  $E$  do
7:      $\text{APPEND}(Q_p, (q_{\text{parent}}, q_{\text{child}}))$ 
8:   end for
9: end for
10: return  $Q_p$ 

```

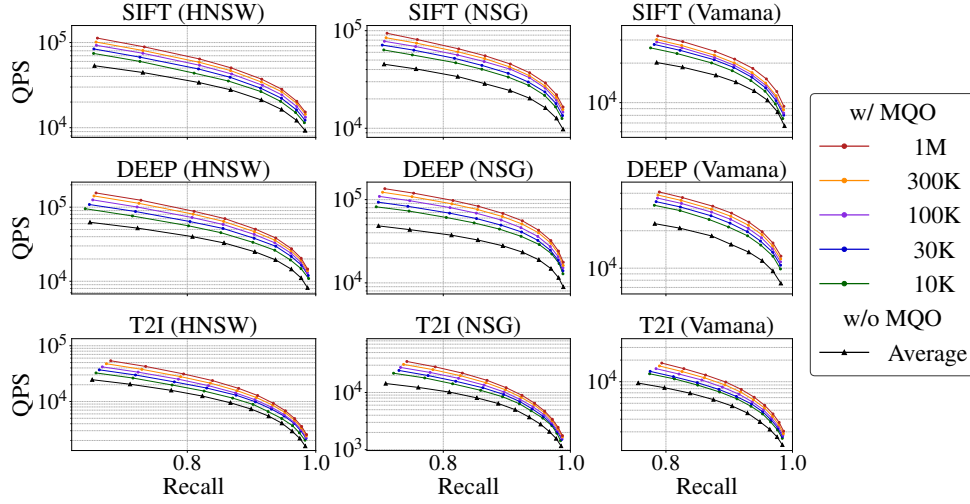


Figure 3: QPS - recall curve with varying query batch size. The reported results exclude preprocessing overhead, and MQO denotes standard MQO here.

For completeness, we outline the full framework in Algorithm 2. When end-to-end speedup is essential and rapid preprocessing is required, we build a spanning forest via partitioning and auxiliary ANNS (Line 3) as described in Section 3.3. Otherwise, computing the exact MST captures reuse opportunities to the fullest extent (Line 5). We then invoke Algorithm 3 to produce sequence of oriented query pairs Q_p , which defines the search plan (Line 7). Finally, we execute the search for each query (Lines 8–11). When a query is a root, we begin from the predefined entry point ep , and otherwise we initialize from the nearest neighbor result obtained for its parent in the tree. More detailed and complete versions of these algorithms are provided in Appendix E.

4 EVALUATION

We evaluate our MQO framework on three widely used graph-based ANNS indexes: HNSW (Malkov & Yashunin, 2018), NSG (Fu et al., 2019), and Vamana (Jayaram Subramanya et al., 2019). For each index, we adopt the official implementations¹, extending only the search API to allow specification of an external entry node. Unless stated otherwise, we use the default parameters when constructing each graph index. We used search on each graph index without applying MQO as the baseline. Experiments are conducted on three datasets: SIFT (uint8, 128-dim, Euclidean) (Amsaleg & Jégou, 2010), DEEP (float, 96-dim, Euclidean) (Yandex & Lempitsky, 2016; Baranchuk & Babenko, 2021a), and Yandex Text-to-Image (T2I) (float, 200-dim, inner-product) (Baranchuk & Babenko, 2021b). From the original dataset of 1B vectors, we randomly sample 1M vectors to form the dataset $\mathcal{D}$ and another 1M for the query batch Q , drawn from the same distribution without overlap. We report recall@1 for all experiments. The detailed setup for the evaluation can be found on Appendix F.

4.1 FEASIBILITY AND EFFECTIVENESS OF THE PROPOSED MQO FRAMEWORK

We begin by evaluating our MQO framework under the assumption that the MST, which captures the full extent of reuse opportunities among queries, is provided in advance. We refer to this setting as standard MQO. While Sections 3.1 and 3.2 provide intuitive explanations and theoretical guarantees, it remains essential to verify the effectiveness of standard MQO under practical conditions—namely, on pruned proximity graphs (Section 2.2), non-Euclidean metrics, and general beam sizes ($w > 1$).

¹HNSW (Malkov & Yashunin, 2018): <https://github.com/nmslib/hnswlib>

NSG (Fu et al., 2019): <https://github.com/ZJULearning/nsg>

Vamana (Jayaram Subramanya et al., 2019): <https://github.com/microsoft/DiskANN>

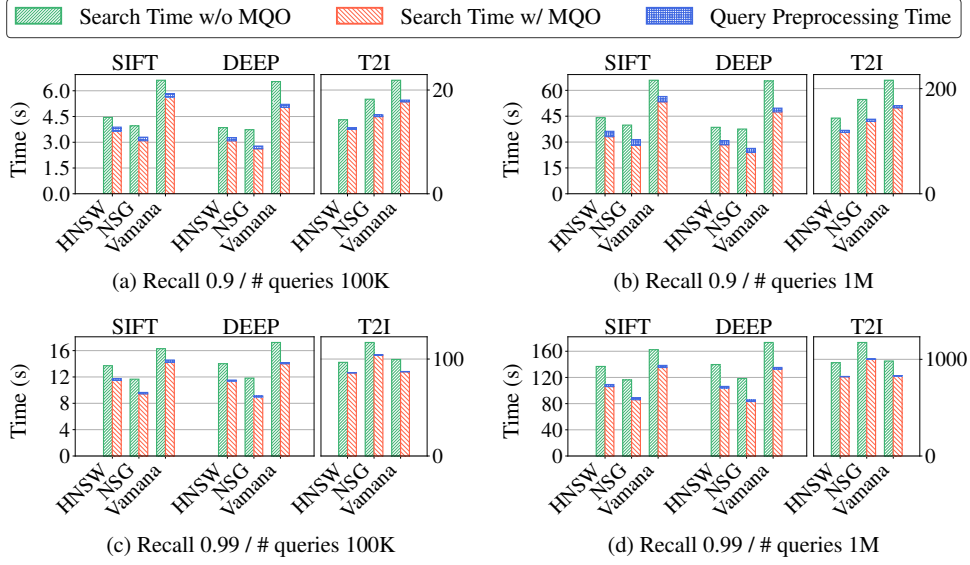


Figure 4: End-to-end time comparison for different recall and # queries. The results include preprocessing overhead, and MQO denotes lightweight MQO here.

