ISSN: 2634-9221
European Journal of Applied Sciences; Vol. 14 No. 03 (2026) (43-66)
https://doi.org/10.14738/aivp.1403.6884

# Bridging the Indexing Gap: Empirical Evaluation of Application-Side Optimisation Techniques for Native Vector Search in MySQL 9.0

Amos Ngoah[1], Dawit Abdisa Jebessa[2], & Yubo Yan[1]

1. School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
2. School of Software Engineering, University of Science and Technology of China, Suzhou, China

**Abstract:** The integration of vector similarity search into relational database management systems has become important for applications built around Retrieval-Augmented Generation. MySQL 9.0 introduces a native VECTOR data type, but the Community Edition still lacks server-side vector indexes and native distance functions, forcing similarity search to rely on a client-side full-scan model. This paper evaluates the practical consequences of that design through experiments on synthetic datasets of up to 10^5 vectors, together with a SIFT1M validation and small-scale sentence-embedding checks. The study focuses on where time is spent during execution by separating the costs of transfer, parsing, and computation. Results show that native binary VECTOR storage substantially improves data-handling efficiency over JSON-based storage, achieving up to 5.9x lower end-to-end latency in the tested settings, and that the dominant bottleneck shifts toward data movement as scale and dimensionality grow. Two established deployment strategies are then assessed. Hybrid Search, which combines scalar B-Tree filtering with vector ranking, achieves up to 54x speedup when selective metadata predicates are available. A two-pass strategy based on Principal Component Analysis (PCA) reduces transfer volume and achieves up to a 15.5x speedup for high-dimensional workloads, though its recall depends on the compression setting and data domain. Additional tests show that Hybrid Search is the most stable option under modest concurrency. The paper concludes with a practical decision framework for choosing between native VECTOR storage, scalar pre-filtering, PCA compression, and migration to a system with native approximate nearest-neighbour execution.

**Keywords:** MySQL, Vector Search, Database Optimisation, RAG, High-Dimensional Data.

## INTRODUCTION

The advent of the Transformer architecture [1] and the subsequent proliferation of Large Language Models (LLMs) have fundamentally transformed data management requirements. Modern applications increasingly rely on Retrieval-Augmented Generation (RAG) [2], [3] to ground generative models in proprietary data. By retrieving relevant context from a trusted knowledge base and injecting it into the generation process, RAG significantly mitigates the risk of hallucinations and factually incorrect or nonsensical outputs that plague standard LLMs when operating on out-of-distribution queries [4]. This paradigm necessitates the efficient storage and retrieval of high-dimensional vector embeddings generated by models such as Sentence-BERT [5].

### Challenges of Polyglot Persistence

Traditionally, this requirement has led to a bifurcated data architecture: structured relational data resides in a Relational Database Management System (RDBMS) (e.g., MySQL, PostgreSQL), while unstructured vector embeddings are offloaded to specialised vector databases (e.g., Milvus [6], Pinecone, Weaviate, Qdrant) [7]. While effective for similarity search, this polyglot persistence approach, as popularised by [8], adds significant architectural complexity. Foremost is the dual-write consistency challenge: ensuring atomic updates across two disparate systems is notoriously difficult without distributed transactions, often leading to synchronisation drift where the vector index becomes stale relative to the source of truth.

Furthermore, this architecture imposes a heavy operational burden, requiring engineering teams to manage separate infrastructure stacks, distinct security models, and independent backup strategies. The need to join query results from the vector store with metadata in the relational database introduces additional network latency and application-side complexity, undermining the real-time requirements of user-facing RAG applications.

### Converged Database Architectures

To address these challenges, the database community is moving towards converged architectures that embed vector search capabilities directly into the relational kernel. PostgreSQL, with its pgvector extension [9], has set the standard for open-source implementations, offering robust indexing support (HNSW, IVF-Flat) directly within the database engine. This trend is mirrored in the commercial MySQL ecosystem, where managed providers have introduced proprietary vector enhancements ahead of the upstream community. PlanetScale leverages the Vitess clustering system [10] to shard vector indexes across multiple MySQL nodes, enabling horizontal scalability that native MySQL lacks. Alibaba Cloud's PolarDB [11] adopts a decoupled storage-compute architecture, offloading vector index maintenance to a specialised distributed storage layer to avoid buffer pool contention. Similarly, Google Cloud has augmented Cloud SQL for MySQL with native vector indexing [12] by integrating approximate nearest-neighbour algorithms directly into the query execution engine, thereby validating the industry-wide shift towards unified SQL-Vector systems.

### MySQL 9.0 Native Vector Support

In July 2024, Oracle released MySQL 9.0 (Innovation Release) [13], marking a pivotal moment in the database's history by introducing a native VECTOR data type. This new binary type enables efficient storage of high-dimensional embeddings directly in InnoDB tables, replacing inefficient text-based workarounds. However, the feature set available in the open-source Community Edition is notably sparse compared to the commercial MySQL HeatWave offering [14]. Crucially, the Community Edition lacks support for the VECTOR INDEX statement, so it cannot build approximate nearest neighbour (ANN) indexes such as HNSW. Furthermore, it omits essential server-side scalar functions such as DISTANCE() or COSINE_SIMILARITY(). Consequently, developers are forced to retrieve raw binary vectors from the application layer to compute similarity scores, effectively degrading the database into a passive vector storage engine. This architectural limitation mandates O(N) full table

scans for every query, shifting the computational burden to the client and saturating network bandwidth with data that is largely discarded after ranking.

