QBAT: Model-based Query Budget Autotuner for Clustering-based Approximate Nearest Neighbor Search

Jonghyun Bae*
Google
Mountain View, California
jonghyunbae@google.com

Tae Jun Ham
Google
Mountain View, California
taejunham@google.com

Alan Li
Google
Mountain View, California
alanwli@google.com

Supawit Chockchowwat*
Google
Mountain View, California
supawit@google.com

Yannis Papakonstantinou
Google
Mountain View, California
yannispap@google.com

ABSTRACT

Approximate nearest neighbor search (ANNS) is a critical component in modern data-intensive applications, but its performance is often hindered by the use of a static query budget parameter. This one-size-fits-all approach, even if well-tuned, fails to account for the varying difficulty of individual queries, inevitably leading to suboptimal latency on easy queries and poor accuracy on hard ones. This paper introduces QBAT, a query-aware budget autotuner designed to resolve this dilemma. By analyzing query-specific features offline, QBAT dynamically allocates an appropriate budget for each query. We explore two predictive models: a highly accurate gradient-boosted decision tree and a simple, interpretable heuristic formula derived using the AlphaEvolve framework. These models can optimize budget allocation for both system performance or recall consistency priorities. Evaluations on large-scale datasets demonstrate that QBAT reduces total searched budget by up to 68.8% in the consistency mode on ScaNN, the state-of-the-art clustering-based ANNS method, while simultaneously enforcing a strict per-query recall target, a scenario where static budgets are notoriously inefficient and wasteful.

PVLDB Reference Format:
Jonghyun Bae, Tae Jun Ham, Alan Li, Supawit Chockchowwat, and Yannis Papakonstantinou. QBAT: Model-based Query Budget Autotuner for Clustering-based Approximate Nearest Neighbor Search. PVLDB, 19(11): 3091 - 3104, 2026.
doi:10.14778/3836663.3836675

1 INTRODUCTION

The proliferation of vector embeddings has precipitated a fundamental transformation in data processing [36, 37, 44] and information retrieval [2, 3, 7, 30, 49] across numerous domains. For instance, recommendation systems identify similar items by comparing their vector representations, while natural language processing can now retrieve documents based on conceptual relevance rather than simple keyword matching. To perform vector similarity search at scale, the industry has widely adopted approximate nearest neighbor search (ANNS) algorithms [22] that trades a small, often negligible, loss in quality for substantial gains in computational efficiency. This trade-off is managed by algorithm-specific user-selected parameters, called *query budget*, such as the number of partitions to scan in clustering-based methods [11, 18, 41] or the size of the dynamic candidate list in graph-based methods [10, 19, 25, 35]. A larger budget increases the likelihood of finding the true nearest neighbors (higher recall) by searching more partitions, at the cost of higher latency and computational expense. Conversely, a smaller budget accelerates the query but increases the risk of missing relevant results from unselected partitions (lower recall).

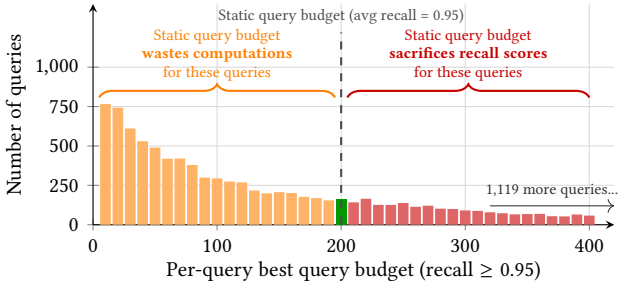


Figure 1: The distribution of optimal budgets among BigANN queries reveals a static query budget (200) wasting resources on easy queries and neglecting hard ones. QBAT aims to dynamically set query budget per query to reclaim the resources and stabilize search quality.

Core Problem. However, selecting a static query budget for all queries (i.e., query-agnostic budget), even if tuned correctly, inevitably *wastes computing resources* and *sacrifices recall scores* simultaneously, because a one-size-fits-all budget is incapable of adapting to different difficulty levels across queries (Figure 1). When the static budget is set too low, the search fails to provide sufficient recall for more difficult queries. If set high enough to satisfy these difficult queries, it wastes resources on easy queries that could have been resolved with a much smaller budget. Therefore, this one-size-fits-all approach leads to suboptimal and inconsistent search results. An empirical analysis on the BigANN dataset [1], which consists

*These authors contributed equally to this work.

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836675

of 100M vectors, reveals the severity of this issue: when using a carefully tuned fixed query budget, a staggering 28.2% of queries fail to achieve the target recall of 0.95 at 100 neighbors. Existing query difficulty estimation [20, 24] requires ground truth, limiting its use in predicting query budget during search.

Our Proposal. Therefore, we propose to select query budget dynamically per each query by predicting the query difficulty strictly based on information available at query search. Our analysis shows that query difficulty is strongly predicted by two features: standard deviation among centroid distances and distance to the nearest centroid. Intuitively, these features indicate when a query is ambiguously located between multiple data partitions. Easy queries are close to their nearest centroid, with a large standard deviation, allowing confident searching within a few partitions with a small budget. Hard queries are further from their closest centroid, with a small standard deviation, indicating proximity to multiple partition boundaries. This requires a larger search budget as true nearest neighbors are likely spread across multiple partitions.

Thanks to this observation, we introduce QBAT, a model-based query budget autotuner for vector indexes. QBAT optimizes for distinct system goals (e.g., throughput, service quality) by allocating resources based on query difficulty. Our three-stage methodology includes: (1) offline analysis to determine ideal budgets, (2) model training to predict budgets using centroid distance and standard deviation, and (3) runtime estimation for new queries. We derive and experiment with two budget predictors: a gradient boosting decision tree and an AlphaEvolve-derived heuristic. Budgets can be defined with performance mode (minimizing budget while maintaining overall recall score) or consistency mode (prioritizing recall consistency). QBAT demonstrates substantial improvements in both system efficiency and search quality over exhaustively tuned static baselines: achieving up to 34.0% lower budget (translating to 13.8% higher throughput) in performance mode and reducing the variability of recall scores by 68.7%.

Our contributions are as follows:

- We identify key features among many to measure query hardness: the normalized distance to the closest centroid and the standard deviation of distances to centroids.
- We suggest two different optimization strategies for defining the ideal budget: (1) performance mode minimizing total query budget, and (2) consistency mode ensuring per-query recall consistency.
- We introduce a high-performance gradient boosted decision tree (GBDT) model that utilizes quantile regression to provide highly accurate budget predictions, and a simple, and computationally trivial heuristic formula derived from the GBDT model's behavior using the AlphaEvolve.
- We prototype QBAT on ScaNN, a state-of-the-art clustering-based ANNS method, and evaluate it using various vector embedding datasets to demonstrate its effectiveness for maximizing overall QPS, and ensuring a consistent QoS.

2 BACKGROUND

This section provides the technical context for large-scale similarity search, establishing the shift from exact to approximate methods. Section 2.1 introduces Approximate Nearest Neighbor Search

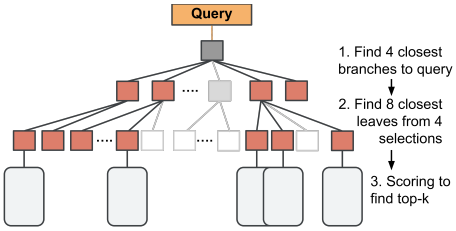


Figure 2: ScaNN searches as many leaves of its clustering tree as allowed by the query budget (num_leaves_to_search). Higher budget improves recall, but incurs more computation.

(ANNS) and its three primary algorithmic families: clustering-based, graph-based, and hashing-based methods. Section 2.3 details the ScaNN library [18, 46], explaining its use of k-means clustering and hierarchical trees for search space pruning. Finally, Section 2.3 highlights the inefficiency of static budgets, which over-search easy queries while failing to provide sufficient recall for hard ones.

2.1 Approximate Nearest Neighbor Search

The rise of large language models (LLMs) and retrieval-augmented generation (RAG) paradigm [2, 3, 7, 30, 49] has created a fundamental need for efficient similarity search mechanisms on a massive scale. While exhaustive k-nearest neighbor (k-NN) search [26, 32, 33] guarantees perfect accuracy, its linear computational complexity regarding the number of data points (N) and dimensions (d) makes it prohibitively slow for real-world applications involving millions or billions of items. Consequently, the field has converged on Approximate Nearest Neighbor Search (ANNS) [22] as a practical solution that trades absolute accuracy for orders-of-magnitude improvements in search speed. ANNS algorithms must balance between query speed, accuracy, memory usage, and index build time, requiring practitioners to select and tune algorithms whose specific trade-off profiles align with application requirements.

A Taxonomy of ANNS. ANNS algorithms can be broadly categorized based on the core strategy they employ to partition the vector space and prune the search. This paper focuses on the three primary families that are most prevalent in modern systems:

- Clustering-based [11, 18, 41]: These methods, such as the inverted file (IVF) index [11] and ScaNN [18], partition the dataset into clusters and restrict the search to a small subset of these clusters.
- Graph-based [10, 19, 25, 35]: These methods like HNSW [35] and DiskANN [25] represent data points as nodes in a proximity graph and perform the search by traversing this graph.
- Hashing-based [6, 14, 21, 23]: This family, primarily represented by locality-sensitive hashing (LSH) [23] and its variants, uses specialized hash functions to map similar vectors to the same hash buckets.