Figure 3 presents the evaluation results, with each row corresponding to a dataset and each column to an ANNS index. In all cases, standard MQO yields consistent performance improvements, irrespective of the underlying index construction algorithm, the dataset, or the target recall level.

In particular, even at a small batch size of $|Q| = 10K$, standard MQO achieves a $1.32\times$ geometric mean speedup at a recall 0.9. Moreover, as predicted in Section 3.2, QPS further increases with batch size. For large batch of $|Q| = 1M$, the speedup rises to $1.84\times$ relative to the baseline without MQO.

We also observe a decrease in speedup as the recall target increases, due to larger beam sizes (w). For instance, with $|Q| = 1M$, the speedup is $1.84\times$ at recall 0.9 but decreases to $1.62\times$ at recall 0.99. This behavior arises because the search algorithm persists until the beam is fully populated and convergence is reached. When the beams are large, the algorithm may continue exploring even after the true nearest neighbor has been discovered. Consequently, the advantage conferred by standard MQO’s improved entry points is partially diminished by the convergence-based termination criterion.

4.2 END-TO-END SPEEDUP

Next, we consider a scenario in which the entire query batch is revealed immediately before the search begins, leaving no opportunity for extensive preprocessing. In such cases, constructing the MST as a preprocessing step can become more expensive than the search itself, especially as the batch size increases, as illustrated in Appendix G. To address this, we bypass MST construction and instead adopt the rapid query preprocessing method introduced in Section 3.3, aiming to optimize the overall runtime across both the preprocessing and search phases. We denote this variant lightweight MQO, and compare its end-to-end performance against a baseline search without MQO.

Figure 4 presents the results. lightweight MQO achieves a $1.22\times$ geometric mean speedup over the non-MQO baseline, averaged across recall targets $\{0.9, 0.99\}$ and batch sizes $\{100K, 1M\}$. Even when including preprocessing latency, significant gains arise from reordering query execution and reassigning entry points. Notably, a net end-to-end speedup is maintained at large batch sizes (e.g., $|Q| = 1M$), as our rapid query preprocessing scales efficiently with batch size.

Specifically, when comparing speedups at recall targets of 0.9 and 0.99, we observe comparable performance. As detailed in Section 4.1, the absolute gap between MQO and the non-MQO baseline narrows at higher recall levels. However, achieving higher recall requires longer search durations, which reduces the relative impact of preprocessing. These opposing effects effectively cancel out,

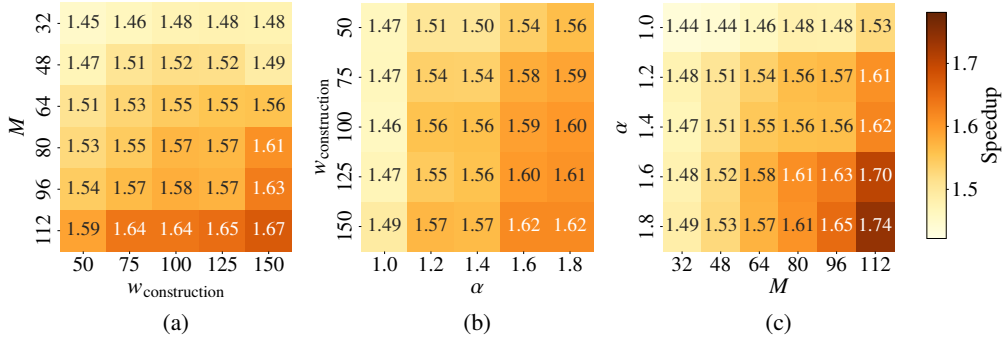


Figure 5: Geometric mean of speedup by standard MQO with Vamana for pairs of parameters (a) $w_{\text{construction}}-M$, (b) $\alpha-w_{\text{construction}}$, and (c) $M-\alpha$.

yielding a near-constant end-to-end speedup across the evaluated recall range. The detailed analysis of our approximation method is given in Appendix G.

4.3 EFFECT OF GRAPH PARAMETERS

Another question of practical interest is how the performance of our MQO framework evolves as the graph index is tuned. To investigate this, we use the DEEP dataset with $|Q| = 100\text{K}$ and systematically vary the three construction knobs of Vamana—maximum out-degree M , the construction beam size $w_{\text{construction}}$, and pruning coefficient α (denoted R , L , and α respectively in (Jayaram Subramanya et al., 2019)). Figure 5 reports the speedup at recall 0.9 for each pairwise combination of these parameters, with the third parameter aggregated via its geometric mean.

In summary, more complex, higher-quality graphs consistently yield greater speedups. Specifically, increasing M and $w_{\text{construction}}$ enhances graph size and fidelity, while raising α enforces more aggressive pruning. Each of these adjustments improves graph quality, thereby reducing the beam size required to achieve a given recall and, consequently, translating into higher speedup (Section 4.1). Comprehensive results on correlation between each parameter and required beam size or the beam size and the speedup are provided in Appendix H.

5 CONCLUSION AND DISCUSSION

In this paper, we have presented a multiple-query optimization framework toward approximate nearest neighbor search, thereby broadening the scope of optimization. By strategically reordering query execution and selecting tailored entry points for each query, our approach enhances search performance without any modification to the underlying index structure or the search algorithm.

We note that our MQO framework can achieve greater benefits in real-world scenarios. For our empirical evaluation, we assumed that both the dataset $\mathcal{D}$ and the query batch Q follow the exact same distribution, which is arguably the least favorable setting for MQO, as queries exhibit no inherent locality, interleave with data points, and preclude any natural clustering. In contrast, real-world workloads often yield batched queries with strong spatial locality. A prime example is retrieval-augmented generation (RAG) systems, in which a user’s original query is rewritten or augmented by multiple methods (Mao et al., 2021; Gao et al., 2023; Wang et al., 2024; Shen et al., 2024; Chan et al., 2024; Wang et al., 2025), producing a batch of related queries on which ANNS is performed. Since these queries share a common origin, they tend to cluster in the embedding space, increasing opportunities for computation reuse and amplifying the advantages of MQO.

