A Benchmark and an Evaluation for SQL Queries with k-Nearest Neighbor Search

Raj Bapat
University of California, Davis
Davis, CA
rbapat@ucdavis.edu

ABSTRACT

AI/ML is quickly becoming core to OLTP applications. Applications are augmenting public LLM models with private data within databases through vector similarity search. SQL databases are adding storage for vectors, SQL operators for vector distance calculation and k-Nearest Neighbor (kNN) Search constructs as an extension to SQL. While there are benchmarks and evaluation of approximate KNN search, there is limited understanding of the performance and quality of SQL query processing of vectors, particularly when combined with kNN search. This paper presents a new OLTP database benchmark, TPCCV, to evaluate SQL processing with vectors. The benchmark builds on TPC-C, the industry standard for benchmarking OLTP databases, extending it by adding queries on tables with vectors including SQL that combines select, project, filter, and joins with approximate kNN search. This paper provides an overview of the benchmark system, an evaluation of PostgreSQL using this benchmark and points out opportunities for future research.

PVLDB Reference Format:
Raj Bapat. A Benchmark and an Evaluation for SQL Queries with k-Nearest Neighbor Search. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Historically, much of the focus of vectors in high dimensional spaces, their storage and processing has been in data mining and search. With the advent of deep learning and now with embedding APIs from LLM services such as Google and OpenAI, high dimensional vectors are becoming the norm to represent all forms of unstructured data such as an audio recording of a support call, product images, and textual item descriptions. Vector embeddings are enabling creation of smarter OLTP applications that can unlock the value of unstructured data by using vector distances as a similarity measure to enhance user experience with new insights such as the sentiment in an audio recording or offering similar products that match a user's purchase history. As a result, the use of Nearest Neighbor (kNN) Search on vectors is now becoming integral to the vast space of OLTP applications.

Vector embeddings derived from public LLM models are typically produced from text and other unstructured data associated with a business's private databases. These vectors are then used by applications together with other relational data in databases, thus driving demand for vector data management and query processing. Retrieval augmented generation (RAG) [15] enables the use of vector databases to store memory or context of conversations or user history to provide a more contextual generation of responses from LLMs. Applications that deal with text such as support case management systems, are adopting vector kNN search to provide responses from knowledge bases that are contextualized with the case's own history. Applications that deal with documents are augmenting text search engines with vector kNN search to perform semantic searches based on embedding vectors that are computed on chunks of those documents using LLMs. The need for building vectors into database applications is further spurred by SaaS (Software as a Service) services which have multi-tenant relational databases with additional security and privacy constraints. They need to perform a SQL queries to narrow down the query to vectors that belong to a specific tenant, and perform vector kNN search only on a subset of documents or text fields that are accessible to a specific tenant. A tenant (such as a business) may have multiple users. This imposes an additional constraint of an even narrower set of rows that an individual user within the tenant is authorized to access. These constraints are typically implemented by SaaS services using SQL join and filter predicates and are necessary for security and privacy preserving query processing on vectors.

To support these application needs, kNN data management and query processing is being adopted by special-purpose and general-purpose databases. While there are a number of specialized vector databases already in the market that specialize primarily in storage and search of vectors, **kNN search is being increasingly adopted in SQL databases**. Bringing vectors and kNN search to SQL databases is resulting in the creation of a powerful unified platform that can manage both traditional structured data and vectors, and offer a single query processing language and execution environment for both structured and unstructured data.

Specialized vector databases are primarily developed by startups like Pinecone [5] and Milvus [3]. Building kNN search into SQL Databases is being pursued by startups, big tech companies as well as open source products. Alibaba has incorporated vector search in their OLTP database service, Pase [24] and in their analytic database service, AnalyticDB-V [22].

The fastest growing database, PostgreSQL, offers vector search as a SQL extension called pgvector [4]. The pgvector extension is

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

already supported in the Azure Database for PostgreSQL, Google Cloud's Cloud SQL for PostgreSQL and Amazon Web Service's RDS for PostgreSQL database services. The pgvector extension introduces a vector data type, vector similarity operators, and graph-based vector indexes for approximate kNN search. SQL's expressive power expands to supporting vector storage and **hybrid queries** - that combine SQL constructs with vector similarity search expressions. Hybrid queries can be used to retrieve the top k nearest vectors according to the specified distance function and also satisfying the rest of the SQL query constraints. This paper with focus on SQL that combines Select, Project, Join, Insert operations with vectors and vector search.

Current state-of-the-art in benchmarks. Currently, ANN-Benchmarks [9] is the leading framework for evaluating kNN search. It is well suited in evaluating nearest neighbor search algorithms in the isolated context of a search space of vectors, with no non-vector filters applied. The underlying algorithms and systems being evaluated by ANN-Benchmarks are mostly not relational database management systems with SQL query planning or execution capabilities. These specialized systems are targeting standalone vector search use cases that do not require the expressive power of SQL required by most relational database applications.