In addition to these primary indexing strategies, techniques like product quantization (PQ) [27, 28] are often used as a compression layer on top of other indexes to reduce memory footprint and accelerate distance calculations. Understanding the mechanics, strengths, and weaknesses of each of these families is essential for any practitioner building systems for large-scale similarity search.

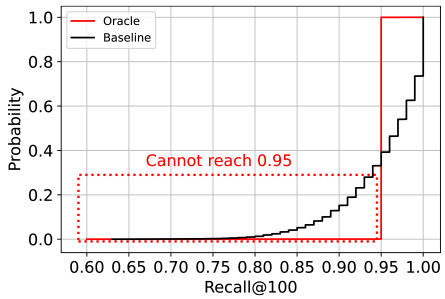


Figure 3: CDF of recall@100 using test query of BigANN. The budget value (num_leaves_to_search of ScaNN) is 200.

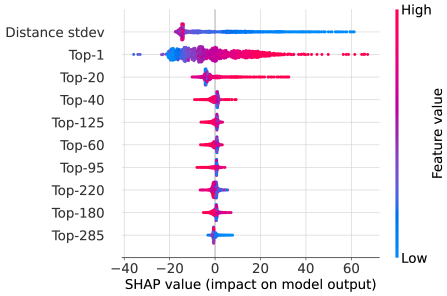


Figure 4: Top 10 important features by SHAP on BigANN.

2.2 Clustering-Based Vector Index

One of the state-of-the-art clustering-based methods is ScaNN (Scalable Nearest Neighbors) [18] library with advanced hashing and partitioning method [46]. To generate an index in ScaNN or other clustering-based approaches, the entire dataset of vectors is divided into a large number of clusters, often referred to as leaves. Each resulting partition is represented by a single centroid vector, which serves as a high-level summary for that cluster. Finally, ScaNN recursively merges lower centroids until meeting the root node, resulting in a structure called k-means clustering tree.

ScaNN Query Process. Once the index is built, the query process is executed in a multi-stage pipeline designed to rapidly prune the search space. Figure 2 illustrates the general query process of clustering-based index. The core of the pruning mechanism is controlled by the **query budget**, a.k.a. *num_leaves_to_search* parameter in ScaNN or *nprobe* in IVF index. Based on the distance between the query vector and the centroids of the partitions, the algorithm selects the target number of closest leaves. All other partitions are pruned, and no longer considered for distance calculations. Finally, the system computes the similarity scores or distances between the query vector and all data points within this limited set. After the initial scoring, ScaNN can perform an optional but highly recommended re-ranking step to refine the results.

2.3 Inefficiency of Single Query Budget

The fundamental flaw in a static budgeting approach is its failure to account for the heterogeneous nature of queries. Figure 3 illustrates the cumulative distribution function of evaluating queries using index generated by ScaNN on BigANN dataset [1] (100M vectors and 10K queries). We used a static budget of 200 leaves and found top-100 nearest neighbors of query. In this instance, an average recall score of 0.95 is achievable. Notably, a well-tuned static budget of 200 leaves still results in 28.2% of queries failing to meet the 0.95 recall@100 target, which shows a long tail of queries falling well below the desired target recall threshold. This data provides a clear, quantitative problem statement: a static query budget cannot simultaneously achieve high efficiency and consistent quality of service across a diverse workload. The system requires a more adaptive, intelligent approach to resource allocation.

The underlying reason for this phenomenon is that the appropriate query budget varies due to the differing difficulty levels of individual queries. The histogram in Figure 1 provides a visualization of query difficulty within BigANN queries. The figure reveals a crucial insight: the effort required for a successful search is not uniform across all queries. The large peak on the left side of the graph shows that a majority of queries are easy. However, the long tail extending to the right represents a minority of "hard" queries. These queries require searching a much larger number of partitions to find their true nearest neighbors. This often happens when a query vector lies near the boundary between multiple partitions, causing its nearest neighbors to be scattered across them.

This distribution highlights the core inefficiency of applying a single, static query budget to all queries in the database: *it forces a choice between sacrificing accuracy with a low budget or wasting resources with a high budget.* We need an indexing system that can dynamically adjust the search effort for each query.

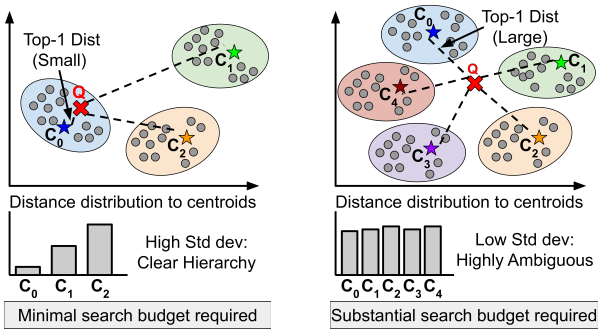
3 MOTIVATION

This section establishes the foundation for QBAT. Section 3.1 quantitatively identifies two features that predict query hardness. Section 3.2 provides a geometric interpretation of these features, explaining how they indicate whether a query is easy or hard.

3.1 Features of Query Hardness Measurement

Existing research on measuring query difficulty [20, 24, 53] commonly relies on distance-based information between the query vector and its ground truth. However, a limitation of these methods is their prerequisite knowledge of the ground truth for each query, which precludes their application in predicting the optimal query budget during the search process itself. This section undertakes an analysis of a methodology for predicting query difficulty solely utilizing information generated during the query search. To this end, we train a decision tree model that receives various types of data and predicts high-accuracy digital query budget, and then analyzed which features exert the highest influence through Shapley Additive Explanations (SHAP) value analysis [34]. SHAP value is a method used to explain the output of decision tree models, attributing the contribution of each feature to a particular prediction.

SHAP Values Analysis. Figure 4 provides a SHAP summary plot of BigANN-100M query. We trained gradient decision boosting tree (GBDT) model using LightGBM [29] as an input of SHAP value analysis. We used set of distances from queries to centroids, average distance, and standard deviation of distance distribution as an input of GBDT model to predict ideal query budget which achieves exactly



(a) Easy query case **(b) Hard query case**

Figure 5: Distance distribution intuitively indicates hardness.

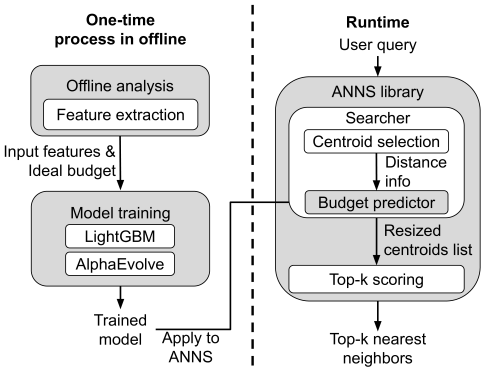


Figure 6: System architecture of QBAT.

0.95@100 recall score. The features are listed on the y-axis. "Top-N" means a distance between the query and Nth closest centroid, and "Distance stdev" is a standard deviation of the distances from the query vector to the set of index centroids. The features are ordered from top to bottom based on their global importance to the model. The x-axis represents the SHAP value. This value quantifies the impact of a feature on the model's output for a specific prediction. Positive SHAP values (to the right of zero) indicate that the feature's value for that instance pushed the model's prediction higher, or vice versa. The color of the dot encodes the actual value of that feature for that specific data point. By convention, high feature values are represented by red, and low feature values are represented by blue. This coloring is the key to understanding the directionality of a feature's effect.

The most dominant predictor is "Distance stdev". The plot shows a strong, clear pattern: blue dots, which represent low values of this feature, are concentrated almost exclusively on the right side of the plot, corresponding to positive SHAP values. It means that the GBDT model has learned a powerful, negative correlation between this feature and the model's output. The "Top-1" feature, meaning the distance to the closest centroid, is ranked as the second most important. Its pattern is the inverse of "Distance stdev": red dots (high feature values) are associated with positive SHAP values, while blue dots (low feature values) are associated with negative SHAP values. Therefore, the model has learned a strong, positive correlation. The features "Top-20", "Top-40", "Top-60", and other "Top-N" features, show a clear pattern of progressively decreasing importance as they move down the y-axis. Furthermore, the horizontal spread of their SHAP values becomes tighter and more centered around zero as "N" increases.

A Remark on Interpretability. While SHAP is a prominent framework for local interpretability, it may exhibit interpretability limitations, particularly in domains with highly correlated or dependent features [40]. We utilize SHAP primarily as a feature selection and hypothesis-generation tool. The definitive validation of these features is established not by the SHAP values themselves, but by the downstream success of the simple heuristic model described in Section 4, which uses only these two features and demonstrates strong predictive performance without the complexity of incorporating many features via machine learning.