There remains ample opportunity to extend our preliminary MQO framework. One interesting direction is to consider index construction. In graph-based ANNS, building the graph index essentially amounts to executing the search algorithm over an entire batch of queries, a setting that aligns naturally with our framework. Exploring whether the proposed framework can accelerate index building while preserving quality constitutes an intriguing avenue for future work.

6 REPRODUCIBILITY STATEMENT

We provide code in the supplementary material to demonstrate the methods proposed in this paper. A small dataset and a README file are included to facilitate usage. By following the provided instructions, one can reproduce the results obtained with our framework and compare them against the baseline.

REFERENCES

- Dror Aiger, Andre Araujo, and Simon Lynen. Yes, we can: Constrained approximate nearest neighbors for local feature-based visual localization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 13339–13349. IEEE Computer Society, October 2023.
- L. Amsaleg and H. Jégou. Datasets for approximate nearest neighbor search. <http://corpus-texmex.irisa.fr/>, 2010. Liscence: CC0, Accessed: 2025-05-09.
- Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 271–280. Society for Industrial and Applied Mathematics, 1993.
- Ilias Azizi, Karima Echihabi, and Themis Palpanas. Elpis: Graph-based similarity search for scalable data science. *Proceedings of the VLDB Endowment*, 16(6):1548–1559, February 2023.
- Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys)*, pp. 257–264. Association for Computing Machinery, 2014.
- Dmitry Baranchuk and Artem Babenko. Yandex deep dataset. <https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search#14h2>, 2021a. Liscence: CC BY 4.0, Accessed: 2025-05-09.
- Dmitry Baranchuk and Artem Babenko. Yandex text-to-image (t2i) dataset. <https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search#13h2>, 2021b. Liscence: CC BY 4.0, Accessed: 2025-05-09.
- Florian Baud and Alex Aussem. Non-parametric memory guidance for multi-document summarization. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing (RANLP)*, pp. 153–158. INCOMA Ltd., Shoumen, Bulgaria, September 2023.
- Murat Ali Bayir, Ismail H Toroslu, and Ahmet Cosar. Genetic algorithm for the multiple-query optimization problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(1):147–153, 2006.
- Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pp. 322–331. Association for Computing Machinery, 1990.
- Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.
- Jon Louis Bentley. A parallel algorithm for constructing minimum spanning trees. *Journal of Algorithms*, 1(1):51–59, 1980.
- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, T. W. Hennigan, Saffron Huang, Lorenzo Maggione, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and L. Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2021.

- Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. RQ-RAG: Learning to refine queries for retrieval augmented generation. In *First Conference on Language Modeling (COLM)*, 2024.
- Kenneth L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the Tenth Annual Symposium on Computational Geometry (SCG)*, pp. 160–164. Association for Computing Machinery, 1994.
- L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.
- Tansel Dokeroglu, Murat Ali Bayır, and Ahmet Cosar. Integer linear programming solution for the multiple query optimization problem. In *Information Sciences and Systems 2014*, pp. 51–60. Springer, Springer International Publishing, 2014.
- Chao Feng, Wuchao Li, Defu Lian, Zheng Liu, and Enhong Chen. Recommender forest for efficient retrieval. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 38912–38924. Curran Associates Inc., 2022.
- Cong Fu and Deng Cai. Efanna : An extremely fast approximate nearest neighbor search algorithm based on knn graph, 2016.
- Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment*, 12(5):461–474, January 2019.
- Jianyang Gao and Cheng Long. Rabbitq: quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 2(3):1–27, 2024.
- Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1762–1777. Association for Computational Linguistics, July 2023.
- Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, volume 99, pp. 518–529. Morgan Kaufmann Publishers Inc., 1999.
- Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P. A. Lensch. Ggnn: Graph-based gpu nearest neighbor search. *IEEE Transactions on Big Data*, 9(1):267–279, 2023.
- Ben Harwood and Tom Drummond. Fanng: Fast approximate nearest neighbour graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5713–5722. IEEE, 2016.
- Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, pp. 604–613, 1998.
- Young Kyun Jang and Nam Ik Cho. Generalized product quantization network for semi-supervised image retrieval. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3417–3426. IEEE, 2020.
- Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*, volume 32, pp. 13766–13776. Curran Associates, Inc., 2019.
- Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 33(1):117–128, 2010.

- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 9459–9474. Curran Associates Inc., 2020.
- Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 32(8):1475–1488, 2019.
- Yingfan Liu, Jiangtao Cui, Zi Huang, Hui Li, and Heng Tao Shen. Sk-lsh: an efficient index structure for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 7(9):745–756, 2014.
- Yingfan Liu, Chaowei Song, Hong Cheng, Xiaofang Xia, and Jiangtao Cui. Accelerating massive queries of approximate nearest neighbor search on high-dimensional data. *Knowledge and Information Systems*, 65(10):4185–4212, May 2023.
- Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 42(4):824–836, 2018.
- Magdalen Dobson Manohar, Zheqi Shen, Guy Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. Parlayann: Scalable and deterministic parallel graph-based approximate nearest neighbor search algorithms. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pp. 270–285. Association for Computing Machinery, 2024.
- Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Generation-augmented retrieval for open-domain question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4089–4100. Association for Computational Linguistics, August 2021.
- Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *Proceedings of the VLDB Endowment*, 12(11):1705–1718, 2019.
- R. B. Marimont and M. B. Shapiro. Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics*, 24(1):59–70, 08 1979.
- Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 4236–4247, 2024.
- Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.
- Timos K Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.
- Tao Shen, Guodong Long, Xiubo Geng, Chongyang Tao, Yibin Lei, Tianyi Zhou, Michael Blumenstein, and Daxin Jiang. Retrieval-augmented retrieval: Large language models are strong zero-shot retriever. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15933–15946. Association for Computational Linguistics, August 2024.
- J Michael Steele and Timothy Law Snyder. Worst-case growth rates of some classical problems of combinatorial optimization. *SIAM Journal on Computing*, 18(2):278–287, 1989.
- Godfried T Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.
- Immanuel Trummer and Christoph Koch. Multi-objective parametric query optimization. *Communications of the ACM*, 60(10):81–89, 2017.

- Haoyu Wang, Ruirui Li, Haoming Jiang, Jinjin Tian, Zhengyang Wang, Chen Luo, Xianfeng Tang, Monica Xiao Cheng, Tuo Zhao, and Jing Gao. BlendFilter: Advancing retrieval-augmented large language models via query generation blending and knowledge filtering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1009–1025. Association for Computational Linguistics, November 2024.
- Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 14:1964–1978, 2021.
- Shuting Wang, Xin Yu, Mang Wang, Weipeng Chen, Yutao Zhu, and Zhicheng Dou. RichRAG: Crafting rich responses for multi-faceted queries in retrieval-augmented generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 11317–11333. Association for Computational Linguistics, January 2025.
- Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB)*, volume 98, pp. 194–205. Morgan Kaufmann Publishers Inc., 1998.
- Artem Babenko Yandex and Victor Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2055–2063. IEEE, 2016.
- Yuanhang Yu, Dong Wen, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. Gpu-accelerated proximity graph approximate nearest neighbor search and construction. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp. 552–564, 2022.
- Weijie Zhao, Shulong Tan, and Ping Li. Song: Approximate nearest neighbor search on gpu. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1033–1044, 2020.

A PROOF OF THEOREM 1

Proof. **1. The d -dimensional ball.** Define $\mathcal{B}_x^d(r)$ as the ball of radius r centered at $x \in \mathbb{R}^d$:

$$\mathcal{B}_x^d(r) = \{y \in \mathbb{R}^d \mid \rho(x, y) \leq r\}.$$

Under the Euclidean metric, its volume is

$$V_d(r) = \frac{\pi^{d/2}}{\Gamma(d/2 + 1)} r^d = V_d(1) r^d.$$

Since $f(x) \geq f_{\min} > 0$ on $\mathcal{W}$, whenever $\mathcal{B}_x^d(r) \subseteq \mathcal{W}$ we have

$$\Pr[v \in \mathcal{B}_x^d(r)] \geq f_{\min} V_d(r) = \mu r^d, \quad \mu := f_{\min} V_d(1).$$

2. Effective radius R_δ . For any $\delta \in (0, 1)$, define R_δ so that whenever $r > R_\delta$, the probability of having at least M dataset points in $\mathcal{B}_x^d(r)$ is at least $1 - \delta$. Concretely, letting

$$p = \Pr[|\{v \in \mathcal{D} \mid v \in \mathcal{B}_x^d(r) \subseteq \mathcal{W}\}| \geq M],$$

we require $p > 1 - \delta$ given $r > R_\delta$. We show that

$$R_\delta = \left(\frac{1}{\mu(N-M+1)} \ln \frac{N^M}{\delta} \right)^{\frac{1}{d}}.$$

Write $X := |\{v \in \mathcal{D} : v \in \mathcal{B}_x^d(r)\}| \sim \text{Bin}(N, p_0)$ with $p_0 = \Pr[v \in \mathcal{B}_x^d(r)] \geq \mu r^d$. By $(1-x) \leq e^{-x}$, we have

$$\begin{aligned} 1 - p &= \sum_{i=0}^{M-1} \binom{N}{i} p_0^i (1-p_0)^{N-i} \leq \sum_{i=0}^{M-1} \binom{N}{i} (1-\mu r^d)^{N-i} \\ &\leq M \binom{N}{M} (1-\mu r^d)^{N-M+1} \\ &\leq M \binom{N}{M} e^{-\mu r^d (N-M+1)} \leq N^M e^{-\mu r^d (N-M+1)}. \end{aligned}$$

Then, from

$$r > R_\delta \implies \mu r^d (N-M+1) > \ln \frac{N^M}{\delta} \implies N^M e^{-\mu r^d (N-M+1)} < \delta,$$

we conclude $r > R_\delta$ implies $1 - p < \delta$, so indeed $p > 1 - \delta$.

3. Expected decrease in distance. Consider a single hop (iteration) of the greedy search process. Let

$$\Delta = \rho(x, q) - \rho(\text{next}(x), q).$$

be the decrease in distance after we travel from the current node x to one of its neighbor $\text{next}(x)$ that is closest to the query.

For a given δ , with probability higher or equal than $1 - \delta$, all the M nearest neighbors reside inside $\mathcal{B}_x^d(R_\delta)$. In such cases, the distance to the query cannot decrease more than R_δ in that hop. Otherwise (an event of probability at most δ), some points among the M nearest neighbors reside outside $\mathcal{B}_x^d(R_\delta)$. For such cases, the decrease in distance can be bounded above trivially by L . Thus, for any $\delta \in (0, 1)$ we have :

$$\mathbb{E}[\Delta] = \mathbb{E}[\rho(x, q) - \rho(\text{next}(x), q)] \leq R_\delta + \delta L.$$

To obtain a tight upper bound on $\mathbb{E}[\Delta]$, we choose

$$\delta^* = \frac{1}{L} \left(\frac{M \ln N}{\mu N} \right)^{\frac{1}{d}},$$

so that

$$\delta^* L = \left(\frac{M \ln N}{\mu N} \right)^{\frac{1}{d}}.$$

Recall that

$$R_\delta = \left(\frac{1}{\mu(N-M+1)} \ln \frac{N^M}{\delta} \right)^{\frac{1}{d}}.$$

Substituting $\delta = \delta^*$ yields

$$R_{\delta^*} = \left(\frac{1}{\mu(N-M+1)} \left\{ M \ln N + \ln L + \frac{1}{d} \ln(\mu N) - \frac{1}{d} \ln(M \ln N) \right\} \right)^{\frac{1}{d}}.$$

As $\ln N \gg \ln d$, $\ln L$ is negligible compared to $\ln N$. Thus the terms inside the logarithm can be bounded above by $(M+1) \ln N$ when N is large. Furthermore, as $N \gg M$, we have $\frac{M+1}{N-M+1} \leq \frac{2M}{N}$. Thus,

$$R_{\delta^*} \leq 2^{\frac{1}{d}} \left(\frac{M \ln N}{\mu N} \right)^{\frac{1}{d}}$$

Choosing $\delta = \delta^*$ thus balances the contribution of two terms R_{δ^*} and $\delta^* L$ so that neither dominates.

Thus, the expected decrease in distance per hop satisfies

$$\mathbb{E}[\Delta] \leq R_{\delta^*} + \delta^* L \leq (1 + 2^{1/d}) \left(\frac{M \ln N}{\mu N} \right)^{\frac{1}{d}}$$

This completes the derivation of the upper bound on the expected decrease in distance per hop.