### Performance and Bottlenecks

This limitation creates a significant performance bottleneck for production workloads, effectively rendering the database unscalable for datasets that exceed available system RAM, particularly when the working set no longer fits in the configured InnoDB buffer pool. Without server-side indexing, every similarity search necessitates a full table scan, forcing the InnoDB engine to traverse the clustered index and load all vector pages into the buffer pool. This operation not only incurs high I/O latency but also causes severe buffer pool thrashing, where frequently accessed hot relational data is evicted to make room for vector data that is read once and immediately discarded. Furthermore, the requirement to transmit raw binary vectors to the application layer saturates network bandwidth. For a modest dataset of 1 million vectors with D = 1536 (approx. 6GB of data), a single query would require transferring gigabytes of data over the network, creating an insurmountable bottleneck for real-time applications. This phenomenon, often described as the impedance mismatch between row-oriented storage and vector-access patterns [11], underscores the critical need for specialised access methods.

### Contributions and Organizations

This study makes two core contributions. First, it provides a component-level empirical characterisation of MySQL 9.0 Community Edition's native VECTOR execution path, rigorously comparing it with legacy JSON storage and separating the end-to-end cost into transfer, parse, and compute stages. This framing makes it possible to identify where the new binary format helps materially, where the bottleneck remains structural, and why the absence of server-side ANN primitives leaves the Community Edition constrained even after the storage-format improvement. Second, building on that bottleneck analysis, the study evaluates two established application-side optimisation techniques in this MySQL 9.0 setting as practical deployment strategies rather than algorithmic novelties. Hybrid Search is examined as a way to use existing B-Tree indexes on scalar columns to shrink the candidate set before vector computation, while Two-Pass PCA Search is examined as a way to reduce network I/O during the initial scan when selective pre-filtering is unavailable. The resulting synthesis is a practical decision framework, developed in Section 5, for choosing among exact full scans, scalar pre-filtering, PCA compression, and migration to a system with native ANN execution.

## BACKGROUND AND RELATED WORK

### Evolution of Similarity Search

Similarity search in high-dimensional spaces is difficult because classical spatial indexes such as R-trees [15] and KD-trees [16] degrade rapidly as dimensionality increases. This is the familiar Curse of Dimensionality [17], [18]: partitions become less informative, distances concentrate [19], and exact nearest-neighbour search tends toward expensive linear scanning. As a result, modern systems typically rely on Approximate Nearest Neighbour

(ANN) methods, which trade a small amount of accuracy for much lower query cost. The main ANN families include hashing and quantisation methods such as LSH [20], PQ [21], and ScaNN [22], as well as graph-based methods, of which HNSW [23], [24] has become the dominant in-memory baseline because it offers a strong empirical trade-off between accuracy and latency. For larger-than-memory workloads, disk-aware designs such as DiskANN [25] and SPANN [26] extend ANN search to SSD-backed search. For this paper, the key background point is therefore not the full ANN taxonomy, but the distinction between systems that can support ANN execution inside the database engine and MySQL 9.0 Community Edition, which does not.

### Vector Search in Relational Databases

Integrating ANN structures into transactional RDBMSs poses unique engineering challenges, primarily due to the conflict between the random-access patterns of graph traversals and the page-oriented I/O models of traditional storage engines. Furthermore, maintaining ACID compliance for vector indexes is non-trivial; index updates must be atomic and durable, requiring complex Write-Ahead Logging (WAL) integration [27] to prevent corruption during crashes. PostgreSQL has successfully navigated these challenges with the pgvector extension [9], which leverages the database's extensible index access method interface to provide first-class support for HNSW and IVF-Flat indexes. This integration allows vector operations to participate in standard transactions and query planning. In stark contrast, the MySQL ecosystem has historically lacked equivalent native capabilities. Before version 9.0, developers were forced to rely on a fragmented landscape of solutions: external vector databases that compromise transactional consistency, proprietary cloud-native extensions such as Alibaba's AnalyticDB-V [11], [28] that are not available in the open-source kernel, or experimental forks that lag the official release cycle.

### The Impedance Mismatch

A fundamental architectural hurdle in retrofitting vector search into traditional row-oriented RDBMSs lies in the discordance between the storage engine's memory management and the access patterns inherent to high-dimensional similarity search. MySQL's InnoDB storage engine relies on a buffer pool managed as a cache of fixed-size pages (defaulting to 16KB). This design is highly optimised for B-Tree [30] index traversals and range scans, where data exhibits strong spatial locality; accessing one record often implies that adjacent records on the same page will be accessed shortly thereafter [29], [30].

In contrast, vector search algorithms, particularly graph-based methods like HNSW, exhibit stochastic memory access patterns. Traversing the proximity graph involves jumping between disparate nodes that are unlikely to be physically contiguous on disk. This behaviour triggers severe read amplification: to retrieve a single vector (e.g., 512 bytes for D = 128), the engine must load an entire 16KB page into memory. This inefficiency not only saturates I/O bandwidth but also pollutes the buffer pool. The Least Recently Used (LRU) eviction policy [31], designed to keep hot pages in memory, struggles to distinguish between truly popular data and the transient pages loaded during a vector search traversal, potentially evicting critical transactional data and degrading overall system performance.

### MySQL 9.0 Vector Architecture

MySQL 9.0 marks a paradigm shift with the introduction of the native VECTOR data type, a first-class binary structure designed specifically for high-dimensional embeddings. Unlike legacy workarounds that utilised JSON arrays or BLOB fields, the VECTOR type enforces strict typing and efficient storage. Internally, vectors are serialised as contiguous arrays of 4-byte single-precision floating-point numbers (IEEE 754), stored in little-endian format [13]. This binary representation yields substantial storage dividends: a 1536-dimensional embedding from OpenAI's text-embedding-3-small model consumes exactly 6,144 bytes, whereas a corresponding JSON text representation (e.g., [0.123, -0.456, ...]) can easily exceed 15KB depending on floating-point precision. This results in a storage penalty of approximately 2.5x and incurs significant CPU overhead for text parsing during retrieval.