3.2 Intuition Behind Distance Features

The *top-1 closest centroid distance* serves as a proxy for query centeredness. When this distance is small, the query lies deep within a cluster, far from decision boundaries where true nearest neighbors are likely lost to neighboring partitions. Conversely, a large distance indicates the query is positioned near the periphery, suggesting its neighbors may be scattered across adjacent partitions. Complementing this, the *standard deviation of centroid distances* quantifies structural ambiguity. A high standard deviation implies a clear hierarchy among centroids, allowing the algorithm to prune the search space with high confidence. In contrast, a low standard deviation indicates the query is nearly equidistant from multiple centroids, signifying that it resides in a dense region where boundaries converge and pruning is risky.

Example of Query Hardness Identification. Figure 5 visualizes this intuition. An easy query (Figure 5a) is close to its nearest centroid and has a large standard deviation among centroid distances. This creates an unambiguous search space where a small query budget is sufficient to find neighbors with high recall. Conversely, a hard query (Figure 5b) is far from its closest centroid and has a small standard deviation, indicating a dense boundary region. To achieve high recall in such cases, the algorithm cannot confidently prune partitions and must inspect a larger number of them. Together, these features enable a query-aware autotuner to distinguish between easy queries that require only a few partition inspections and hard queries that demand a substantial search budget.

4 QBAT DESIGN

Based on the intuitions, we propose QBAT, a dynamically adjusting query budget autotuner. Figure 6 shows the overall methodology of QBAT. Our methodology is model-based and comprises three stages. Initially, we conduct an offline analysis of query data to ascertain the ideal search budget for each query. Subsequently, a model is trained to predict this target budget utilizing two predictive features: the distance to the top-1 closest centroid and the standard deviation of distances. Finally, at runtime, this trained model estimates a customized budget for each new query. We have investigated two distinct implementation solutions: a sophisticated machine learning model employing a gradient boosting decision

Algorithm 1 Pseudocode of target budget selection for performance mode using greedy algorithm

```
Input: NumQ          // Number of all query vectors
Input: RecallTbl     // An offline profiled budget-recall data table
Input: TgtRecall     // Target average recall (e.g., 0.95)
Output: FinalBdgts   // List containing the optimal budgets
1:  MaxLeaves = length(RecallTbl[0])
2:  DP = Array[NumQ][MaxLeaves + 1] initialized to inf
3:  Trace = Array[NumQ][MaxLeaves + 1]
4:  for i in range(NumQ) do
5:      for j in range(MaxLeaves) do
6:          DP[i][j] = DP[i-1][j]
7:          for k in range(length(RecallTbl[i]) do
8:              DP[i][j] = Max(DP[i][j], DP[i][j - k] + RecallTbl[i][k])
9:              Trace[i][j] = k if DP[i][j] updated
10:         end for
11:     end for
12: end for
13: Opt_j = min Opt, s.t. DP[NumQ-1][Opt] / NumQ ≥ TgtRecall
14: CurBudget = DP[NumQ-1][Opt_j]
15: for i from NumQ-1 to 0 do
16:     FinalBdgts[i] = Trace[i][CurBudget]
17:     CurBudget = CurBudget - Trace[i][CurBudget]
18: end for
19: return FinalBdgts
```

tree, and a more interpretable heuristic formula model derived using AlphaEvolve [42]. Additionally, the target search budget can be defined in two ways: a consistency mode, which prioritizes achieving a specific recall for every query, and a performance mode, which optimizes for the best recall per number of leaves searched.

4.1 Defining the Target Budgeting Modes

Before a predictive model can be trained, a ground truth or ideal target budget must be defined for each query in the training dataset. We examine two distinct target search definitions, each optimized for different objectives, and explores their corresponding ideal behaviors in query processing and information retrieval. The first definition, the performance mode, prioritizes expediting query execution and maximizing system throughput. In stark contrast, the consistency mode prioritizes a consistent and uniformly high level of accuracy for every individual query. This choice is not merely technical but represents a core business decision, allowing system owners to align database performance with strategic goals, whether they be maximizing hardware efficiency or guaranteeing a consistent user experience.

Performance Mode. The performance mode aims to minimize the total query budget searched across all queries while maintaining a specified target average recall score. The performance mode focuses on maximizing system throughput (QPS) by maintaining a low total query budget allocation. While this maximizes computational efficiency, it comes at the expense of consistent Quality of Service (QoS). In this mode, the recall of difficult queries could be intentionally compromised to preserve overall system speed, making it unsuitable for applications that demand uncompromisable accuracy for every individual query.

To establish the ground truth for this strategy, we employ a optimization algorithm with dynamic programming. The procedure is formally described in Algorithm 1. The algorithm accepts three inputs: the total number of query vectors ($NumQ$), a pre-computed recall table ($RecallTbl$) where $RecallTbl(qid, k)$ provides the recall score for a specific query given k searched leaves, and the target average recall ($TgtRecall$).

The core is performed via a nested iterative process. The outer loops iterate through each query index i and each potential total budget capacity j. The state $DP[i][j]$ is defined as the maximum cumulative recall achievable using the first i queries within a strict total budget j (Line 2), while a $Trace$ table records local optimal choices (Line 3). The algorithm proceeds via a nested iteration over queries and budget capacities. For each state (i, j), the algorithm evaluates assigning k leaves to the current query. It updates $DP[i][j]$ if combining the current query's recall ($RecallTbl[i][k]$) with the optimal remaining budget ($DP[i][j-k]$) exceeds the previously known maximum (Line 8-10). This exhaustively explores substructures to maximize global recall for every budget constraint. After that, the optimal total budget Opt_j is determined by scanning the final DP state ($DP[NumQ-1]$) for the minimum cost where the average recall satisfies $TgtRecall$ (Line 14). The final budget vector ($FinalBdgts$) is then reconstructed via backtracking: starting from Opt_j, the algorithm retrieves the selected leaf count from Trace, assigns it, and decrements the remaining budget iteratively until all queries are processed (Line 15-19).

Consistency Mode. The consistency mode establishes a stringent accuracy target by stipulating that every individual query achieves a specified recall score, such as 0.95. Unlike performance mode, this identifies the ideal budget for each query by determining the specific number of leaf explorations required to reach that exact target recall, ensuring that even the most challenging queries are fully satisfied.

The consistency mode system inherently exhibits high accuracy in responding to individual queries. Its robust accuracy ensures that for each specific information request, the system consistently delivers reliable and correct results, fostering user trust and satisfaction. However, this commitment to uncompromised accuracy introduces a trade-off: achieving the desired recall score for certain complex or rare queries can be computationally intensive. Difficult queries may necessitate significantly more leaf explorations to meet the stringent recall target. This increased computational demand for complex queries can, in turn, impact the overall number of leaves to search of the system, potentially leading to lower throughput compared to a performance mode model during periods of high query complexity. Therefore, the choice between these two modes heavily depends on the specific application's priorities.

4.2 Input Features and Profiling

To predict the query budget, we utilized the two features mentioned in Section 3, which are capable of determining the query's difficulty. Prior to the extraction of these input features, QBAT first calculates the distances between the query and the N nearest centroids.

(1) **Top-1 closest centroid distance** $D_1(q)$: The distance from the query to the nearest of the N selected centroids. Rather than

using the raw distance value, this feature is normalized by the average distance from the query to all N centroids.

(2) **Standard deviation** $\sigma(q)$: The standard deviation is calculated from the distribution of distances for a specific subset of the N selected centroids. First, the centroids are sorted in ascending order of their distance to the query. The subset is then determined by the ranks corresponding to the p20 and p80 percentiles of the ideal target budget distribution. For example, if the query budget values at the p20 and p80 percentiles of the ideal budget are 10 and 50, respectively, the standard deviation is computed using the distances of the centroids ranked from the 10th to the 50th position. This range is chosen to capture the variance among the bulk of the query distribution while excluding extreme outliers at either end, leading to a more stable and robust feature for prediction.

Preserved Data Privacy. A key advantage of this feature extraction process is its minimal data requirement and inherent privacy preservation. QBAT operates only on *distances* between the query embedding and the coarse quantization centroids. Since these distances are purely relative and do not expose the raw coordinates of the embeddings or the underlying content, the privacy profile of QBAT is at least as secure as that of the base vector index itself, as it uses no new information.

Overhead of Calculating Input Features. For instance, consider the BigANN dataset. A static approach could achieve an average recall@100 of 0.95 by consistently selecting and searching the 44 nearest centroids. QBAT, in contrast, operates through a dynamic, multi-stage process. It begins by selecting an initial candidate set of 210 centroids, a query budget sufficient to satisfy the ideal target for 99% of all queries. Based on the distance information to these 210 centroids, the system extracts the distance to the top-1 closest centroid and calculates the standard deviation of these centroid distances. This information is then used as input for the model, which predicts a new, optimized query budget for the actual top-k vector search. While it is possible for the predicted budget to exceed the initial 210 candidates, necessitating additional distance calculations and sorting to expand the candidate set, this potential overhead is minimal. The computational cost of this step constitutes a small fraction of the total query time, and the efficiency gains from reducing the number of partitions to be searched by QBAT are substantially greater.