As vector similarity search becomes mainstream in SQL databases, there is a need to assess the **performance and quality of hybrid queries** for use cases in a relational database setting. For example, a benchmark that covers hybrid queries needs to assess recall in approximate kNN search that have highly selective filter predicates, such as those typical in OLTP database environments.

Traditional database benchmarks, TPC-C and TPC-H have been the mainstay for evaluating database performance for 30 years and TPC-C continues to be the primary measure for OLTP databases to date [7]. There is a need to extend these benchmarks to include management of vectors and kNN search.

Contribution. This is the first benchmark and evaluation of a relational database that systematically assesses how such a database performs at a combination of traditional SQL and kNN operations. This paper makes two key contributions.

- First, **a new benchmark, TPCCV,** for hybrid databases is presented, derived from the most popular and standard OLTP database benchmark by extending it to include performance and quality of SQL that includes operations on vectors. The benchmark is fully automated and automatically generates output plots, along with the detailed run output and logs. The plots in this paper were all generated by the benchmark.
- Second, for the first time, an **evaluation of PostgreSQL's hybrid queries** is presented. While the power of having SQL and vector search in a single SQL Database is a boon to application developers, the paper identifies **challenges with hybrid queries** in PostgreSQL's pgvector extension and provides directions for future research.

The benchmark is shared with the community as an Open Source project to enable others utilize it for further evaluation and research.

2 RELATED WORK

There has been significant body of work in kNN search algorithms. The range of techniques developed include hashing-based [8, 11, 14], tree-based [10, 12] and graph-based [12, 13, 16] algorithms.

Still, there is limited research in data management, query processing and query optimization for the combination of unstructured (vector) and structured constraints. An even more nascent area of work is that of vector similarity search in combination with complex SQL processing, such as joins.

To deepen our understanding of this area, this paper evaluates the state-of-the-art in storage, transactions and query processing of vector in SQL databases, with the focus on PostgreSQL.

2.1 Current state of vectors in SQL Databases

In a SQL database, vectors are associated with structured data, as a structured attribute of a relation in a SQL database. Further, these attributes may be connected (through foreign keys) to other relations in the SQL database when represented in a normal form. Vectors need a way to be represented in the schema, stored in the database, support ACID transaction semantics, work operationally with operations like backup/restore and provide query optimization and processing that spans vector and structured data.

Vector representation and operators in SQL Databases. PostgreSQL, PASE, AnalyticsDB-V all support representing vectors in SQL databases. PostgreSQL pgvector extends the PostgreSQL SQL language with the introduction of the VECTOR data type, which holds an array of 32-bit floating point numbers. It also defines operators, <=> for cosine distance, <#> for negative inner product and <-> for euclidean distance and additional functions on vectors.

A table can be created with a vector simply with:

```
CREATE TABLE item_vector
  (..., iv_vector VECTOR(960));
```

The table's vector column can have an HNSW-type approximate kNN search index on it for speeding up approximate searches:

```
CREATE INDEX ON item_vector
USING HNSW (iv_vector vector_cosine_ops);
```

Transaction and Recovery Model. PostgreSQl pgvector provides fully integrated data management for vectors within the SQL database. So database operations like backup/restore, point-in-time recovery, crash recovery, and transaction processing (inclusive of DML statements) follows standard PostgreSQL semantics. It ensures data integrity of vector data together with the structured data with ACID-compliant transaction semantics. This is a critical operational advantage of a unified database for both vector and structured data, in addition to the ability to write queries that combine vector and structured data.

Query Processing with filters. Search in a vector space in a SQL databases often need to be constrained by **filters** on the non-vector attributes associated with the vector. There is limited recent research in this field of queries that include both vector and

structured constraints. Recently, there has been recognition of the need to constrain kNN search with filters on associated structured attributes [1]. Most of the current research in approximate kNN search with filters is **limited to simple equality or "in list" filters**. For example, Milvus [17] supports simple range predicates on attribute columns (for example, *age* >= 18). The SQL databases, pgvector, PASE and AnalyticsDB-V, support adding vector predicates and vector search to a wide range of filter predicates.

PostgreSQL pgvector extends the SQL query language and allows to easily write SQL predicates that combine arbitrarily complex attribute filters with vector similarity comparisons as below:

```
SELECT * from item_vector iv
WHERE iv.iv_v <=> [1, 2, 3] > 0.9 AND ...
```

PostgreSQL pgvector also makes it easy to express a kNN search query by simply adding ORDER BY and LIMIT clauses whith filtering:

```
SELECT * from item_vector iv
WHERE iv.iv_price > 100
ORDER BY iv.iv_v <=> [1, 2, 3]
LIMIT 10
```

The above query may choose to use a approximate kNN search index if one is available.