4. Expected running time of the algorithm. Let τ be the sequence of nodes visited in the greedy search algorithm.

$$\tau = (x_0 = ep, x_1, x_2, \dots, x_{T-1}, x_T = q),$$

where T is the stopping time, the number of hops needed for the algorithm to reach q .

$$T = \min\{i \geq 0 : x_i = q\}.$$

Also let

$$\Delta_i = \rho(x_{i-1}, q) - \rho(x_i, q)$$

denote the decrease in distance from q at the i -th hop.

As we prove for the case where greedy search successfully converged to q , by construction $\Delta_i \geq 0$ and $\sum_{i=1}^T \Delta_i = L$.

In the previous step, we showed that for every x

$$\mathbb{E}[\Delta_i \mid x_{i-1} = x] \leq \bar{\Delta}, \quad \bar{\Delta} := (1 + 2^{1/d}) \left(\frac{M \ln N}{\mu N} \right)^{1/d}.$$

Consequently

$$\mathbb{E}[\Delta_i] \leq \bar{\Delta} \quad \text{for all } i \geq 1.$$

With the non-negativity of the Δ_i , Tonelli's theorem gives us

$$L = \mathbb{E}\left[\sum_{i=1}^T \Delta_i\right] = \sum_{i=1}^{\infty} \mathbb{E}[\Delta_i \mathbf{1}_{\{i \leq T\}}] \leq \sum_{i=1}^{\infty} \mathbb{E}[\Delta_i] \leq \bar{\Delta} \sum_{i=1}^{\infty} \Pr\{T \geq i\} = \bar{\Delta} \mathbb{E}[T].$$

Therefore,

$$\mathbb{E}[T] \geq \frac{L}{\bar{\Delta}} = \frac{\mu^{1/d}}{1 + 2^{1/d}} \left(\frac{N}{M \ln N} \right)^{1/d} L.$$

Connecting the runtime with the number of hops is straightforward. Since at each hop the algorithm inspects M edges and a single ℓ_2 distance evaluation in $\mathbb{R}^d$ costs $\Theta(d)$ arithmetic operations, there exists an absolute constant $\alpha > 0$ such that the expected running time satisfies

$$\mathbb{E}[\mathcal{T}(ep, q)] \geq (\alpha d) M \mathbb{E}[T].$$

Combining with the bound on $\mathbb{E}[T]$ and absorbing α into the constant gives

$$\mathbb{E}[\mathcal{T}(ep, q)] \geq \underbrace{\frac{\alpha d \mu^{1/d}}{1 + 2^{1/d}}}_{=: c(d, f_{\min})} M \left(\frac{N}{M \ln N} \right)^{1/d} L.$$

This matches the statement of Theorem 1. □

B PRUNING IN GRAPH-BASED ANNS

Instead of using a naive k NN graph as a graph index, ANNS methods employ an additional pruning step. The pruning step is inspired by the relative neighborhood graph (RNG) (Toussaint, 1980) and its relaxed variant, the sparse neighborhood graph (SNG) (Arya & Mount, 1993). In an RNG, an edge (v_i, v_j) is considered redundant and pruned if there exists a third node v such that (v_i, v_j) is the longest side of triangle $\triangle v_i v_j v$. Equivalently, an edge (v_i, v_j) is retained in RNG if

$$\forall v \in V, \rho(v, v_i) \geq \rho(v_i, v_j) \text{ or } \rho(v, v_j) \geq \rho(v_i, v_j).$$

The SNG applies the similar principle with fewer eliminations so that it guarantees the strict greedy search, $\text{GREEDYBEAMSEARCH}(ef, q, 1, G)$, always converge to its true nearest neighbor for all $q \in \mathcal{D}$. This desirable property motivates the use of an SNG as the underlying proximity graph for many state-of-the-art ANNS methods (Malkov & Yashunin, 2018; Fu et al., 2019; Jayaram Subramanya et al., 2019; Harwood & Drummond, 2016).

Constructing the exact SNG is computationally prohibitive, as it requires applying the pruning rule to the full node set V for every $v \in V$. Instead, ANNS methods approximate SNG by invoking Algorithm 1 during graph construction to obtain a compact set of candidates per node and running pruning only on this smaller set. For instance, (Malkov & Yashunin, 2018) prune edges within the beam returned by Algorithm 1, whereas (Fu et al., 2019) and (Jayaram Subramanya et al., 2019) apply pruning to the accumulated visited set.

C MQO ON TRADITIONAL DATABASES

Here, we show a further demonstration of *multiple-query optimization* upon relational databases with an extended example from Section 2.3. Table `User` stores a unique identifier `uid`, each customer’s `name` and their `age`. `SalesOrder` records individual orders with a primary key `oid`, a foreign-key `uid` that links back to `User`, and a Boolean `valid` field that is `true` when the order has not been canceled.

Listing 1: Query A		Listing 2: Query B	
SELECT	<code>U.name</code>	SELECT	COUNT(*)
FROM	<code>User U, SalesOrder S</code>	FROM	<code>User U, SalesOrder S</code>
WHERE	<code>U.uid = S.uid</code>	WHERE	<code>U.uid = S.uid</code>
AND	<code>U.age > 20</code>	AND	<code>U.age > 45</code>
AND	<code>S.valid = TRUE</code>	AND	<code>S.valid = TRUE</code>

The queries can be written down in relational algebra as below.

$$A = \Pi_{name} (\sigma_{age > 20} (U) \bowtie \sigma_{valid = \text{true}} (S))$$

$$B = \gamma_{\text{COUNT}} (\sigma_{age > 45 \wedge valid = \text{true}} (U \bowtie S)) \tag{1}$$

$$= \gamma_{\text{COUNT}} (\sigma_{age > 45} (U) \bowtie \sigma_{valid = \text{true}} (S)) \tag{2}$$

Both queries rely on the same base join $J = U \bowtie \sigma_{valid=true}(S)$. Query B applies the stricter predicate $age > 45$, which implies $age > 20$; hence its input is a *subset* of Query A's. Computing J once and sharing it therefore avoids a second full scan of `SalesOrder` and a second join.