Profiling for Training Set. To generate the training data for the predictive models, QBAT performs a preliminary profiling phase on query samples. During this phase, the system extracts the input features described above and records the actual recall achieved across a range of query budgets (i.e., budget-recall data table). Crucially, this analysis does not necessitate the full dataset or a massive workload; a representative sample of a few thousand queries (e.g., at most 2,000 in our evaluation) is sufficient to capture the distribution of query difficulty and train robust models that generalizes effectively to the remaining workload, discussed in Sections 4.3 and 4.4.

4.3 Accurate Budget Prediction: QBAT (ML)

The first and more complex predictive model is a Gradient-Boosted Decision Tree (GBDT). GBDT is a powerful ensemble learning technique that constructs a highly accurate prediction model by sequentially adding simple decision trees. A critical technical detail of the GBDT implementation is its use of quantile regression rather than standard least-squares regression. Instead of predicting the conditional mean of the target budget, quantile regression predicts a specific conditional quantile. The model is designed such that the "estimation > target_budget for 50+% of the cases". This is a precise description of median regression (i.e., 50th percentile or α=0.5 quantile regression). By predicting the median, the model provides a robust estimate that is less sensitive to extreme outliers in the training data. This approach ensures that the predicted budget is sufficient for at least half of the queries with similar feature characteristics, providing a balanced estimate. By choosing a quantile slightly higher than 0.5, a conservative safety margin could also be introduced. For this work, we trained our GBDT model using LightGBM [29]. The GBDT model's training completes in up to 700 ms when training 8,000 queries (BigANN), a duration that is negligible within the context of the entire query process. This swift training time ensures that the model's computational overhead has an insignificant impact on the overall end-to-end latency.

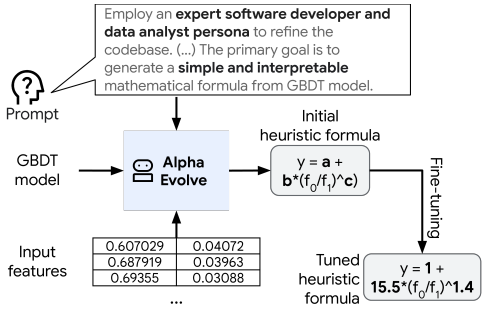


Figure 7: Process of evolving heuristic model by AlphaEvolve.

4.4 Interpretable Budget Prediction: QBAT (HR)

While GBDT models are highly accurate, they are often considered "black boxes" because their internal decision-making logic is *opaque* and *difficult to interpret*. In a production database environment, this complexity introduces operational challenges over reliability, observability, and maintenance. A simpler model, even if slightly less accurate, is often preferable for production.

Toward this end goal, a second model is developed using the AlphaEvolve program synthesis framework. The objective of this process is to generate a simple, interpretable formula based on our query hardness analysis. The workflow is as follows: (1) The trained GBDT model serves as an oracle. AlphaEvolve is prompted to find a simple symbolic expression that best mimics the input-output behavior of this oracle based on the top-1 closest centroid distance $f_0 = D_1(q)$ and the standard deviation $f_1 = \sigma(q)$. (2) AlphaEvolve explores a space of mathematical functions and evolves a formula that abstracts the GBDT's predictions:

$$\hat{B}(q) = a + b \times \left(\frac{D_1(q)}{\sigma(q)} \right)^c \tag{1}$$

The primary value of this heuristic model lies not in its absolute accuracy but in its operational simplicity. It has *effectively zero inference latency*, can be *easily implemented* in the database's query

execution engine without external dependencies, and its behavior is *easy to reason about*. In many production scenarios, these advantages can make it a more practical choice than a more complex machine learning model.

Retrospective Interpretation. This synthesized ratio $\frac{D_1(q)}{\sigma(q)}$ roughly captures the geometric reality of searching cluster-based indices like K-means by approximately decomposing the required search budget (i.e., the number of clusters to search) into two factors: (1) the *size of the required search region* multiplied by (2) the *density of clusters* within that region. First, to achieve a target recall, a query q must explore an area whose size scales with the distance to the closest centroid, $O(D_1(q)^c)$, with exponent c to adjust for a fixed vector dimension. Second, K-means naturally packs centroids tighter in denser areas. The local density of these alternative clusters is inversely proportional to the spread of their distances, $O(1/\sigma(q)^c)$. By multiplying these two factors, the required number of clusters to search is proportional to $O\left(\left[\frac{D_1(q)}{\sigma(q)}\right]^c\right)$. Here the leading coefficients and exponents are empirically fitted, representing factors specific to a given index, dataset, and target recall.

This resulting ratio acts as a practical measure of "cluster ambiguity." When $D_1(q)$ is small and $\sigma(q)$ is large, the query is safely inside a single cluster, requiring a minimal search budget. Conversely, while extremely rare, an edge case occurs if $\sigma(q) \to 0$; here, the query is perfectly equidistant from all candidate centroids which may equally likely hold the true neighbors. In this scenario, the ratio $\frac{D_1(q)}{\sigma(q)} \to \infty$, prompting QBAT to search the entire list of initial candidate clusters (e.g., 210 clusters in our evaluation).

4.5 Discussions

Extension for Incremental Indexing. A key property of QBAT is that its predicted budget $\hat{B}(q)$ for any query q remains invariant under such updates, as the input features depend solely on distances to the fixed centroids. If the new points are drawn from the same underlying distribution (stationary update), they are naturally distributed among clusters without altering relative densities. Therefore, if a predicted budget $\hat{B}(q)$ produces recall $R(q, \hat{B}(q))$ on the original index, the recall after the update, $R'(q, \hat{B}(q))$, will be approximately equal to $R(q, \hat{B}(q))$, maintaining both search efficiency and accuracy. On the other hand, the updates that introduce a distribution shift (non-stationary update) may subsequently degrade overall throughput and/or recall, even with a static budget. Existing works handle this problem by either restructuring the index (e.g., splitting or merging imbalanced clusters) [48] or rebuilding the index altogether [5, 38, 45]. As part of these maintenance operations or when a measured drift metric exceeds a certain threshold, QBAT can simply profile a small set of query samples and retrain its lightweight model, incurring only a small fraction of the cost of either restructuring or rebuilding the index itself.

Case of Filtered Vector Search. Although filtered search reduces the initial candidate set using metadata, it does not resolve the fundamental challenge of heterogeneous query difficulty within the resulting subset. Query hardness remains a function of local data geometry and a query's proximity to partition boundaries. Therefore, QBAT 's core features—distance to the nearest centroid and distance standard deviation—remain essential for identifying queries in ambiguous regions where neighbors are scattered across multiple clusters. Since static budgets are inherently inefficient regardless of the candidate pool size, QBAT could serve as an orthogonal optimization layer.

Table 1: Dataset characteristics

Dataset	# of vectors / queries	Vector dimension	Dataset size	Target framework
BigANN [1]	10M / 10K	128	4.7 GB	ScaNN, IVF
	100M / 10K	128	47.6 GB	AlloyDB
Cohere [13]	10M / 1K	768	28.6 GB	ScaNN, IVF, AlloyDB
OpenAI [43]	5M / 1K	1,536	28.6 GB	ScaNN, IVF, AlloyDB
T2I [4]	10M / 10K	200	7.6 GB	ScaNN, IVF
	100M / 10K	200	74.5 GB	AlloyDB

Table 2: Baseline indexing parameters and index size

	ScaNN params (num_leaves, avq, $soar$)	Faiss params ($nlist$, $nbits$)	ScaNN Index size	Faiss Index size
BigANN	20K, 2.5, 1.5	20K, 8	5.5 GB	4.9GB
Cohere	20K, 2.5, 1.5	20K, 8	28.6 GB	29GB
OpenAI	10K, 2.5, 1.5	10K, 8	28.6 GB	29GB
T2I	20K, 2.5, 1.5	20K, 8	7.6 GB	7.6GB

Limitations and Future Directions. Future research may explore the following directions to build upon the QBAT framework. First, extending the methodology to graph-based index structures like HNSW revealed significant potential, yet left a much room for improvement arising from challenges modeling query budget from graphical features and defining just-right granularity of "budget" (Section 5.9. Second, given QBAT's demonstrated data efficiency (requiring only 5% to 10% of queries for training), developing online learning and adaptation techniques for incremental indexing scenarios appears highly impactful. This would allow the system to continuously adapt to evolving data distributions without requiring full offline retraining. Finally, exploring hardware-aware query budgeting, particularly for modern hardware accelerators like GPUs where batching and memory access patterns differ fundamentally from CPUs, could unlock further performance gains in large-scale vector search systems.

5 EVALUATION

Setup. We conduct our experimental evaluation on a server with Intel Xeon CPU @ 2.60GHz (96 cores), 256GB main memory, and 1024GB SSD (when applicable). All algorithms are implemented in C/C++, embedded in the ScaNN [18] and Faiss inverted file index with product quantization (Faiss-IVF) [11], with AVX2 support. Our GBDT model of QBAT (ML) is trained using the LightGBM [29] library. All queries are executed using single-threaded execution. Each query is processed sequentially on a dedicated core to establish the pure latency for a single unit of work.