Despite these advances in storage, the functional ecosystem remains sharply bifurcated. Oracle has adopted a cloud-first strategy, reserving critical execution capabilities, specifically the VECTOR_INDEX statement for ANN search and server-side scalar functions like DISTANCE() and COSINE_SIMILARITY(), exclusively for MySQL HeatWave, its proprietary managed cloud offering [14]. This leaves the open-source Community Edition in a precarious position: while it can store vectors efficiently, it lacks the primitives to query them effectively. Consequently, developers are forced to adopt a client-side search pattern, in which the database serves merely as a passive storage layer. To perform a single similarity search, the application must retrieve the entire dataset of raw binary vectors over the network, deserialise them, and compute distances in application memory. This architecture inverts standard database principles, shifting the bottleneck from the database CPU to network bandwidth and client processing power, effectively rendering the system unscalable for production datasets [32].

### Related Optimisation Techniques

Given the limitations of the MySQL 9.0 Community Edition, specifically the lack of server-side ANN indexing, developers must rely on application-side strategies to achieve acceptable performance. Two primary classes of optimisation are relevant: search-space pruning (Hybrid Search) and data-volume reduction (Dimensionality Reduction).

Hybrid Search (Pre-filtering): This strategy leverages the RDBMS's mature B-Tree indexing to filter the dataset based on scalar attributes (e.g., tenant ID, category, timestamp) before performing vector comparisons. By applying a highly selective filter (e.g., WHERE user_id = 123), the candidate set N is reduced from the total table size to a much smaller subset. If N filtered is sufficiently small (e.g., < 10, 000 vectors), a brute-force linear scan becomes computationally feasible even without specialised vector indexes. This approach effectively bypasses the impedance mismatch by loading only relevant data pages into the buffer pool. However, its effectiveness depends on the filter's selectivity; for broad queries where N filtered ≈ N, performance degrades to that of a full-table scan.

Dimensionality Reduction (PCA): When scalar filtering is not applicable or insufficiently selective, reducing the vector payload size becomes necessary to mitigate network bottlenecks. Principal Component Analysis (PCA)[33] projects high-dimensional vectors (e.g., D = 1536) into a lower-dimensional subspace (e.g., D′ = 32) that maximises variance. The theoretical foundation for such projections rests on the Johnson-Lindenstrauss

lemma [34], which guarantees that pairwise distances can be approximately preserved when projecting into a sufficiently large subspace. In a client-side search architecture, this enables a Two-Pass retrieval strategy: first, the client fetches the compressed vectors to perform a coarse global ranking; second, it retrieves the full-precision vectors only for the top-k candidates for re-ranking. This technique directly addresses network bandwidth saturation by reducing the data transfer volume by a factor of $D/D'$, albeit at the cost of an additional network round-trip and the potential loss of recall due to compression artefacts. Recent work on Matryoshka Representation Learning [35] has demonstrated that embeddings can be trained to be effective at multiple dimensionalities natively, offering a learned alternative to post-hoc PCA projection.

To the best of current knowledge, prior work has neither systematically benchmarked the native VECTOR type in MySQL 9.0 Community Edition nor examined application-side optimisation strategies to address the absence of server-side indexing. Existing evaluations have focused on proprietary managed offerings (e.g., MySQL HeatWave [14], PolarDB [11]) or on PostgreSQL's pgvector extension [9], leaving a significant gap in the understanding of the open-source MySQL ecosystem's vector capabilities.

## METHODOLOGY

To evaluate the efficacy of MySQL 9.0's native vector capabilities in a production-like context, a comprehensive suite of experiments was designed. The methodology focuses on three critical performance dimensions: query latency, system throughput (Queries Per Second), and storage efficiency. The study follows the rigorous testing principles outlined in the standard ANN benchmark framework [36] to ensure reproducibility and comparability. The experimental design is structured to isolate the impact of the new binary storage format relative to legacy JSON implementations and to quantify the performance of selected application-side optimisation techniques across varying data scales and dimensionalities. Figure 1 illustrates the overall system architecture, showing the client-side search pattern in which the application retrieves raw vectors from MySQL and computes distances locally.

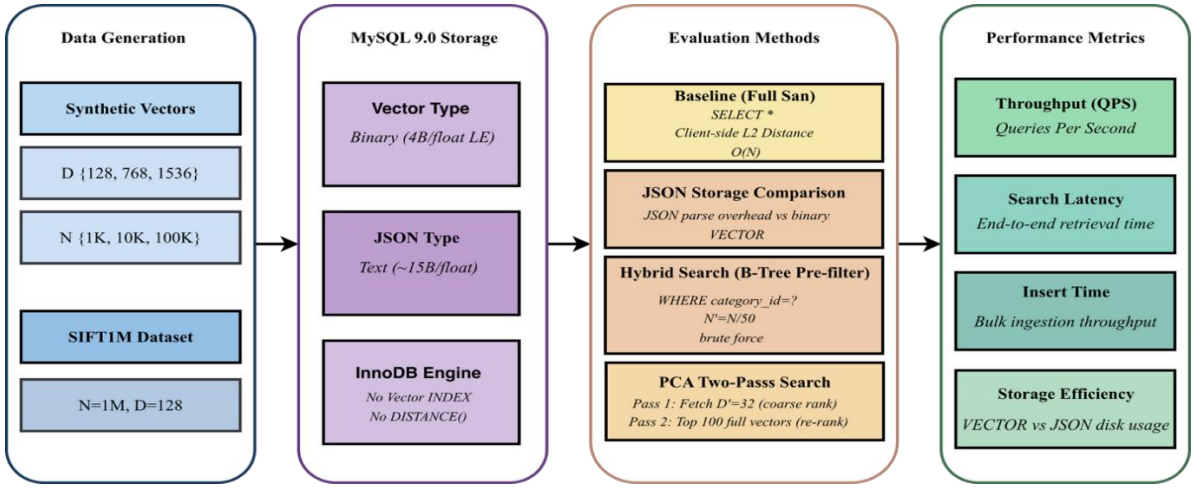


**Figure 1: System Architecture Overview. The client-side search pattern retrieves raw vectors from MySQL and computes distances in application memory.**

**Experiment Setup**

All experiments were conducted in a controlled cloud environment hosted on a Hostinger VPS running Ubuntu 24.04 LTS. The server infrastructure was powered by an AMD EPYC 9354P Processor (2 vCPUs), 8GB of RAM, and a 100GB SSD. The database instance ran MySQL 9.0.0 Innovation Release (Community Edition) within a Docker container. The InnoDB buffer pool was explicitly configured to 6GB; this sizing ensures that the working set remains memory-resident for smaller datasets (N = 10^4) while inducing realistic I/O pressure for larger, high-dimensional datasets (N = 10^5).