Eq. (1) pushes both predicates into the join, while Eq. (2) postpones the age test on `User`. If users older than 45 are rare, Eq. (1) produces fewer rows and is usually cheaper to share; if filtering by age is inexpensive but reading `SalesOrder` dominates, Eq. (2) may be preferable. Cost estimates stored in catalog statistics guide the optimizer toward the better choice.

Early studies framed MQO as a shortest-path search over plan graphs (Sellis, 1988), followed by meta-heuristic techniques such as genetic algorithms (Bayir et al., 2006) and exact formulations using integer linear programming (Dokeroglu et al., 2014). Modern work augments these rules with learning-based models that predict when sharing pays off under shifting workloads. Regardless of method, the key steps are (i) detect overlapping sub-trees like J and (ii) select the reuse strategy that minimizes total runtime and I/O.

D COMPARISON OF QUERY PLANNING STRATEGIES

We compare our MST-based query planning (MST) with other possible solutions, evaluating the impact of different preprocessing methods on query search performance.

As baselines, we considered not applying MQO, k-means clustering-based planning (Clustering) and a traveling salesman problem-based planning (TSP). Since solving exact TSP is infeasible, we implemented it based on the Adaptive Large Neighborhood Search algorithm.

We used batch size of 10K at recall@1 = 0.9, and the results are shown in Table 1. To ensure a fair comparison, the preprocessing time across all strategies was unified by setting it equal to the time required for MST construction. The reported results do not include preprocessing overhead.

Table 1: Search latency (s) when applying different query planning strategies.

Dataset	Index	w/o MQO	Cluster	TSP	MST
DEEP	HNSW	0.402	0.338	0.315	0.298
	NSG	0.431	0.302	0.287	0.276
	Vamana	0.621	0.516	0.477	0.467
SIFT	HNSW	0.477	0.424	0.395	0.376
	NSG	0.402	0.333	0.306	0.299
	Vamana	0.707	0.667	0.589	0.571
T2I	HNSW	1.369	1.214	1.145	1.126
	NSG	2.012	1.678	1.592	1.573
	Vamana	2.206	1.834	1.725	1.735

As expected, the MST-based planning achieves the best performance overall as it minimizes the distance between entry point and the query compared to other methods.

E OVERALL ALGORITHM

Algorithm 4 MQO-ANNS(Q, G, ep, w, K, τ)

Require: Query batch $Q = \{q_1, \dots, q_{|Q|}\}$, graph index G , predefined entry point ep , beam size w , desired number of groups K , size threshold τ
Ensure: array $A[1 \dots |Q|]$ with the search result for every query

- 1: $A \leftarrow \text{EMPTYLIST}(|Q|)$
- 2: **if** *needRapidPreprocessing* **then**
- 3: $F \leftarrow \text{MAKEFOREST}(Q, K, \tau)$
- 4: **else**
- 5: $F \leftarrow \{\text{MST}(Q)\}$
- 6: **end if**
- 7: $Q_p \leftarrow \text{REORDER}(F)$
- 8: **for each** (q_p, q_c) **in** Q_p (left-to-right) **do**
- 9: **if** q_c is the root **then**
- 10: $A[c] \leftarrow \text{GREEDYBEAMSEARCH}(ep, q_c, w, G)$
- 11: **else**
- 12: $A[c] \leftarrow \text{GREEDYBEAMSEARCH}(A[p], q_c, w, G)$
- 13: **end if**
- 14: **end for**
- 15: **return** A

Algorithm 5 MAKEFOREST(Q, K, τ)

Require: Query batch Q , desired number of groups K , size threshold τ
Ensure: A spanning forest $F = \{T_1, \dots, T_K\}$ of Q

- 1: $F \leftarrow \emptyset$
- 2: $\{Q^{(1)}, \dots, Q^{(K)}\} \leftarrow \text{PARTITION}(Q, K)$
- 3: **for** $i \leftarrow 1$ **to** K **do**
- 4: **if** $|Q^{(i)}| \leq \tau$ **then** ▷ small group: use exact MST
- 5: $T_i \leftarrow \text{MST}(Q^{(i)})$
- 6: **else** ▷ large group: use ANNS
- 7: $H_i \leftarrow \text{BUILDDANNSGRAPH}(Q^{(i)})$
- 8: $H_i \leftarrow \text{SYMMETRIZE}(H_i)$ ▷ make the ANNS graph undirected
- 9: $T_i \leftarrow \text{MST}(H_i)$
- 10: **end if**
- 11: $F \leftarrow F \cup \{T_i\}$ ▷ add component tree to the forest
- 12: **end for**
- 13: **return** F

Algorithm 6 REORDER(F)

Require: forest $F = \{T_1, \dots, T_K\}$ ▷ T_i denotes a tree, represented by set of edges
Ensure: list Q_p of (parent, child) pairs in DFS order with parent = $\emptyset$ for each root

- 1: $Q_p \leftarrow []$
- 2: **for each** tree T **in** F **do**
- 3: $q_r \leftarrow \text{CHOOSEROOT}(T)$ ▷ pick any node as root
- 4: $\text{APPEND}(Q_p, (\emptyset, q_r))$ ▷ add edge for root
- 5: **for each** edge (q_p, q_c) **in** $\text{DFS}(T, q_r)$ **do** ▷ DFS on T starting from q_r
- 6: $\text{APPEND}(Q_p, (q_p, q_c))$
- 7: **end for**
- 8: **end for**
- 9: **return** Q_p

Algorithm 7 PARTITION(Q, K)

Require: Query batch $Q \subset \mathbb{R}^d$, desired number of groups K
Ensure: K disjoint subsets $\{Q^{(1)}, \dots, Q^{(K)}\}$ with $\bigcup_{i=1}^K Q^{(i)} = Q$

```

1:  $\mathcal{C} \leftarrow \{Q\}$  ▷ current collection of groups
2: for  $t \leftarrow 1$  to  $K - 1$  do
3:    $Q^{\max} \leftarrow \arg \max_{C \in \mathcal{C}} |C|$  ▷ pick the largest group
4:    $\mathbf{h} \sim \text{Uniform}(\mathbb{S}^{d-1})$  ▷ (d-1)-dimensional hyperplane
5:    $A \leftarrow \emptyset, B \leftarrow \emptyset$ 
6:   for all  $q \in Q^{\max}$  do ▷ split by the hyperplane
7:     if  $\mathbf{h} \cdot q \geq 0$  then
8:        $A \leftarrow A \cup \{q\}$ 
9:     else
10:       $B \leftarrow B \cup \{q\}$ 
11:     end if
12:   end for
13:    $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{Q^{\max}\}) \cup \{A, B\}$  ▷ replace  $Q^{\max}$  with its two children
14: end for
15: return  $\mathcal{C}$ 

```

F ADDITIONAL EXPERIMENTAL SETTINGS

We evaluate our MQO framework on three widely used graph-based ANNS indices: HNSW (Malkov & Yashunin, 2018), NSG (Fu et al., 2019), and Vamana (Jayaram Subramanya et al., 2019). For each index, we use the official implementation² without modifying its core indexing or search algorithms, extending only the search API to accept an external entry node.