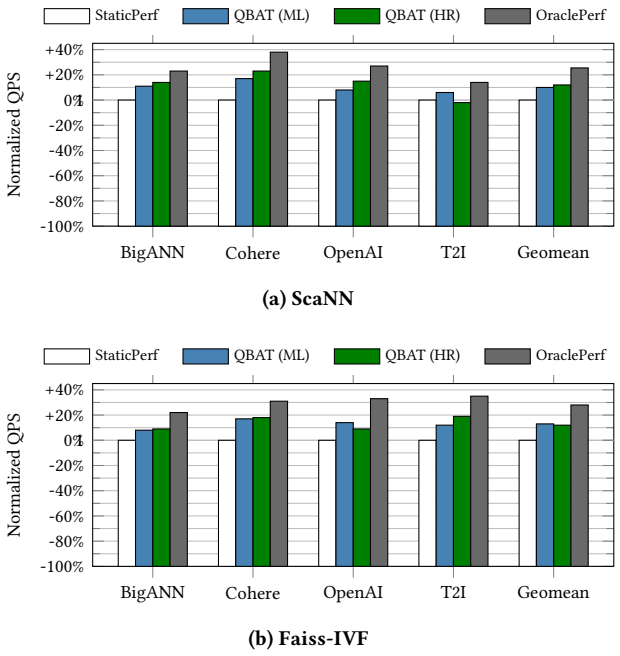


Figure 8: Normalized throughput (QPS) on performance mode across two indexes: (a) ScaNN (b) Faiss-IVF.

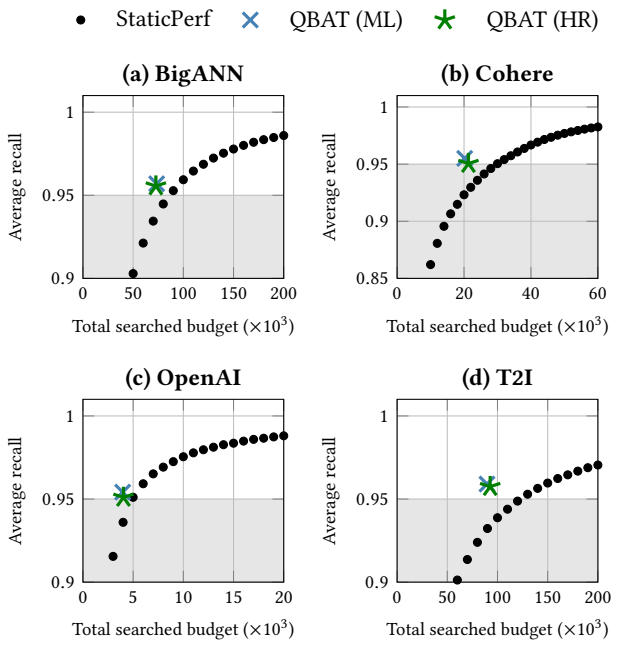


Figure 9: Budget-Recall curve on performance mode (ScaNN) shows QBAT consistently reduces searched budget while maintaining average recall above the target score (0.95).

Dataset. Table 1 summarizes the datasets. We evaluated performance using four public datasets: BigANN [1], Cohere [13], OpenAI [43], and Text2Image (T2I) [4]. These cover diverse real-world scenarios with 5M to 100M vectors and dimensions ranging from 128 to 1,536. Specifically, BigANN consists of SIFT image descriptors, Cohere and OpenAI represent high-dimensional text embeddings generated by modern language models, and T2I contains cross-modal image-text embeddings. This selection spans a wide range of data distributions, modalities (visual, textual, and cross-modal), and embedding techniques, providing a comprehensive coverage on production workloads. For enterprise-scale comparisons, we utilized the 100M-vector versions of BigANN and T2I. Training and testing workloads were randomly sampled from each dataset's query repository, using an 80/20 split for GBDT model training and evaluation. We establish plain ScaNN and Faiss-IVF indexes as baselines. ScaNN indexing employed asymmetric hashing, AVQ partitioning [18], and SOAR [46], while Faiss-IVF utilized 8-bit subvector quantization. Table 2 shows detailed parameters for indexing and generated index size. All indexes are fully memory-resident during evaluation (except in Section 5.5).

Budgeting Baselines. To evaluate the effectiveness of our query-aware budget tuning, we establish baselines and oracles based on the two target budgeting modes. (1) *StaticPerf* applies a constant search budget to all queries. It is trained for performance mode by selecting the minimum fixed budget that achieves an average recall meeting the target recall across the training set. (2) *StaticConst* similarly applies a constant search budget but is trained for consistency mode by selecting the budget that minimizes the mean absolute error between the achieved recall and the target recall across the training set. (3) *OraclePerf* represents the upper bound of dynamic budgets for performance mode. It is trained by memorizing the per-query optimal budgets from Algorithm 1. During query time, it extracts features from the query, performs a table lookup, and returns the memorized budget. (4) *OracleConst* similarly represents the upper bound of dynamic budgets for consistency mode. It is trained by memorizing the per-query minimum budgets that results in per-query recalls equal to the target recall. (5) *QBAT (ML)* and (6) *QBAT (HR)* refer to QBAT with models in Sections 4.3 and 4.4.

5.1 Performance Mode Comparison

Figure 8 presents the normalized search throughput (QPS) for the performance mode, where higher values indicate better performance. This mode targets an average recall of 0.95 at $k = 100$. For ScaNN, QBAT (ML) and QBAT (HR) improve throughput by 11% and 17% respectively, nearing OraclePerf's 26% improvement. In the Faiss-IVF scenario (Figure 8b), QBAT (ML) achieves up to 17% improvement while QBAT (HR) provides 19%. These results confirm that both machine-learning and heuristic models deliver significant performance enhancements even when optimizing solely for raw speed.

The budget-recall curves across the BigANN, Cohere, OpenAI, and T2I datasets demonstrate that the QBAT identifies lower optimal costs compared to the StaticPerf. As illustrated in Figure 9, the QBAT (ML) and QBAT (HR) models consistently achieve the average recall target of 0.95 with a smaller total search budget than the StaticPerf, which must search a higher number of leaves to reach the same quality threshold. For instance, in the BigANN dataset,

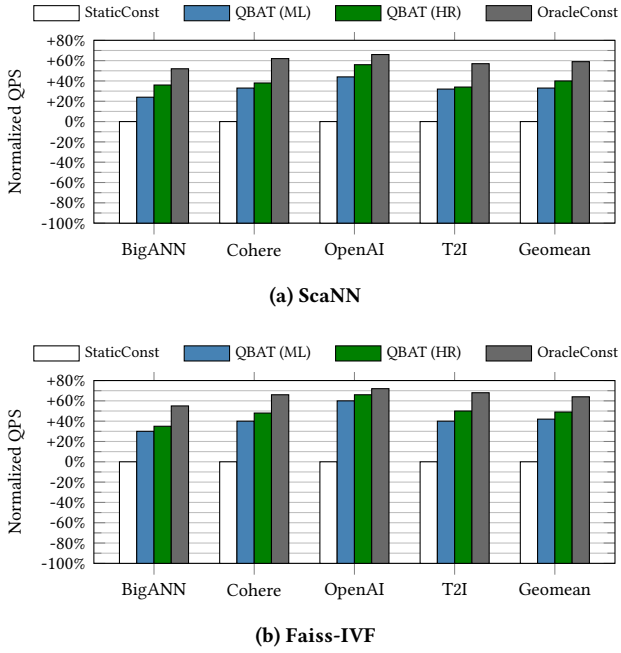


Figure 10: Normalized throughput (QPS) on consistency mode across two indexes: (a) ScaNN (b) Faiss-IVF.

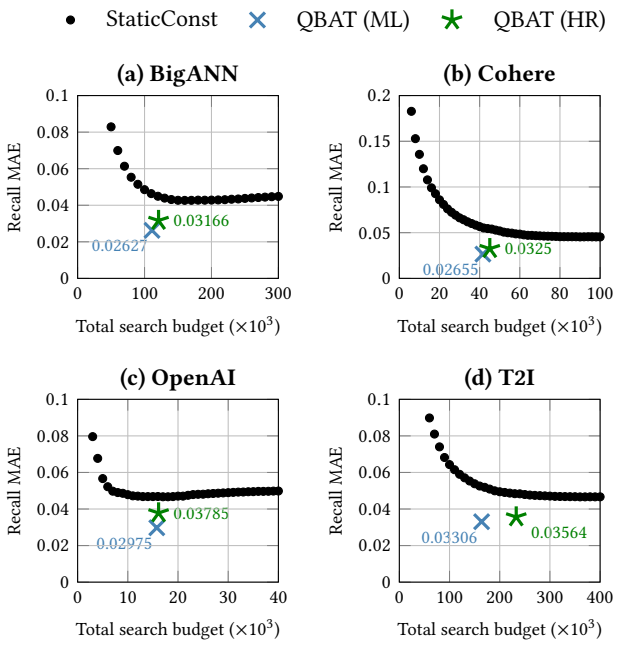


Figure 11: Budget-MAE curve on consistency mode (ScaNN) shows QBAT stabilizes recall scores among queries around the target score (0.95) better than any baseline could.

QBAT reaches the target recall at a budget where the baseline's performance is still below 0.95. This efficiency stems from QBAT's ability to dynamically allocate appropriate budgets by identifying query hardness through features like distance standard deviation and proximity to the nearest centroid.