The benchmarking client was executed locally on the same server to minimise variation in network latency. Note that running MySQL within a Docker container may introduce minor I/O overhead compared to bare-metal installations [37]; this is considered an acceptable trade-off for reproducibility. A Python 3.12 environment with MySQL Connector/Python (v9.5.0) for connectivity and NumPy (v2.3.5) for vectorised operations was used. For client-side distance computation at scale, libraries such as FAISS [38], [39] could further accelerate brute-force search; however, NumPy was used to isolate the database access overhead from highly optimised GPU/SIMD computation.

To evaluate performance across varying levels of complexity, synthetic datasets of random floating-point vectors were generated from the uniform distribution U (−1, 1). These data are useful for controlled latency and throughput stress tests because they allow dimensionality and table size to be varied systematically. However, they should not be treated as realistic proxies for transformer embeddings. In high-dimensional spaces, uniformly distributed vectors exhibit strong distance concentration, so pairwise distances cluster within a narrow range; consequently, conclusions about storage overhead, transfer cost, and brute-force latency remain informative, whereas accuracy-related conclusions, especially the recall behaviour of PCA compression, are less likely to transfer directly to real embedding workloads. The evaluation covered three dimensionality settings ($D \in \{128, 768, 1536\}$) corresponding to SIFT descriptors, BERT, and modern LLM embeddings, with dataset sizes scaling from N = 10^3 to N = 10^5 vectors. The dimensionality D = 768 corresponds to the output of BERT-based encoders [40], while D = 1536 matches modern commercial embedding APIs [41]. Additionally, to broaden external validity beyond synthetic data, the SIFT1M dataset [21], a standard benchmark comprising one million 128-dimensional SIFT vectors, was incorporated. SIFT1M provides a realistic benchmark distribution for large-scale nearest-neighbour search, but it should still be interpreted cautiously because SIFT descriptors differ substantially from sentence-transformer embeddings in terms of geometry and semantic structure. To further probe transferability, a small-scale semantic-text validation was also conducted using 5,000 natural-language sentences extracted from the 20 Newsgroups corpus, which were encoded with the sentence-transformer model all-MiniLM-L6-v2 (384 dimensions). Using 100 held-out queries, we evaluated the PCA two-pass strategy at 16, 32, and 64 dimensions against exact Recall@10. This was then extended with a PCA sensitivity study at 64 dimensions, varying the PCA training set size from 1,000 to 10,000 examples and testing how a projection trained on one semantic domain transfers to another. Finally, to address the practical limitation of single-threaded evaluation, a basic concurrency validation was conducted on a representative high-dimensional workload ($N = 10^5$, D = 1536) using 1, 4, and 8 simultaneous clients for the Baseline, Hybrid Search, and PCA execution patterns.

**Experiment Design**

Four distinct experimental scenarios are defined to isolate specific performance characteristics:

***Baseline Performance (Full Scan)***

This scenario establishes the worst-case performance baseline for exact nearest neighbour search. A k-NN query (k=1) is executed by retrieving the entire dataset to the client via a full table scan (SELECT id, vector_col FROM table). The retrieval process involves fetching the raw binary data, deserialising it into NumPy arrays, and computing Euclidean distances entirely within application memory. Finally, the client sorts the distances to identify the top-k candidates. This method has a time complexity of O(N) and is expected to be bound by network bandwidth and the client's CPU.

***Storage Efficiency Comparison***

The storage footprint and retrieval latency of the native VECTOR type are compared with those of the legacy JSON data type. Disk usage is quantified by querying the data_length from information_schema.tables, while fetch latency is measured as the wall-clock time required to retrieve all N rows into the client application. The expectation is that the binary VECTOR type will significantly outperform JSON by eliminating text parsing overhead and leveraging a compact 4-byte floating-point representation.

***Hybrid Search Evaluation***

To evaluate the impact of search space pruning, a synthetic scalar column, category_id, is added with a cardinality of 50, and indexed using a standard B-Tree [42]. The search query is modified to:

*SELECT id, vector_col*

*FROM table*

*WHERE category_id = ?*

This effectively reduces the search space to N’≈ N/50. The end-to-end latency of this filtered query is measured to quantify the speedup gained by leveraging existing relational indexes. Importantly, this method is exact only within the filtered subset defined by the scalar predicate. If the true nearest neighbour in the full dataset lies outside the selected category_id, Hybrid Search will not retrieve it; consequently, its global recall depends on whether the scalar filter is semantically aligned with the retrieval task.

***PCA-Based Two-Pass Search Evaluation***

To minimise network I/O, a dimensionality reduction strategy based on Principal Component Analysis (PCA) is evaluated. The workflow begins with an offline training phase in which a PCA model learns a projection matrix from a subset of 10,000 vectors, reducing the dimensionality from D to D′ = 32. In the database schema, both the full-precision vector (D)

and the compressed representation (D′) are stored. The retrieval process operates in two phases: first, the client fetches all reduced vectors (D′ = 32) to compute approximate distances and identify the top 100 candidates (Coarse Filter). Second, it retrieves the full vectors (D) for only these candidates by primary key to perform exact re-ranking (Fine Re-ranking). Although this strategy introduces an additional network round-trip, it achieves a substantial reduction in data transfer volume during the initial scan, by a factor of D/D′ (e.g., 48x for D = 1536, D′ = 32).