There are multiple strategies suggested in research for implementing filtering that combine vector search with filters. The filter-first-then-search (F-S), applies filter predicates against all rows to subset the set of vectors to search followed by either an exact kNN search by scanning the relevant vectors. This strategy can work particularly well for low selectivity filters. When the number of rows returned by the filter is large, this strategy is expensive [17, 22]. The search-first-then-filter Second is post-filtering (S-F), applying approximate kNN search to subset the set of vectors and prune them further by applying predicate filters on the vectors returned by the kNN search.

Both the S-F and F-S algorithms and their variants suffer from drawbacks in performance or quality. The S-F algorithm increases computational and I/O expense by having to expand the candidate list (referred to as amplification[22]) to compensate for the reduction in the set of vectors after applying the filter in the second stage. The F-S algorithm eliminates the ability to use a pre-computed index structure, increasing compute and I/O costs by necessitating a full scan of vectors and an exact comparison with each that qualified the filter.

PostgreSQL pgvector uses the F-S strategy when using exact kNN search. It can be forced to use S-F strategy with a query rewrite using a common table expression.

An alternative to S-F and F-S algorithms is to create a "fusion distance metric"[18], a weighted sum of the vector distance and a new distance measure that computes the hamming distance after transforming discrete attribute values to their ordinal space. This new distance function can be applied to graph or tree approximate search algorithms to adapt them to perform a single search that incorporates both vector and attribute distances. HQANN [23] and [20] more accurately computes attribute distances. This method remains limited to only equality predicate on attributes. Currently, PostgreSQL does not implement a fusion metric based proximity graph.

Query Processing with joins. The author is not aware of research in either evaluating or optimizing kNN search with complex SQL processing such as Select-Project-Join (SPJ) queries.

With PostgreSQL pgvector, an SPJ query can include vector similarity comparisons as below:

```
SELECT *
FROM item_vector iv, order_line ol
WHERE ...
ORDER BY iv.iv_v <=> [1, 2, 3]
LIMIT 10
```

Query Optimization. Techniques have been suggested that involve using a query optimizer to choose between pre-filtering and post-filtering [22]. A query pruning optimization has been proposed that, similar to range partition pruning in SQL Databases, narrows down the search space to a subset of partitions of the table based on ranges of an attribute specified in the query. This technique can speed up both pre-filtering and post-filtering techniques by pruning the search space. Milvus [17] leverages this in approach in their specialized vector database.

2.2 Current state of kNN search benchmarks in SQL Databases

A fundamental step towards research is a performance evaluation of a system. Such evaluations can provide deep insight into the challenges that need to be addressed. Additionally, a benchmark can help reproduce the evaluation and measure the effects of new research work by providing a framework for evaluating the benefits of the research work. Except for AnalyticsDB-V, presently there is no research into evaluating vectors search with SQL processing. Presently, there is also no benchmark framework to perform such evaluations.

Most evaluations of performance and recall of vector search has been for standalone vector search queries. This methodology is popularized by ANN-Benchmarks. ANN-Benchmarks has made objective comparison of pure vector search consistently and easily reproducible. ANN-Benchmarks does not support evaluation of vector search when combined with other SQL operators as is common in relational databases.

There has been some research in simple hybrid queries that combine vector search with simple attribute filters (in a non-SQL setting) such as NHQ [19], HQANN [23] and Filtered-DiskANN [19]. However, they do not cover much of the powerful capabilities of processing relational data in SQL. The research limits itself to equality predicates and searches for neighbors that incorporate attribute comparisons and similarity search over vector.

Research on vectors in PostgreSQL with PASE done by [21] is focused on comparing performance of vector operations without

any SQL operators. Their focus is to compare vector index creation and vector search against a standalone vector index outside of a SQL databases. The study does not cover SQL queries such as filters or joins. Research on vectors in PASE included build performance and query performance, although only for Top-1 queries. The work does not include any hybrid queries that combine vector search with SQL filters or joins.

Research on vectors in AnalyticsDB-V evaluates performance and recall for hybrid queries that are single table queries that include vector search and predicates on another column. They do not cover index build performance or performance and recall in the context of more complex queries like joins. AnalyticsDB-V being an analytics database would typically have a need for complex SQL query patterns even when used in conjunction with vector search.

This paper explores a benchmark and an evaluation of complex SQL query patterns and transaction processing in conjunction with vector search for the very first time, shedding new light to this area.

3 BENCHMARK DEFINITION

This section defines goals in designing the TPCCV benchmark. Specific real-world use cases are presented that lead to the the development of the specified SQL operations and query patterns that are part of TPCCV.

Goals. The TPCCV benchmark measures the performance and quality of using a hybrid database for SQL over vectors in a relational database. There are two goals of this benchmark:

- First, the benchmark measures the performance transaction performance of INSERT and DELETE SQL statements on vector data in a SQL database with and without approximate kNN search indexes. While the benchmark includes these operations, that is beyond the scope of this paper.
- Second, the benchmark measures the performance of hybrid queries, SELECT statements that combine traditional select-filter-join statements along with vector operations with exact and approximate kNN. For approximate kNN, the benchmark measures both the performance as well as recall for hybrid queries with approximate nearest neighbor search predicates.