5.2 Consistency Mode Comparison

The consistency mode evaluates throughput improvement while ensuring that all queries achieve 0.95 recall@100. As depicted in Figure 10a, QBAT (ML) and QBAT (HR) achieve average throughput improvements of 33% and 40% on geometric mean, respectively, when compared to StaticConst. Unlike StaticConst, QBAT eliminates unnecessary cluster exploration by allocating a small budget to easy queries instead of a uniformly large one. The performance benefits of the proposed methods are more pronounced in the consistency mode scenario, where the system must meet a specific search quality target. As shown in Figure 10b, under this constraint, the QBAT (ML) and QBAT (HR) shows substantial 42% and 49% throughput improvements. The results clearly support the argument that both the QBAT (ML) and QBAT (HR) successfully improve throughput compared to the well-tuned baseline.

The budget-MAE curves in Figure 11 demonstrate that QBAT establishes a new lower optimal stability for the consistency mode by minimizing the mean absolute error (MAE) relative to the target recall score of 0.95 across all evaluated datasets. While StaticConst's MAE remains relatively high and plateaus even as the search budget increases, both the QBAT (ML) and QBAT (HR) reside on a much lower error plane, indicating a consistent ability to meet the strict 0.95 per-query recall target. This improved stability is a direct result

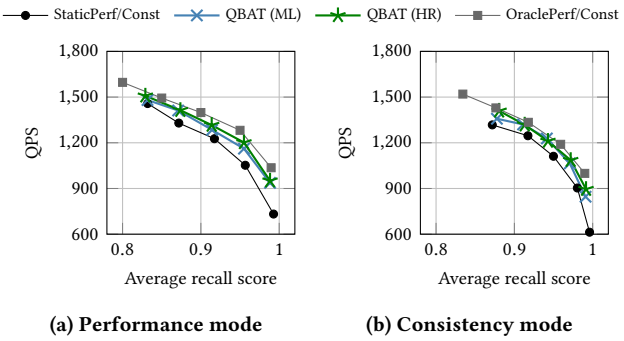
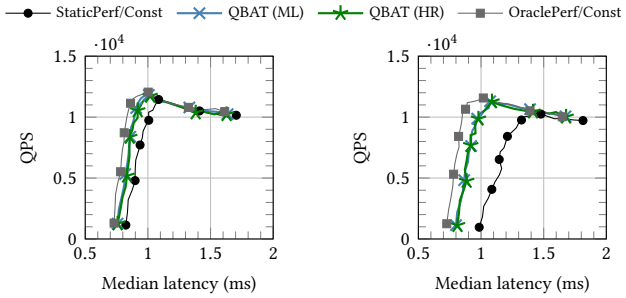


Figure 12: Recall-QPS curve on BigANN across two modes.

of QBAT 's dynamic resource allocation, which accurately identifies hard queries requiring larger budgets while preventing the oversearching of easy queries that typically inflates total costs in static systems.

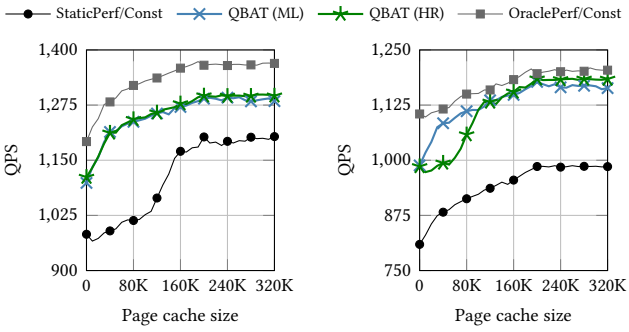
5.3 Trade-off Between Target Recall and QPS

To evaluate the effectiveness of QBAT across a spectrum of quality requirements, we measure the search throughput (QPS) as a function of the achieved recall on the BigANN dataset. We vary the target recall setting across $\{0.8, 0.85, 0.9, 0.95, 0.99\}$, which corresponds to the five markers on each curve from left to right in Figure 12. In both performance mode and consistency mode, the QBAT methods (ML and HR) consistently outperform the Static budget baseline, shifting the curve upward and to the right. This indicates



(a) Performance mode (b) Consistency mode

Figure 13: QBAT outperforms static baselines on throughput-latency curves with 1, 2, ..., 25 threads (top-left is better).



(a) Performance mode (b) Consistency mode

Figure 14: Dynamic query budgeting also improves QPS of indexes stored on disk, across all page cache sizes.

that for a given recall target, QBAT achieves higher throughput, and conversely, for a given throughput, it achieves higher recall. The advantage remains significant across recall targets, demonstrating that the learned budget prediction generalizes well beyond a single fixed operating point.

5.4 Multi-Threaded Scalability

We evaluate the multi-threaded throughput and median latency performance of ScaNN when enhanced with QBAT over 1, 2, . . . , 25 threads. The results, shown in Figure 13, demonstrate that QBAT maintains its advantage in multi-threaded environments: higher throughput at equivalent or lower latencies than StaticPerf and StaticConst, especially at a lower number of threads (1–16). As more threads contend for resource, all methods converge to similar performance in performance mode but QBAT and OracleConst remain slightly better than StaticConst. In both modes, QBAT performs similarly to OraclePerf and OracleConst with a slightly larger room for improvement in consistency mode.

5.5 Evaluation on External Storage

To evaluate cluster-based indexes on very large datasets, we evaluate the impact of QBAT on a system where the index resides fully on disk (SSD) and only partially on limited memory. We applied the principles of systems like DiskANN [25] to implement an on-disk version of ScaNN with a page cache to mimic limited memory where each page stores the full-precision vectors of database points belonging to specific IVF partitions. We sweep the page cache size and measure the search throughput (QPS).

The results, shown in Figure 14, demonstrate the impact of QBAT in non-memory resident scenarios. In both modes, the QBAT methods consistently achieve higher QPS than the static budget baseline across all evaluated page cache sizes. The advantage is particularly noticeable at smaller cache sizes, where the reduction in query budget directly translates to fewer disk reads and significantly better overall throughput. As the cache size increases and more data fits in memory, the system performance improves for all methods, but QBAT maintains its margin similarly to earlier experiments.

Table 3: Recall, total searched leaves (query budget), and QPS by including or reducing features for BigANN on ScaNN with performance mode.

	Recall	Total query budget	QPS
QBAT (ML)	0.956	73,431	1382.14
+ Top-2 to 10 distances	0.957	73,911	1371.44
+ p25, p50, p75 closest dist	0.957	73,970	1383.71
- Standard deviation	0.935	85,543	1295.85
- Top-1 closest dist	0.931	86,896	1281.23

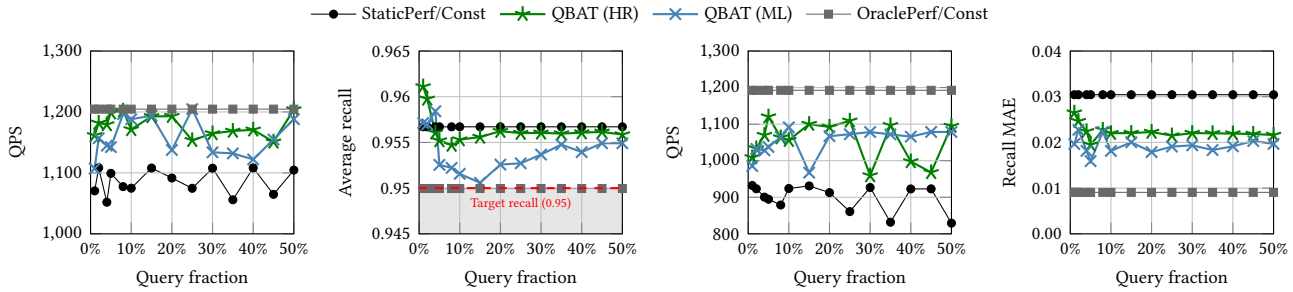
5.6 Ablation Study: Varying Input Features

Table 3 presents an ablation study of an QBAT (ML), validating the importance of its selected features. This study analyzes how performance metrics—recall score, total number of searched query budgets, and QPS—change with the addition or removal of features from the QBAT (ML), using the BigANN dataset in a performance mode scenario on ScaNN. The results indicate that adding more features does not enhance the model's effectiveness, supporting the lean design of the baseline. For instance, including the top-2 to 10 distances shows no improvement, with recall at 0.957 and a slightly lower QPS of 1371.44. Similarly, incorporating the p25, p50, and p75 closest distances to query yields nearly identical performance to the baseline, with a QPS of 1383.71.

Conversely, removing features degrades performance. Without the standard deviation from the baseline, the model's recall dropped notably to 0.935, and the required query budget increased to 85,543. Performance deteriorated even further without the top-1 closest distance, with recall falling to 0.931 and QPS decreasing to 1281.23. In conclusion, this ablation study confirms that the two features in the baseline model are both necessary and sufficient. Removing either feature severely harms performance, while adding more complex features provides no discernible benefit.

5.7 Ablation Study: Varying Training Data Size

To understand the data requirements and practical overhead of QBAT's offline profiling phase, we evaluate the performance of the learned models when trained on varying fractions of the query



(a) QPS (Perf.) (b) Recall (Perf.) (c) QPS (Const.) (d) MAE Recall (Const.)