The synthetic scaling benchmarks were repeated five times per configuration. The median latency over the five trials is reported; error bars in figures indicate the 95% bootstrap confidence interval of the median (5,000 resamples). The median was preferred over the arithmetic mean to reduce sensitivity to occasional cold-start outliers in the first trial. Tables 6 and 7 report the same median-based summaries with their 95% bootstrap confidence intervals. The main exceptions are the targeted validation workloads collected under different protocols: Table 1 is a single instrumented SIFT1M run, because its purpose is stage-level decomposition of one million-row exact scans rather than repeated small-sample inference; Tables 2-5 summarise fixed-query auxiliary evaluations rather than the five-trial benchmark design; and Table 8 reports one synchronised concurrency run per client count, because the quantity of interest is within-run contention among simultaneously active clients. More specifically, Table 2 reports the mean latency over five warmed query repetitions per selectivity setting, Table 3 reports point estimates over 50 held-out SIFT1M queries, and Tables 4 and 5 report point estimates over fixed 100-query sentence-embedding evaluations.

## EVALUATION

This section presents the results of the performance evaluation, comparing the native VECTOR type against legacy JSON storage and analysing the effectiveness of the selected established application-side optimisation techniques.

### Throughput Analysis

Figure 2 illustrates the Queries Per Second (QPS) achieved by each method across varying dataset sizes (N) and dimensions (D). The results highlight a stark contrast between the evaluated application-side techniques and the baseline approaches.

The Hybrid Search strategy consistently outperforms all other methods by a significant margin, achieving a peak throughput of 770 QPS for smaller datasets (N = 1000, D = 128). This performance advantage is directly attributable to the B-Tree index's high selectivity on the category_id column. By filtering the dataset to approximately 2% of its original size (N/50), the database engine avoids scanning the vast majority of vector pages, effectively bypassing the I/O bottleneck that plagues full-table scan methods. As the dimensionality increases to D = 1536, the throughput of the Baseline and JSON methods collapses due to the linear growth in data volume. The Baseline method drops to 0.11 QPS at N = 10^5, rendering it unusable for real-time applications. In contrast, the PCA-based approach demonstrates resilience to high dimensionality. Operating primarily on compressed 32-dimensional vectors, it maintains a throughput of 1.55 QPS, representing a 15.5x improvement over the baseline in the D = 1536, N = 10^5 setting. This supports the

interpretation that network bandwidth, rather than CPU compute time, is the primary bottleneck for high-dimensional client-side search.

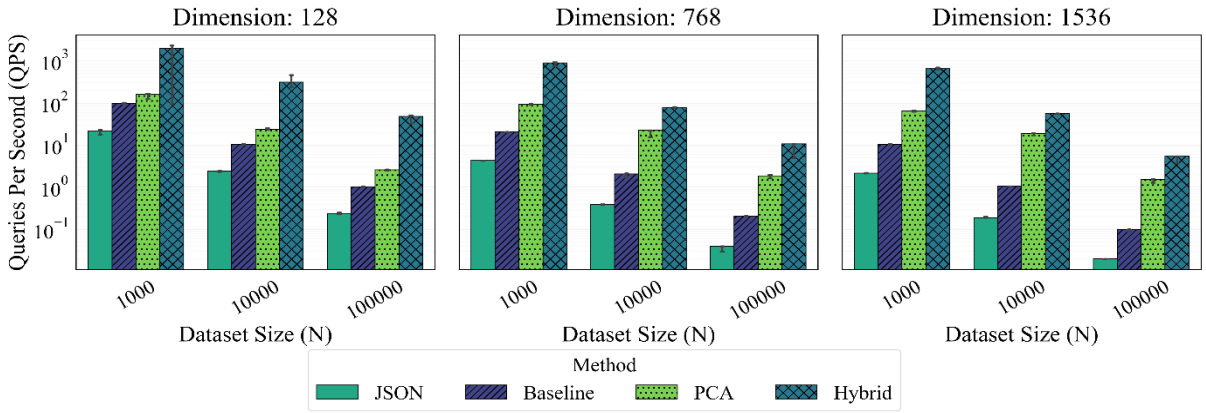


**Figure 2: Throughput (QPS) Comparison. Hybrid Search consistently outperforms other methods.**

**Real-World Dataset Validation (SIFT1M)**

To address critiques about the realism of synthetic data, the evaluation was extended to the SIFT1M dataset, a standard benchmark comprising 1 million 128-dimensional SIFT vectors. This experiment validates the findings on a significantly larger scale (N = 10^6).

Table 1 reports the SIFT1M results for the MySQL configurations using two measures: data preparation time, defined as the combined fetch and parse cost, and total end-to-end latency. The native VECTOR type retains a clear advantage over JSON, reducing data preparation time from 48.94s to 11.39s, a 4.3x improvement, thereby confirming that the binary representation substantially lowers transfer and deserialization overhead at SIFT1M scale. The comparison with PostgreSQL nevertheless highlights the central limitation of the current MySQL Community Edition design. PostgreSQL, using pgvector with an HNSW index, achieved an average query latency of 0.022s (22ms), whereas MySQL's brute-force scan required 13.24s, a gap of nearly 600x. The implication is that storage-format improvements alone do not remove the scalability constraint imposed by the absence of server-side ANN execution.

**Table 1: SIFT1M benchmark results (N = 10^6, D = 128).**

| Metric | MySQL Vector | MySQL JSON | PostgreSQL (HNSW) |
|---|---|---|---|
| Data Preparation (Fetch + Parse) | 11.39s | 48.94s | N/A |
| Total Latency | 13.24s | 49.66s | 0.02s |

The PostgreSQL entry is shown only as a same-host ANN reference: pgvector+HNSW performs server-side search, so fetch+parse is not applicable. In the archived pgvector 0.8.1 benchmark (M = 32, ef_construction = 400), ef_search = 400 reached Recall@10 = 0.9992 with 21.08ms median latency over 10,000 SIFT1M queries; MySQL ran in Docker, whereas PostgreSQL was accessed as a native service on the same host.