Real-world Use Cases. The TPCCV benchmark's schema, data and SQL is tailored to real-world use cases to maximize its utility in OLTP applications. TPCCV uses the scenario of an "order management system", where customers search for items to purchase and place orders against a set of warehouses (stores). The paper presents results from two key real world use cases:

- **Item Search with filters on Items** Search for items with a filter on the table with varying selectivity to exercise similarity search under varying sized subsets of the table.
- **Item Search within a Customer's Purchase History.** Search for the most similar items from customer's purchase history.

Translated scope of SQL Statements. Based on the above use cases, TPCCV defines scope of queries to the following categories of SQL statements:

- **SP-kNN.** Single table Select-Project query with exact kNN search with varying selectivity to exercise similarity search under varying sized subsets of the table.

```
SELECT id, vector_column <->
            '[e1, e2, , en]', ...
FROM {table}
WHERE {predicates}
ORDER BY vector <-> '[e1, e2, , en]'
LIMIT k;
```

- **SPJ-kNN.** Multiple table Select-Project-Join query with high selectivity and exact kNN search (SPJ-kNN)

```
SELECT id, vector_column <->
            '[e1, e2, , en]', ...
FROM table1, table2, ...
WHERE {predicates}
ORDER BY vector <-> '[e1, e2, , en]'
LIMIT k;
```

For each of the SQL queries, the optimizer automatically makes the decision whether to use the approximate kNN search index or not. We run these queries both with and without the approximate kNN search index on the table, to help with performance comparisons as well as enable calculation of ground truth to evaluate the recall for the approximate kNN queries.

4 SYSTEM DESIGN

4.1 System Under Test

The system under test for TPCCV includes a host where the TPCCV benchmark driver runs and a database server where the TPCCV's database workload runs.

TPCCV is configurable to any machine type for both systems. We utilize the db-perf-optimized-N-32 machine type for Cloud SQL for Enterprise Plus database service. The db-perf-optimized-N-32 machine includes 32 vCPUs, 256 GB of DRAM and 1.5TB of local SSDs. In TPCCV, we require the database to be configured with full ACID properties enabled, as would be typical in a real-world OLTP database deployment.

4.2 Database Schema and Data

Since this is an OLTP database benchmark, with the focus on measuring vector operations, the TPCCV benchmark uses as its data model and base data from TPC-C [6], the industry's most popular standard for measuring OLTP performance.

TPC-C is modeled around a wholesale parts supplier that operates a number of warehouses and sales districts where customers buy products. The workload encompass entering customer orders, to delivery of product, order status queries by customers, and stock level checks as some of the core transactions. TPC-C has been the most popular benchmark for nearly 30 years. TPC-C's characterization of the OLTP workload has served as a good representative

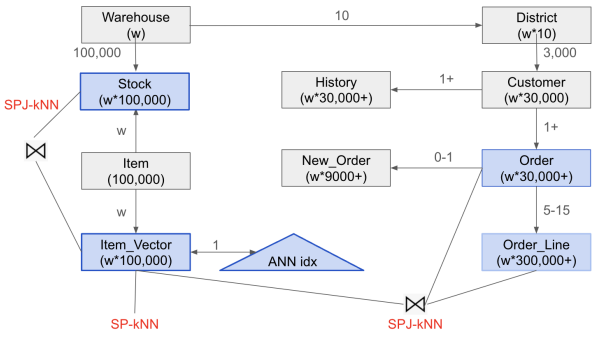


Figure 1: TPCCV schema extends the TPC-C schema through the addition of a table that holds embedding vectors.

of needs of OLTP applications, and it has been widely adopted by database vendors and more recently by Cloud service providers.

Derivative benchmark of TPC-C. TPCCV is new derivative of TPC-C that preserves the benchmark's strength over several decades of OLTP benchmarking, and introduces kNN search to it. Similar to why HammerDB created TPROC-C as a derivative of the well designed and scalable TPC-C specification to implement an OLTP workload that is accurate, repeatable and representative of the real-world [2], we have created an extension of the TPC-C specification that is a derivative of TPROC-C to include kNN search while preserving all the strengths of the TPC-C specification.

In making TPCCV, we adopted the same principles that were behind the creation of TPCCV - we want running the benchmark to be fast, low cost and accessible to all. We are also making TPCCV open source, thereby enabling anyone to compare the performance of databases.

TPCCV data model. TPCCV extends the data model of TPC-C to incorporate vectors to the item table of the schema. As seen in Fig 1 TPCCV adds a second table item_vector that has a 1-on-1 correspondence to stock and is modeled to hold the embedding vector derived from the text description of the item in a specific warehouse - both the description of the item and its location information in the warehouse. The item_vector is stored separately in order to keep the physical layout separate from the stock table, to maximize performance of accessing each individually.