NSG and Vamana are planar proximity graphs. Thus we simply integrate MQO by replacing their default entry points with those selected by our framework. HNSW, in contrast, employs a multi-layer hierarchy. The search begins at the topmost layer, performs a strict greedy search, and propagates the nearest-neighbor result downward through each successive layer until the bottom layer is reached. To support MQO in HNSW, we retain the original hierarchical search for the K queries using the predefined entry point (Section 3.3), and for all remaining queries we initiate search directly at the bottom layer node determined by preceding query results.

We used the following parameters to construct each proximity graph. For HNSW (Malkov & Yashunin, 2018), we adopted the configuration reported in their paper, setting $M = 16$ and $ef_{\text{construction}} = 500$. For NSG (Fu et al., 2019) and Vamana (Jayaram Subramanya et al., 2019), we used the default parameter settings provided in their official implementations. For NSG, we first constructed the EFANNA (Fu & Cai, 2016) k -NN graph using the parameters $K = 200$, $L = 200$, $\text{iter} = 10$, $S = 10$, and $R = 100$. The NSG was then built with $L = 40$, $R = 50$, and $C = 500$. For Vamana, we used $R = 64$, $L = 100$ and $\alpha = 1.2$.

All experiments were conducted on an AMD Ryzen Threadripper PRO 7985WX CPU, single core and 1024GB of main memory.

²HNSW (Malkov & Yashunin, 2018): <https://github.com/nmslib/hnswlib>

NSG (Fu et al., 2019): <https://github.com/ZJULearning/nsg>

Vamana (Jayaram Subramanya et al., 2019): <https://github.com/microsoft/DiskANN>

G MORE ON APPROXIMATION METHOD

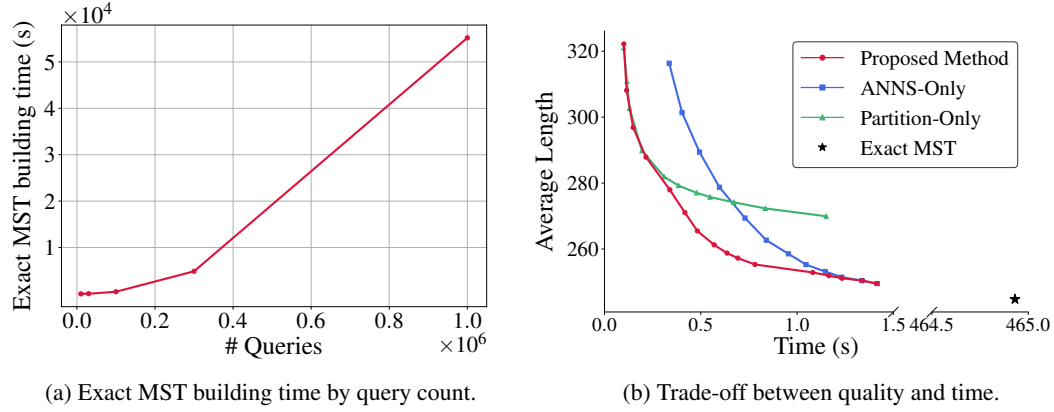


Figure 6: Necessity of our rapid query preprocessing method.

In this section, we illustrate the necessity and the details of our rapid preprocessing method introduced in Section 3.3. Figure 6a shows the time required to construct the exact minimum spanning tree for a query set from the SIFT dataset. While MST construction is feasible for small batch sizes, the quadratic time complexity quickly renders it impractical as the batch size grows. This observation motivates the need for a rapid preprocessing approach when limited preprocessing time is available.

Figure 6b demonstrates both the necessity and effectiveness of combining a partitioning strategy with an auxiliary ANNS proximity graph. Throughout this paper, we adopt HNSW as our auxiliary graph, since NSG construction requires a prior kNN graph and Vamana incurs multiple construction rounds. We plot the preprocessing time on the horizontal axis and the quality of the resulting forest—quantified by the average edge length—on the vertical axis. To account for the number of trees (groups), we compute a weighted average of the batch’s average inter-query distance and the forest’s average edge length, weighting the former by the number of trees and the latter by the batch size minus the number of trees. The curve labeled *ANNS-Only* varies ANNS parameters without partitioning, whereas *Partition-Only* varies the number of partitions without using ANNS. The curve labeled *Proposed Method* consists of Pareto-optimal points obtained by jointly employing partitioning and the auxiliary proximity graph. As the figure clearly shows, neither strategy alone suffices: integrating both yields a synergistic improvement in the quality–time trade-off.

Next, we detail the hyperparameters associated to the approximation method. At first glance, tuning the method may appear challenging due to the number of hyperparameters: the group-size threshold τ , the number of groups K , and the auxiliary ANNS (HNSW) construction parameters. While extensive tuning can yield marginal gains, we found that fixing $\tau = 500$ and setting $M = ef_{\text{construction}} = 7$ for HNSW, while varying only K , is sufficient. For K letting $K = |Q|/250$ is a good starting point, with further tuning yielding only slight improvements in overall speedup.

H MORE ON GRAPH PARAMETER EFFECT

We expand Section 4.3 with the raw measurements and the statistical tests.