### Sensitivity Analysis

The robustness of the Hybrid Search strategy was further analysed by varying the filter selectivity from 1% to 100% on the SIFT1M dataset. Table 2 and Figure 3 present the results. At 1% selectivity (N_filtered = 10,000), the query completes in 1.44s. As selectivity increases, latency generally increases, reaching 12.43s for a full-table scan (100%). Non-monotonic behaviour is observed at intermediate selectivities (e.g., 10% is slightly faster than 5%), which we attribute to InnoDB buffer pool caching: once a critical mass of pages is loaded for the 5% query, subsequent queries at 10% benefit from a warm cache. Despite this variance, the overall trend confirms that Hybrid Search is most effective for highly selective queries (typically < 2–3%) on large datasets.

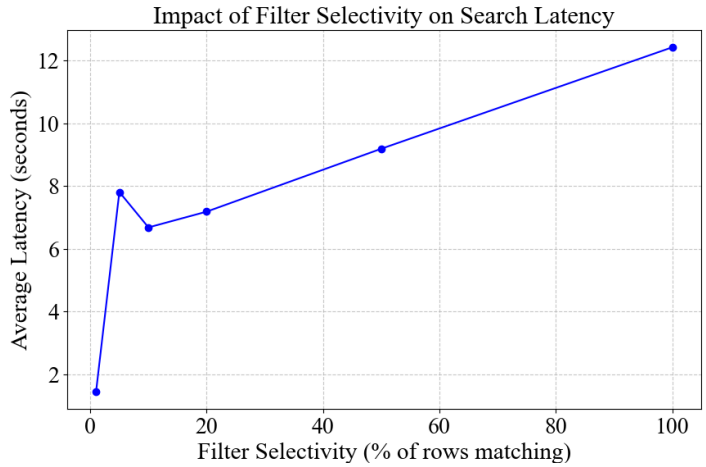


**Figure 3: Sensitivity Analysis: Hybrid Search latency vs. filter selectivity on SIFT1M.**

**Table 2: Sensitivity Analysis (SIFT1M; mean latency over five warmed query repetitions per selectivity setting).**

| Sensitivity % | Latency(s) |
|---|---|
| 1% | 1.44 |
| 5% | 7.81 |
| 10% | 6.68 |
| 20% | 7.18 |
| 50% | 9.19 |
| 100% (Full Scan) | 12.43 |

These latency gains, however, should be interpreted in light of an important limitation in retrieval. Hybrid Search does not approximate the global nearest-neighbour problem; instead, it redefines the candidate set through the scalar predicate and then performs exact search only within that subset. Consequently, excellent latency is obtained only when the filter is both selective and relevant. If the predicate excludes vectors that are semantically close to the query, the method can miss the true nearest neighbour entirely. For this reason, Hybrid Search is best viewed as an optimisation for workloads in which scalar constraints are part of the application semantics, rather than as a general substitute for ANN indexing over the full corpus.

### PCA Accuracy Trade-off

Finally, the accuracy loss introduced by the PCA optimisation strategy was quantified. The 128-dimensional SIFT vectors were reduced to 16-, 32-, and 64-dimensional representations, and Recall@10 was measured against the exact ground truth. Table 3 reveals a significant trade-off. Reducing dimensions to 16 yields the lowest latency (0.049s) but suffers from poor accuracy (Recall = 0.33). Retaining 64 dimensions yields the best results in this experiment, achieving 79% recall while reducing latency by approximately 25% compared to the full-dimensional scan. It should be emphasised, however, that this should be read as a dataset-specific result on SIFT descriptors rather than as a general claim that 2x compression is broadly safe for modern text embeddings. The recall sensitivity of PCA depends strongly on the geometry of the underlying embedding space, which is why the additional sentence-transformer validation is reported in Table 4 below.

**Table 3: PCA accuracy vs latency trade-off (point estimates over 50 held-out SIFT1M queries)**

| Dimensions | Recall@10 | Latency (s) |
|---|---|---|
| 16 | 0.326 | 0.049 |
| 32 | 0.542 | 0.065 |
| 64 | 0.788 | 0.097 |
| 128 (Baseline) | 1 | 0.130 |

### Sentence-Transformer Validation

To test whether the PCA trend observed on SIFT1M transfers to semantic embeddings, we ran a small validation on 5,000 natural-language sentences extracted from the 20 Newsgroups corpus and encoded them using the all-MiniLM-L6-v2 sentence-transformer model (384 dimensions).

**Table 4: Validation on sentence-transformer embeddings (5,000 all-MiniLM-L6-v2 sentence vectors, 100 queries; point estimates over a fixed query set).**

| Dimensions | Recall@10 | Latency (s) |
|---|---|---|
| 16 | 0.569 | 0.00015 |
| 32 | 0.784 | 0.00019 |
| 64 | 0.936 | 0.00025 |
| 384 (Baseline) | 1 | 0.00075 |

We used 100 held-out sentence queries and evaluated the same two-pass PCA procedure with 100 re-ranked candidates. Table 4 shows a qualitatively similar but quantitatively stronger pattern than SIFT1M: aggressive compression to 16 dimensions reduced Recall@10 to 0.569, while 64 dimensions preserved 0.936 Recall@10 with latency still below the full-dimensional baseline. This result does not eliminate the external validity concern, because the corpus remains small and the model family consists of only one representative semantic encoder, but it does show that the central PCA trade-off is not confined to SIFT descriptors.