The item_vector table will be used to perform vector similarity search. Since this is a normalized OLTP schema, queries other than the most general "find me an item similar to X", will **require filters and joins across two or more tables in the TPC-C schema**. For example, a similarity search across the items in the purchase history for a given customer, requires a join across multiple tables (orders ⋈ order_lines ⋈ item_vector).

4.3 Vector Data

TPCCV's item_vector table holds vectors. The benchmark is intended to be used to standardize testing using a small number of data sets, so benchmark results are easily comparable. TPCCV uses GIST1M with 960 dimensions as the default data set and scales it as needed to the size of the item_vector table by pruning or scaling the data set while preserving the original data clustering. There are pros and cons of this approach. On the one hand there is regularity in the shape of the data set across all sizes allowing the results to be self-consistent. However, if the benchmark evaluation is needed on a more diverse set of data sets, TPCCV can be configured to use an alternative data set as desired.

In addition to the base data set of vectors, TPCCV also requires a set of test vectors to be used for search queries. TPCCV uses the query vectors that come with with GIST1M in its queries.

The ground truth for the queries are first computed using exact search (no use of approximate nearest search index) and then used to compute recall when performing approximate search.

4.4 Execution

The benchmark is executed as a pipeline of 5 stages, each stage executed serially:

- The first stage is the resource acquisition and configuration phase.
- The second stage is setting up the benchmark - setting up the TPCCV schema, loading the relational tables and the vectors from the vector dataset.
- The third stage is to run the series of benchmark queries and transactions (inserts/deletes) on the database without any approximate indexes. This phase gives us the ground truths for the queries.
- The fourth stage is to run the series of benchmark queries and transactions (inserts/deletes) on the database with the approximate index.
- The fifth and final stage processes all the output files from the individual stages and produces the computed results such as the recall score for the ANN queries and the plots to visualize the results.
  All plots included in the paper are unmodified results from the above pipeline and were reproduced multiple times by rerunning the benchmark to ensure consistency.

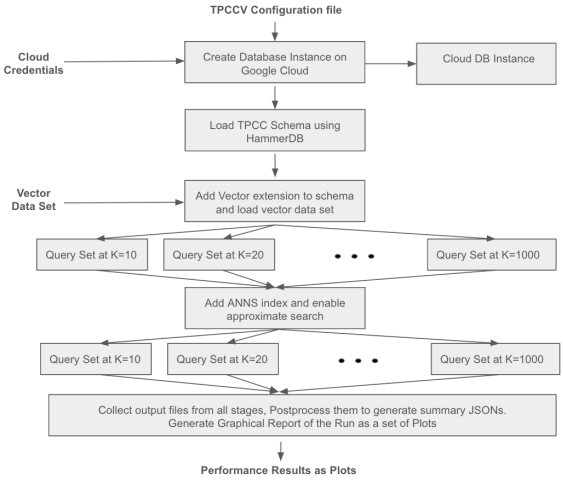


Figure 2: An overview of the automation pipeline that interacts with Google Cloud, HammerDB and data sets.

The architecture is designed in such a way that multiple pipelines can be kicked off on multiple machines (e.g. different sizes of machines or different data sets) in parallel, allowing executing the benchmark at large scale in a short period of time, running and analyzing thousands of queries for their performance and quality.

5 EVALUATION

We report the following findings from varying the selectivity, the ef_search and k values for these queries and measuring recall and elapsed time.

5.1 Exact SP-kNN Performance on 100K vectors.

Exact kNN queries on Items with varying selectivity.

In Fig 3, we see that queries are less than 3 seconds when the selectivity is 1000 rows or lower. Also, the query execution time increases as the selectivity goes up all the way to more than 300 ms. This is 1000 times more than a point lookup in databases. This is consistent with the query plan which first applies the filter and performs exact kNN distance calculation on the rows that pass the filter.

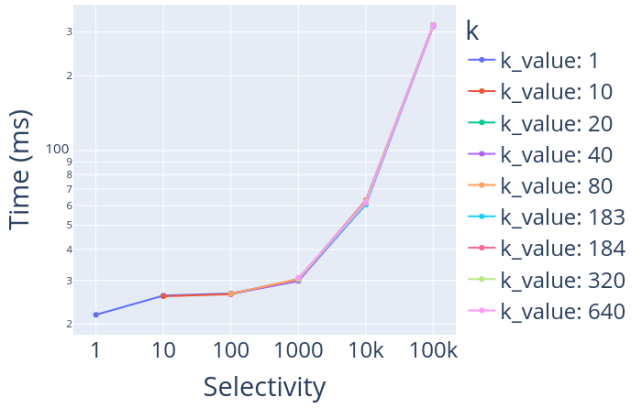


Figure 3: Performance of Exact kNN search using single table select and project with filters of varying selectivity.