Table 2: Partial association between graph parameters and residual speedup

Parameter	ρ	p
M	0.56	7.4×10^{-14}
$w_{\text{construction}}$	-0.18	0.03
α	-0.11	0.20

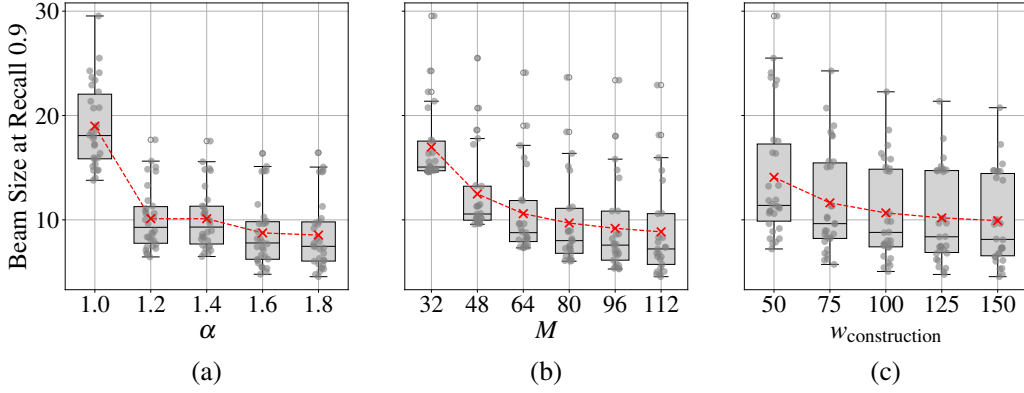


Figure 7: Required search beam versus (a) α , (b) M , and (c) $w_{\text{construction}}$ on 1M DEEP dataset. Red cross marks indicate the average value.

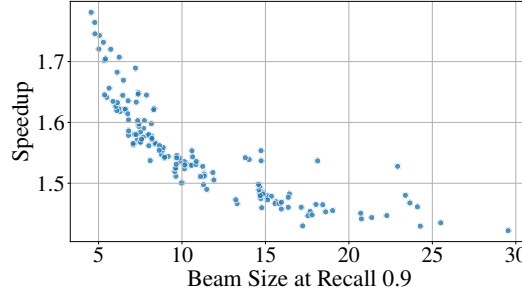


Figure 8: Speedup under varying beam size (recall 0.9).

Figure 7 shows the distribution of the beam required for 0.9 recall. All parameters continuously downscale the beam.

Figure 8 plots speedup against the required beam for every configuration. The points hug a monotonically decreasing curve, justifying to set the beam width as the primary covariate.

After regressing speedup on beam width we compute Spearman partial correlations between the residuals and each knob; results appear in Table 2. Only M , the out-degree limit is found to be positively correlated. $w_{\text{construction}}$ shows $p = 0.03$ which is marginally significant, but falls below the threshold with Bonferroni correction. α is found uninfluential other way than the search beam.

I PERFORMANCE FOR TOP-10 SEARCH

In this section, we examine the effect of our MQO framework when applied to top-10 search. Since the framework initializes the search from an entry point closer to the query, it enables the greedy beam search to converge more quickly, regardless of the final number of neighbors retrieved.

We evaluate search performance using the NSG index with a batch size of 100 K at recall@10=0.99, and include results for top-1 neighbor search for comparison. Preprocessing overhead is not included in the reported results (standard MQO). The results are presented in Table 3.

Although small differences are observed, the general tendency of the speedup is the same for top-1 and top-10 search.

Table 3: Search latency (s) for top- k search with and without MQO (preprocessing overhead excluded).

Top- k search	Dataset	Latency w/o MQO (s)	Latency w/ MQO (s)	Speedup
$k = 10$	DEEP	14.741	9.101	1.62
	SIFT	13.578	9.431	1.44
	T2I	117.364	87.856	1.34
$k = 1$	DEEP	14.650	8.998	1.63
	SIFT	13.397	9.227	1.45
	T2I	115.221	87.197	1.32

J MULTI-THREAD EXTENSION OF THE FRAMEWORK

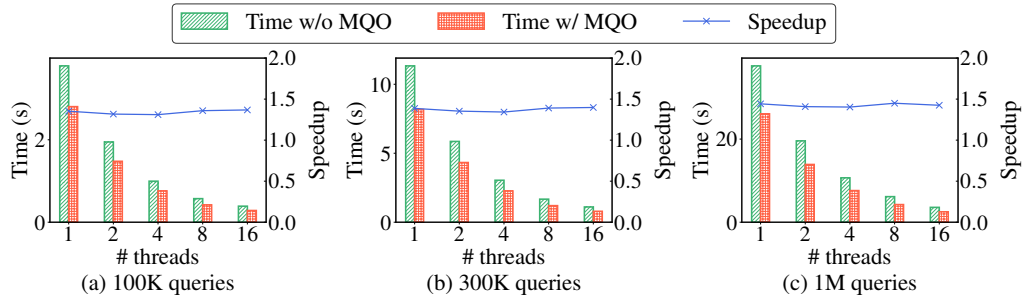


Figure 9: Comparing MQO against non-MQO baselines on multi-threaded configuration. Here MQO denotes lightweight MQO and includes both the time for preprocessing and the actual search.

To demonstrate our algorithmic advancement and computational savings, the experiments presented in the main text were conducted in a single-threaded manner. Nevertheless, we emphasize that the proposed MQO framework is inherently amenable to parallelization. Since our MQO framework partitions the query batch into K groups, both query preprocessing and nearest neighbor search can leverage inter-group parallelism by assigning a thread to each partitioned group.

We conducted a simple experiment to evaluate the scalability of our framework with respect to the number of threads. We compared our lightweight MQO with inter-group parallelism to a baseline that assigns each query to a separate thread without applying MQO. Parallelization was implemented using OpenMP (Dagum & Menon, 1998). The experiment was conducted on the DEEP dataset using the NSG graph index, targeting a recall of 0.9. We present the end-to-end time comparison in Figure 9. As shown, our lightweight MQO with inter-group parallelism exhibited constant speedup over the baseline across varying thread counts, demonstrating that our framework remains efficient under parallel execution.

In addition, we note that even when only small groups are used, or using $K = 1$ in the extreme case, intra-group parallelization remains feasible. In these scenarios, one can employ parallel algorithms for minimum spanning tree construction (Bentley, 1980) or ANNS proximity-graph construction (Manohar et al., 2024; Harwood & Drummond, 2016; Fu et al., 2019) during preprocessing. Furthermore, during the search phase, the branching degree of each node in the spanning tree over the query group presents additional avenues for parallel execution.