**PCA Training Sensitivity and Transfer**

The sensitivity of the PCA projection to the size and origin of the training set was next examined. Using the same all-MiniLM-L6-v2 embedding family, the reduced dimensionality was fixed at 64, and the PCA training set size was varied across 1,000, 5,000, and 10,000 in-domain examples. Table 5 shows that the retrieval outcome is already fairly stable once the training set reaches a few thousand points: Recall@10 changes only marginally from 0.905 at 5,000 training vectors to 0.902 at 10,000, and the learned subspaces are nearly identical (subspace similarity 0.978 relative to the 10,000-vector reference). By contrast, the 1,000-vector PCA is noticeably less stable, with lower subspace similarity (0.858) and a modest drop in recall to 0.898. Cross-domain transfer was also tested by training PCA on a technical/scientific subset of 20 Newsgroups and applying it to a different recreation/politics/religion subset. The matched-domain projection achieved Recall@10 = 0.931, whereas the cross-domain projection fell to 0.775, with subspace similarity 0.661. This suggests that the PCA matrix generalises reasonably within a stable embedding distribution but should not be assumed to transfer unchanged across semantically different corpora.

**Table 5: PCA training-size sensitivity and cross-domain transfer at 64 dimensions on all-MiniLM-L6-v2 embeddings (point estimates over fixed 100-query evaluations).**

| Condition | Recall@10 | Latency (s) | Variance Retained | Subspace Similarity |
|---|---|---|---|---|
| Train size =1,000 | 0.898 | 0.00025 | 0.600 | 0.858 |
| Train size=5,000 | 0.905 | 0.00025 | 0.563 | 0.978 |
| Train size=10,000 | 0.902 | 0.00025 | 0.560 | 1.000 |
| Cross-domain transfer | 0.775 | 0.00026 | 0.560 | 0.661 |
| Matched-domain transfer | 0.931 | 0.00024 | 0.575 | 1.000 |

**Search Latency Comparison**

Figure 4 details the total search time (latency) for a single query. The results unequivocally demonstrate the inefficiency of legacy text-based storage. The JSON method exhibits the highest latency across datasets when parsing JSON strings into floating-point arrays. For N = 10^5 and D = 1536, a single JSON query takes over 55 seconds, whereas the native VECTOR type completes in 10.14 seconds, a 5.4x speedup. Despite this improvement, the native VECTOR baseline still scales O(N) with dataset size linearly, making it unsuitable for interactive applications as data volume grows. Hybrid Search achieves the lowest absolute latency by effectively reducing the problem size. By leveraging the B-Tree index to filter out 98% of the rows, the client only needs to fetch and process 2,000 vectors instead of 100,000, resulting in sub-second response times even for high-dimensional data.

The PCA method presents an interesting trade-off. At lower dimensions (D = 128), the overhead of the two-pass network round-trip outweighs the bandwidth savings, resulting in higher latency than the baseline. However, a crossover point is observed as dimensionality increases. At D = 1536, the massive reduction in data transfer (fetching 32 floats instead of

1536 in the first pass) compensates for the extra round-trip, making PCA a viable strategy for high-dimensional datasets where scalar filtering is unavailable.

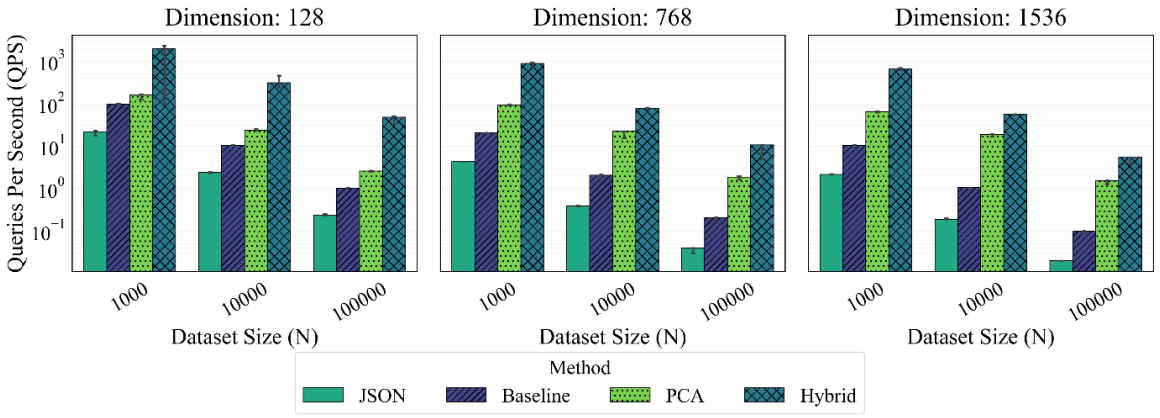


**Figure 4: Search Time Comparison (Log Scale). JSON incurs the highest latency, while Hybrid Search minimises retrieval time**

### Insertion Overhead

Figure 5 compares the insertion performance across varying dimensions. Unlike search operations, which are read-intensive, insertion throughput in a transactional RDBMS is heavily influenced by network round-trip time (RTT) and transaction commit overhead. In the single-threaded benchmark, the native VECTOR type demonstrates a consistent but modest advantage over the JSON method. For D = 1536, the binary format reduces the payload size by approximately 60%, decreasing the time spent on network transmission. More importantly, it eliminates the server-side parsing overhead required to validate and tokenise JSON text strings. However, the performance gap is narrower than observed in retrieval tasks (approximately 1.2x-1.5x speedup) because the total transaction time is dominated by the fixed costs of WAL (Write-Ahead Logging) synchronisation [31] and disk I/O, which mask the efficiency gains of the binary protocol. Nevertheless, in bulk-loading scenarios, the reduced storage footprint of the VECTOR type—6KB per row versus 15KB for JSON—results in significantly lower buffer pool pressure and faster checkpointing.