5.2 Approximate versus Exact SP-kNN Queries

HNSW index is used only up to k=183 for SP-kNN queries.

The charts in Fig 4 plot the elapsed time versus ef_search for different k values. The three charts are for three fixed selectivity of 1000 rows, 10000 rows and 100000 rows. Typical query execution time for HNSW-based kNN search is in the 1 ms to 10 ms range.

To narrow down the specific k value of the inflection point, we show the specific two values at which point the optimizer switches plans.

Each of the chart shows that $k = 183$ represents an inflection point between the use of the HNSW index versus performing an exact search. At $k = 183$ and below, PostgreSQL optimizer generates a plan (independent of ef_search and selectivity) that queries the HNSW index followed by filtering. After $k = 183$, the plans switch

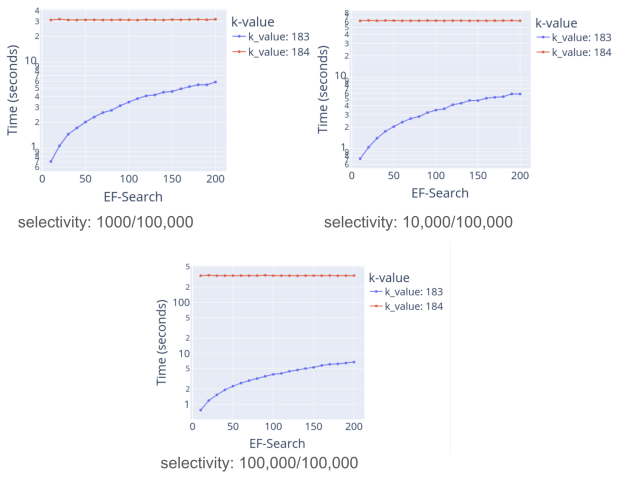


Figure 4: Performance of Approximate kNN search using single table select and project with filters of varying selectivity.

to using exact kNN search. As can be seen from the figure, this results in more than 5x increase in elapsed time for queries.

5.3 Low Recall for Approximate SP-kNN Queries with default ef_search

Under k=183, recall is poor for default configuration of search, ef_search=40

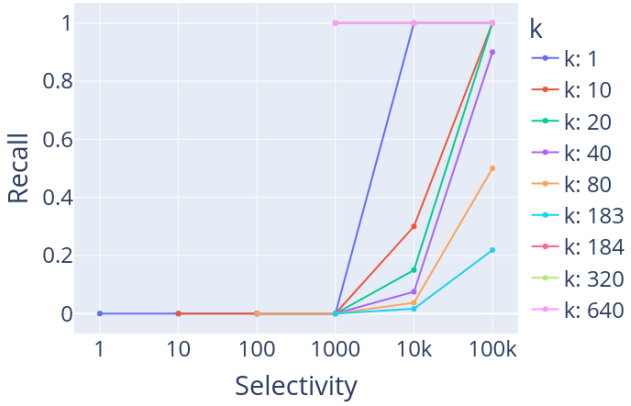


Figure 5: Recall of Approximate kNN search using single table select and project with filters of varying selectivity but fixed ef_search = 40.

As can be seen in Fig 5 recall is zero up to selectivity of 1000 rows, for the default ef_search setting of 40. This is an issue with the query plan, since post kNN search, low selectivity filters will reject most of the results of the approximate search. Recall remains low for higher selectivity. As noted earlier, k higher than 183 gets a recall of 1 because the query plan resorts to exact search.

5.4 Low Recall for Approximate SP-kNN Queries where selectivity is low

Recall remains very low for low selectivity queries even when increasing ef_search significantly

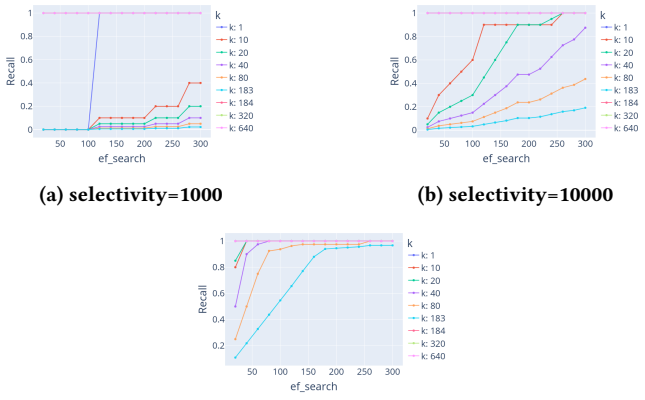
(a) selectivity=10 **(b) selectivity=100**

Figure 6: Recall with increasing ef_search.

As seen in Fig 6, recall remains zero for low selectivity cases even when we increase ef_search. Note that recall remains zero even with the highest levels of ef_search. This is because, HNSW search results in so few rows that all get rejected by the postfiltering applied in the S-F strategy.