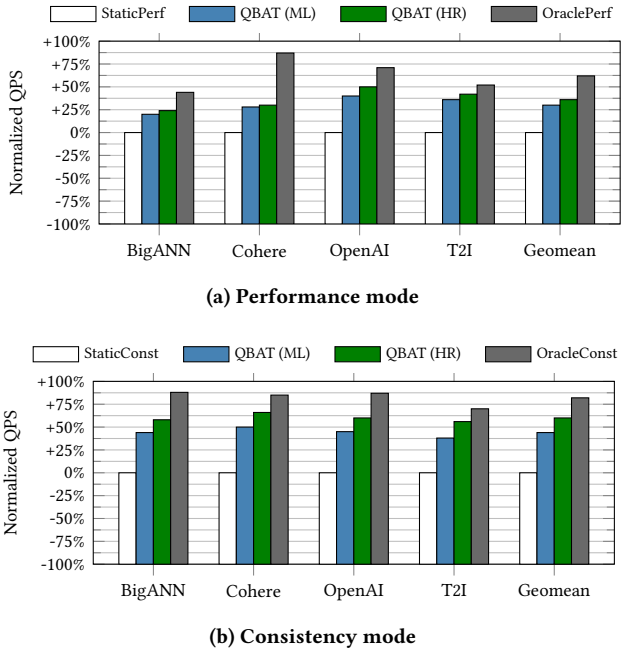
Figure 15: Effect of training sample size on QPS and quality across performance mode (Perf.) and consistency mode (Const.).

workload, ranging from 1% to 50%. This analysis helps determine the minimum amount of data needed to learn effective budget prediction models. As shown in Figure 15, QBAT is remarkably data-efficient. Even at small query fractions of 5% to 10%, both QBAT (ML) and QBAT (HR) deliver substantial QPS improvements over the static baselines while maintaining the target recall in performance mode (Figure 15b) and minimizing recall error in consistency mode (Figure 15d). The performance quickly stabilizes and approaches the Oracle, suggesting that a large training set is not necessary. This low data requirement significantly reduces the offline profiling overhead and confirms that QBAT can operate effectively with a small representative sample of the workload.

5.8 QBAT on Alloy DB

We applied QBAT to AlloyDB Omni [16, 17], which now supports ScaNN for ANNS service, to evaluate its effectiveness in a real-world, industry-focused database. Figure 16 shows the normalized total searched leaves for the QBAT (ML) and QBAT (HR) compared to a baseline and a theoretical oracle. In the performance mode scenario, the QBAT (HR) reduces total searched leaves by an average of 38.8% over the baseline, according to the geometric mean, and the QBAT (ML) also provides a substantial boost, reducing the average total searched leaves by 44.7%. The performance advantages of the proposed methods are even more pronounced in the consistency mode scenario, which reflects a more practical use case where search accuracy is critical. The QBAT (ML) and QBAT (HR) deliver an average total searched leaves reduction of 68.8% and 65.9% on average over the baseline. The data supports the conclusion that both the QBAT (ML) and QBAT (HR) improve cost reduction on realistic Vector DB system.

We applied QBAT to AlloyDB Omni [16, 17], which now supports ScaNN for ANNS service, to evaluate its effectiveness in a real-world, industry-focused database. Figure 16 shows the normalized search throughput (QPS) for the QBAT (ML) and QBAT (HR) compared to a baseline and a theoretical oracle. In the performance mode scenario, QBAT (HR) improves throughput by an average of 36% over StaticPerf and QBAT (ML) also provides a substantial 30% boost. The performance advantages of the proposed methods are even more pronounced in the consistency mode scenario, which reflects a more practical use case where search accuracy is critical. The QBAT (ML) and QBAT (HR) deliver an average throughput



(a) Performance mode

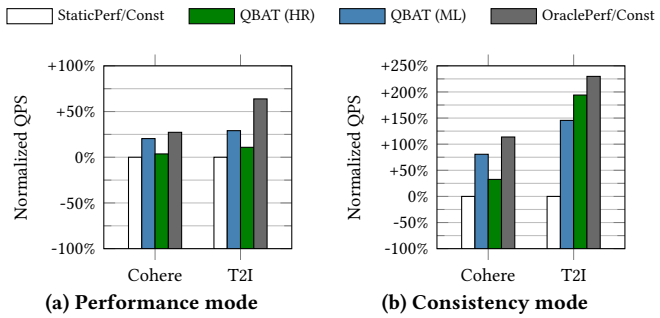
(b) Consistency mode

Figure 16: Normalized total searched query budget reduction in AlloyDB across two requirements.

improvement of 44% and 60%, respectively, over the baseline. The data supports the conclusion that both the QBAT (ML) and QBAT (HR) improve throughput on a production-grade vector database system.

5.9 Preliminary Results on Graph-Based Index

To test the generalizability of the QBAT methodology, we applied it to HNSWlib—a graph-based approximate nearest neighbor search algorithm. We replicate the end-to-end process from query profiling to feature selection and model training using both LightGBM and AlphaEvolve for QBAT (ML) and QBAT (HR) respectively. However, unlike in cluster-based methods where we predict the number of leaves to search, for HNSW we predict the search parameter `ef` as



(a) Performance mode **(b) Consistency mode**

Figure 17: Normalized total searched query budget reduction in HNSWLib across two requirements. BigANN and OpenAI are omitted because static and dynamic budgets uniformly allocate `ef=1` across all queries.

"query budget," which controls the size of the dynamic list of candidates. Instead of the two features in ScaNN, our feature selection suggests five features (in order of mean absolute importance): (1) distance to the Layer 0 entry point; (2) standard deviation of distances to its neighbors; (3) number of distance evaluations required to traverse upper layers; (4) degree of the Layer 0 entry point; (5) distance to the global entry point.

We observe that for the BigANN and OpenAI datasets, the target recall of 0.95 is achieved even with the minimum search budget of `ef=1` across all queries, making static and dynamic budget equivalent. Therefore, we focus our evaluation on the more challenging Cohere and T2I datasets. The results, shown in Figure 17, demonstrate that the same principles can yield significant gain in graph methods as well. While QBAT (ML) provides noticeable throughput improvement of up to 29% in performance mode and over 145% in consistency mode, there remains a substantial gap from QBAT to the Oracle, indicating significant room for further improvement. Estimating query difficulty from local graph features involves capturing the discrete topology and path-dependent greedy traversal of the navigation graph, which is inherently more complex than estimating from continuous point-to-set distances in cluster-based methods. Identifying finer-grained query budget knobs (to optimize other workloads like BigANN and OpenAI) and budget modeling for graph-based indexes remain open research questions.

6 RELATED WORKS

Query-aware Performance Knob Tuning. Previous research [8, 12, 51, 52] highlights that fixed search configurations are inefficient as they ignore unique query characteristics. LIRA [51] utilizes a machine learning model to predict neighbor partitions and introduces a redundancy method to duplicate long-tail data points, reducing the search space. LEQAT [52] employs a simple DNN to estimate kNN distributions and generates custom search plans for partition probing. ProS [12] provides progressive intermediate answers with quality guarantees, allowing users to confidently terminate searches early based on improvement probability. Unlike these methods—which often focus on graph-based indices or use low-accuracy DNNs for small centroid counts—QBAT handles realistic workloads with over 10K centroids by separately optimizing for performance and QoS. Similarly, while DARTH [8] probes an ML model during index scanning to assess recall targets, it requires more runtime features than QBAT to fuel its early-stop logic.

Query Hardness Measurement. Query hardness research suggests that search difficulty is inherent to the data's local geometry [20, 24]. Local intrinsic dimensionality (LID) [20] measures how quickly data points increase in a hypersphere around the query; high LID indicates a dense local neighborhood and a harder query. Relative contrast (RC) [24] offers a global perspective by comparing the distance to the k-th nearest neighbor with the mean distance to all points. Zoumpatianos1 et al. [53] formalized ϵ-hardness as the fraction of the dataset within a hyper-sphere around the query's true k-th nearest neighbor, primarily used for generating controlled query workloads. Wang et al. introduced steiner-hardness [47], a graph-native measure based on graph connectivity for graph-based indexes. While these methods predict difficulty using arithmetic operations, they require ground truth information, making runtime measurement problematic. In contrast, our autotuner predicts query difficulty using distance information between the query and centroids generated during the search process.

Vector DBMS Performance Tuning. Automated database tuning has evolved from rule-based methods to machine learning to optimize complex systems [9, 15, 31, 39, 50]. For relational databases, QTune [31] uses reinforcement learning to recommend system settings based on SQL features. VDTuner [50] utilizes multi-objective Bayesian optimization (MOBO) to adapt to various index types without prior domain knowledge, employing a holistic model and a polling strategy to efficiently allocate its tuning budget. Beyond parameter tuning, systems like Quake [39] restructure indices based on a cost model that estimates latency from partition sizes and access frequencies. In contrast, our research focuses on DB configuration for vector management operations like inserting, modifying, or deleting data. Our autotuner concentrates specifically on predicting and optimizing the query budget for individual user queries, making it orthogonal and complementary to existing database tuning research.