5.5 Recall can be improved for Approximate SP-kNN Queries where selectivity is high

Improving recall for high selectivity queries requires increasing ef_search significantly



(a) selectivity=1000 **(b) selectivity=10000**

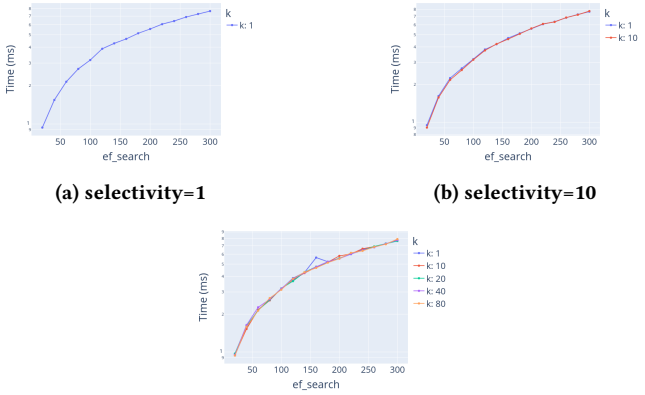
(c) selectivity=100000

Figure 7: Recall with increasing ef_search.

On the other hand, Fig 7 shows that recall improves for high selectivity cases when we increase ef_search significantly. Note that for medium selectivity queries where selectivity is 1000 rows, recall remains mostly below 0.5 even for the highest values of ef_search. This is an artifact that S-F strategy suffers on recall for cases where filters are returning fewer rows. When a significant number of rows pass the filter, increasing ef_search improves recall to be greater than 0.5 for small values of k. For large values of k, recall remains low.

5.6 Performance impact of increasing recall for Approximate SP-kNN Queries

Improving recall slows down queries linearly

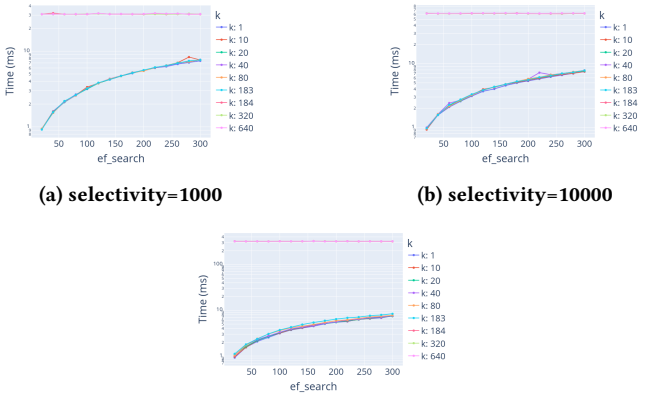


(a) selectivity=1 **(b) selectivity=10**

(c) selectivity=100

Figure 8: Time with increasing ef_search.

For low selectivity queries, as seen in Fig 8 increasing ef_search linearly increases the time executed by the query. Note that since there is no improvement in selectivity, as noted earlier, the increased cost is not warranted.



(a) selectivity=1000 **(b) selectivity=10000**

(c) selectivity=100000

Figure 9: Time with increasing ef_search.

For high selectivity queries, as seen in Fig 9 increasing ef_search also linearly increases the time executed by the query. Since at ef_search = 300, we got the highest recall, while the query time is still under 10 milliseconds, recommend use of high ef_search values as long as the corresponding linear increase in cost is acceptable.

5.7 Recall for Approximate SPJ-kNN Queries

Joins force use of exact search even when HNSW index is present

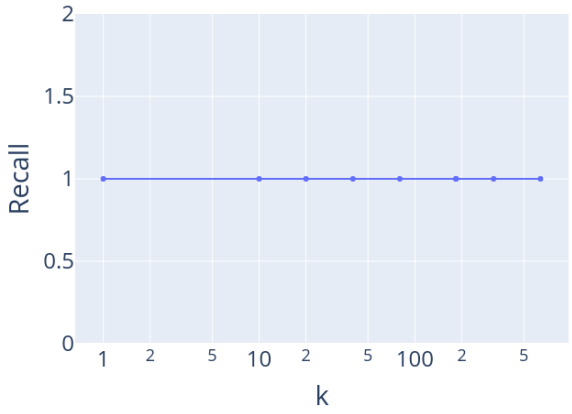


Figure 10: Recall versus k for SPJ (join) queries

As we see in Fig 10, when approximate searches are part of a join across tables, the kNN search is performed using exact search irrespective of k or number of rows selected. This is an understandable choice by the optimizer, but may cause significant costs for the query when a large number of rows qualify the join and filter conditions.

Because these are exact searches, there is no impact to recall.

5.8 Practical challenges tuning ef_search and build parameters.

The approach of requiring the user to set ef_search parameter repeatedly to try to get a good recall from the search is not practical and not feasible to implement in an OLTP environment, where we need to bound the number of tries of an already slow SQL operation. Most users are likely to not recognize the need to do a heuristics search on ef_search and may settle on a low quality result. Additional research is needed to improve usability of the search configuration settings.

The build configuration settings (m) are further out of reach of the typical user, as doing a comprehensive search is extremely time consuming, particularly given the slow build times of these indexes, and the best settings dependent on the data set. Additional research is needed to automatically learn from input data and have the system pick the build parameter. Alternatively, need for a high performance tool to experiment with different build parameters without requiring a full rebuild.

6 CONCLUSION AND FUTURE DIRECTION

This paper presents the first comprehensive evaluation of hybrid queries in a SQL database together with a benchmark for standardizing such evaluation in future research work.

The paper points out to challenges posed by combining approximate kNN search together with filter and join query operations when presented with different selectivity and k values. In order to tune recall and performance, we present the following key findings:

- A simple kNN search using an HNSW index is (starts at a few milliseconds) an order of magnitude slower than regular index operations in databases.
- Exact kNN search, while accurate in its response, is two orders of magnitude slower than an HNSW index search.
- Approximate kNN search is used by PostgreSQL only up to k=183. The optimize is hardwired to perform exact search when additional rows are requested.
- SP-kNN queries provide near zero recall when selectivity is low (most rows are filtered out in the S-F plan).
- SP-kNN queries provide low recall when selectivity is high (but not all rows). Recall can be tuned for these queries by increasing ef_search. Except for the most selective queries, recall may still remain low below 0.5 for high ef_search settings.
- SP-kNN queries will run significantly slower (by up to 5x) with high values of ef_search. Such tuning is onerous or impractical in real-world settings, since it is both data dependent and query dependent.
- SPJ-kNN queries does not use HNSW index and instead relies on exact search. This strategy works well when only a small number of rows qualify, but can quickly get expensive when 1000s of rows need to be evaluated for kNN search.

These findings point to a number of different avenues of research. We expect recall to follow the spirit of relational databases and transition to be "declarative", replacing the need to tune input parameters like ef_search. Additionally, we expect the query optimizer and execution engine to evolve to better optimize hybrid queries that include kNN searches. Additionally, the inherent impedance mismatch between two disparate systems in relational data processing and vector index navigation needs to be bridged, and future systems will need a more integrated approach to enable better recall **and** performance for hybrid SQL queries.

REFERENCES

[1] [n.d.]. *Getting Started with Hybrid Search.* Retrieved Dec, 2023 from https://www.pinecone.io/learn/hybrid-search-intro/

[2] [n.d.]. *HammerDB Benchmark.* Retrieved Dec, 2023 from https://www.hammerdb.com/

[3] Zilliz [n.d.]. *Milvus Vector Database.* Zilliz. Retrieved Dec, 2023 from https://milvus.io/

[4] [n.d.]. *pgvector PostgreSQL extension.* Retrieved Dec, 2023 from https://github.com/pgvector/pgvector

[5] Pinecone [n.d.]. *Pinecone Vector Database.* Pinecone. Retrieved Dec, 2023 from https://pinecone.io/

[6] [n.d.]. *TPC benchmark C standard specification.* Retrieved Dec, 2023 from https://www.tpc.org/tpcc/

[7] Google 2023. *AlloyDB for PostgreSQL - Transactional (OLTP) Benchmarking Guide.* Google. Retrieved Dec, 2023 from https://services.google.com/fh/files/misc/alloydb_oltp_benchmarking_guide.pdf

[8] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2017. Accelerated nearest neighbor search with quick adc. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval.* 159–166.

[9] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2017. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *International conference on similarity search and applications.* Springer, 34–49.

[10] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning.* 97–104.

[11] 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. *arXiv preprint arXiv:2111.08566* (2021).

[12] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228*

(2016).

[13] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).

[14] 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*. 604–613.

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[16] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[17] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[18] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2022. Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints. *arXiv preprint arXiv:2203.13601* (2022).

[19] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2024. An Efficient and Robust Framework for Approximate Nearest Neighbor Search with Attribute Constraint. *Advances in Neural Information Processing Systems* 36 (2024).

[20] Tian Wang, Cui Chen, Yubiao Chang, Ding Zhan, Chao Tian, Yunzhe Tian, Kang Chen, Endong Tong, Wenjia Niu, and Jiqiang Liu. 2023. Constrained Query Optimization in Differentiated Feature Spaces. (2023).

[21] Yunan Zhang Shige Liu Jianguo Wang. [n.d.]. Are There Fundamental Limitations in Supporting Vector Data Management in Relational Databases? A Case Study of PostgreSQL. ([n. d.]).

[22] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3152–3165.

[23] Wei Wu, Junlin He, Yu Qiao, Guoheng Fu, Li Liu, and Jin Yu. 2022. HQANN: Efficient and Robust Similarity Search for Hybrid Queries with Structured and Unstructured Constraints. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 4580–4584.

[24] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 2241–2253.