7 CONCLUSION

This paper demonstrates the inherent inefficiency of using a static query budget for approximate nearest neighbor search (ANNS). A fixed budget compromises between speed for easy queries and accuracy for difficult ones, failing to adapt to the heterogeneous difficulty of queries. To address this, we developed a query-aware budget autotuner that dynamically allocates query budget based on query-specific features. Our work makes a conceptual contribution through its dual-objective framework (performance mode vs. consistency mode), aligning system behavior with strategic goals. We introduced two predictive models, including a highly accurate GBDT model. QBAT consistently outperformed a well-tuned static baseline, achieving up to a 68.8% and 72.2% reduction in query budget on ScaNN and Faiss-IVF, respectively, while maintaining target recall levels. Whether optimizing for minimum cost with the performance mode or for consistent stability with the consistency mode approach, the QBAT provides a more efficient and intelligent solution for large-scale similarity search.

REFERENCES

[1] Laurent Amsaleg and Herve Jegou. 2010. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr/.

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of the Twelfth International Conference on Learning Representations*. Vienna, Austria, 30. https://openreview.net/forum?id=hSyW5go0v8

[3] Andrea Bacciu, Florin Cuconasu, Federico Siciliano, Fabrizio Silvestri, Nicola Tonellotto, and Giovanni Trappolini. 2023. RRAML: Reinforced retrieval augmented machine learning. arXiv:2307.12798 [cs.CL] https://arxiv.org/abs/2307.12798

[4] Dmitry Baranchuk and Artem Babenko. 2021. Benchmarks for billion-scale similarity search. https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search.

[5] Dmitry Baranchuk, Matthijs Douze, Yash Upadhyay, and I Zeki Yalniz. 2023. Dedrift: Robust similarity search under content drift. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 11026–11035.

[6] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: Self-tuning indexes for similarity search. In *Proceedings of the 14th International Conference on World Wide Web* (Chiba, Japan) *(WWW '05)*. Association for Computing Machinery, New York, NY, USA, 651–660. https://doi.org/10.1145/1060745.1060840

[7] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162. PMLR, Baltimore, MD, USA, 2206–2240. https://proceedings.mlr.press/v162/borgeaud22a.html

[8] Manos Chatzakis, Yannis Papakonstantinou, and Themis Palpanas. 2025. DARTH: Declarative Recall Through Early Termination for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 4, Article 242 (Sept. 2025), 26 pages. https://doi.org/10.1145/3749160

[9] Chao Chen, Shixin Huang, Xuehai Qian, and Zhibin Yu. 2025. Swift: Fast performance tuning with GAN-generated configurations. In *Proceedings of the 2025 USENIX Annual Technical Conference*. USENIX Association, Boston, MA, 1553–1568.

[10] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient billion-scale approximate nearest neighbor search. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 398, 14 pages.

[11] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The Faiss library. arXiv:2401.08281 [cs.LG] https://arxiv.org/abs/2401.08281

[12] Karima Echihabi, Theophanis Tsandilas, Anna Gogolou, Anastasia Bezerianos, and Themis Palpanas. 2022. ProS: Data series progressive k-NN similarity search and classification with probabilistic quality guarantees. *The VLDB Journal* 32, 4 (Nov. 2022), 763–789. https://doi.org/10.1007/s00778-022-00771-z

[13] Hugging Face. 2023. Cohere/wikipedia-22-12. https://huggingface.co/datasets/Cohere/wikipedia-22-12/tree/main/en.

[14] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 541–552. https://doi.org/10.1145/2213836.2213898

[15] Victor Giannakouris and Immanuel Trummer. 2025. λ-Tune: Harnessing large language models for automated database system tuning. *Proceedings of the ACM on Management of Data* 3, 1, Article 2 (Feb. 2025), 26 pages. https://doi.org/10.1145/3709652

[16] Google. 2023. AlloyDB for PostgreSQL. https://cloud.google.com/alloydb.

[17] Google. 2023. AlloyDB Omni. https://cloud.google.com/alloydb/omni.

[18] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 364, 10 pages.

[19] Ben Harwood and Tom Drummond. 2016. FANNG: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Las Vegas, Nevada, USA). 10.

[20] Michael E. Houle. 2017. Local Intrinsic Dimensionality I: An extreme-value-theoretic foundation for similarity applications. In *Similarity Search and Applications*. Springer International Publishing, Cham, 64–79.

[21] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endow.* 9, 1 (Sept. 2015), 1–12. https://doi.org/10.14778/2850469.2850470

[22] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 604–613. https://doi.org/10.1145/276698.276876

[23] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. 1997. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (El Paso, Texas, USA) *(STOC '97)*. Association for Computing Machinery, New York, NY, USA, 618–625. https://doi.org/10.1145/258533.258656

[24] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. arXiv:1810.07355 [cs.DB] https://arxiv.org/abs/1810.07355

[25] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., San Jose, CA, USA.

[26] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[27] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[28] Yannis Kalantidis and Yannis Avrithis. 2014. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Columbus, Ohio, USA). 8.

[29] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

[30] 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. 2020. Retrieval-augmented generation for knowledge-intensive NLP Tasks. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[31] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. QTune: a query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.* 12, 12 (Aug. 2019), 2118–2130. https://doi.org/10.14778/3352063.3352129

[32] Shengren Li and Nina Amenta. 2015. Brute-force k-nearest neighbors search on the GPU. In *Similarity Search and Applications*. Springer International Publishing, Cham, 259–270.

[33] Jimmy Lin. 2009. Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Boston, MA, USA) *(SIGIR '09)*. Association for Computing Machinery, New York, NY, USA, 155–162. https://doi.org/10.1145/1571941.1571970

[34] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 4768–4777.

[35] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (April 2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[36] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Proceedings of the Advances in Neural Information Processing Systems* (Long Beach, CA, USA), Vol. 30. Curran Associates, Inc., 12.

[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv:1301.3781 [cs.CL] https://arxiv.org/abs/1301.3781

[38] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Umar Farooq Minhas, Jeffery Pound, Cedric Renggli, Nima Reyhani, Ihab F Ilyas, Theodoros Rekatsinas, and Shivaram Venkataraman. 2024. Incremental ivf index maintenance for streaming vector search. *arXiv preprint arXiv:2411.00970* (2024).

[39] Jason Mohoney, Mengze Tang, Shihabur Rahman Chowdhury, Anil Pacaci, Ihab F. Ilyas, Theodoros Rekatsinas, and Shivaram Venkataraman. 2025. Quake: Adaptive indexing for vector search. In *Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Boston, MA, 153–169.

[40] Edoardo Mosca, Ferenc Szigeti, Stella Tragianni, Daniel Gallagher, and Georg Groh. 2022. SHAP-based explanation methods: a review for NLP interpretability.

In *Proceedings of the 29th international conference on computational linguistics*. 4593–4603.

[41] Marius Muja and David G. Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36 (2014), 14.

[42] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv:2506.13131 [cs.AI] https://arxiv.org/abs/2506.13131

[43] OpenAI. 2023. OpenAI text embeddings. https://platform.openai.com/docs/guides/embeddings.

[44] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 2227–2237. https://doi.org/10.18653/v1/N18-1202

[45] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search. *arXiv preprint arXiv:2105.09613* (2021).

[46] Philip Sun, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar. 2023. SOAR: Improved indexing for approximate nearest neighbor search. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 141, 16 pages.

[47] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-Hardness: A query hardness measure for graph-based ANN indexes. *Proc. VLDB Endow.* 17, 13 (Sept. 2024), 4668–4682. https://doi.org/10.14778/3704965.3704974

[48] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, et al. 2023. Spfresh: Incremental in-place update for billion-scale vector search. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 545–561.

[49] Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. 2024. Retrieval-augmented generation with knowledge graphs for customer service question answering. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 2905–2909. https://doi.org/10.1145/3626772.3661370

[50] Tiannuo Yang, Wen Hu, Wangqi Peng, Yusen Li, Jianguo Li, Gang Wang, and Xiaoguang Liu. 2024. VDTuner: Automated performance tuning for vector data management systems. In *Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE)* (Utrecht, Netherlands). IEEE, Los Alamitos, CA, 4357–4369. https://doi.org/10.1109/ICDE60146.2024.00332

[51] Ximu Zeng, Liwei Deng, Penghao Chen, Xu Chen, Han Su, and Kai Zheng. 2025. LIRA: A learning-based query-aware partition framework for large-scale ANN search. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) *(WWW '25)*. Association for Computing Machinery, New York, NY, USA, 2729–2741. https://doi.org/10.1145/3696410.3714633

[52] Pengcheng Zhang, Bin Yao, Chao Gao, Bin Wu, Xiao He, Feifei Li, Yuanfei Lu, Chaoqun Zhan, and Feilong Tang. 2022. Learning-based query optimization for multi-probe approximate nearest neighbor search. *The VLDB Journal* 32, 3 (Sept. 2022), 623–645. https://doi.org/10.1007/s00778-022-00762-0

[53] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. 2018. Generating data series query workloads. *The VLDB Journal* 27, 6 (Dec. 2018), 823–846. https://doi.org/10.1007/s00778-018-0513-x