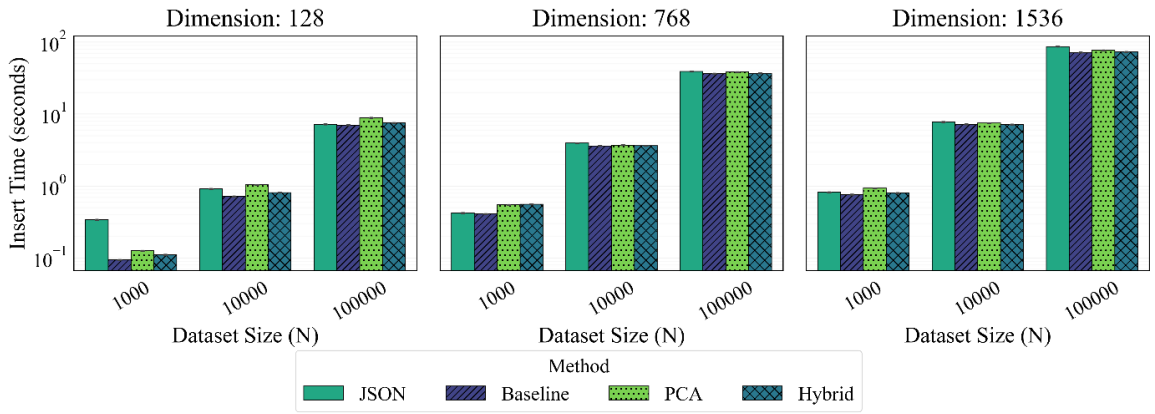


**Figure 5: Insertion Time Comparison (Log Scale). Binary VECTOR storage offers slight improvements over JSON text storage.**

**Numerical Summary**

Tables 6 and 7 summarise the key numerical findings.

**Table 6: Search Latency: Vector VS JSON (N=10^5; median [95% bootstrap CI] over 5 trials).**

| Dim (D) | Vector (s) | JSON (s) | Speedup |
|---|---|---|---|
| 128 | 0.96 [0.94, 0.97] | 4.16 [4.01, 4.31] | 4.3x |
| 768 | 4.72 [4.70, 4.94] | 27.67 [27.05, 29.24] | 5.9x |
| 1536 | 10.14 [9.66, 10.64] | 55.11 [54.73, 60.26] | 5.4x |

The results demonstrate a significant and consistent performance advantage for the native VECTOR type across all tested dimensions. The strongest observed gain in Table 6 is 5.9x at D = 768, while the high-dimensional D = 1536 setting still shows a 5.4x reduction in latency relative to JSON. This confirms that the overhead of JSON serialisation becomes increasingly punitive as payload size grows. The speedup is primarily driven by two factors: reduced network bandwidth consumption from the compact IEEE 754 binary representation (approximately 2.5x smaller than the text-based JSON equivalent), and the elimination of the CPU-intensive JSON parsing step on the client side.

Because this comparison holds the retrieval task fixed and changes only the storage representation, the Table 6 speedups should be interpreted primarily as reductions in storage format and transfer overhead. That mechanism is substantially less sensitive to the underlying vector distribution than recall-oriented results are, so the direction of the advantage is expected to generalise even though the exact constants may shift on real embedding workloads.

However, it is crucial to note that while the native type significantly lowers the constant factor of the linear scan, the asymptotic complexity remains O(N). Consequently, for datasets with more than 10^5 vectors, even the optimised binary fetch becomes prohibitively slow for interactive use cases, underscoring the need for the algorithmic optimisations discussed below.

**Table 7: Throughput (QPS) Comparison (N=10^5 median [95% bootstrap CI] over 5 trials).**

| Dim | Baseline | PCA (Two-Pass) | Hybrid (Filter) |
|---|---|---|---|
| 128 | 1.04 [1.03, 1.06] | 2.60 [2.51, 2.65] (2.5x) | 48.31 [31.89, 51.62] (46x) |
| 768 | 0.21 [0.20, 0.21] | 1.88 [1.64, 1.99] (9x) | 10.72 [5.01, 10.78] (51x) |
| 1536 | 0.10 [0.09, 0.10] | 1.55 [1.28, 1.58] (15.5x) | 5.44 [3.73, 5.46] (54x) |

The optimisations on the application side provided substantial throughput gains, validating their necessity in the absence of server-side indexing. Hybrid Search emerged as the most effective strategy overall, yielding speedups of 46x-54x across the representative settings in Table 7. This massive improvement confirms that using mature B-Tree indexes to prune the search space is the most reliable method for scaling vector search in MySQL 9.0 today, given the availability of selective metadata predicates. In scenarios where metadata

filtering is not applicable, the PCA strategy proved robust in terms of latency, particularly for high-dimensional data. Although its benefit was moderate at lower dimensions (2.5x speedup for D = 128) due to the fixed latency of the two-pass network protocol, its efficiency scaled super-linearly with vector size. For D = 1536, PCA achieved a 15.5x speedup. Table 7 gains should likewise be read primarily as reductions in bytes transferred and candidates scored under MySQL's client-side execution model: Hybrid Search shrinks the candidate set before vector transfer, whereas PCA shrinks the first-pass payload before exact re-ranking. The exact ratio will depend on selectivity and embedding geometry, but the source of the gain is overhead reduction rather than an intrinsic improvement in retrieval quality. It is worth noting that Hybrid Search is exact only with respect to the filtered subset, not necessarily with respect to the full dataset: Recall@k = 1.0 holds conditionally once the predicate has selected the candidate set, but global recall can be lower if the scalar filter excludes semantically relevant vectors. By contrast, PCA introduces a different recall–latency trade-off through compression. Table 3 shows that this trade-off is substantial on SIFT1M, Table 4 depicts that the same qualitative pattern also appears on a small sentence-transformer workload, and Table 5 shows that the learned PCA projection becomes quite stable once the training set reaches roughly 5,000 examples but transfers less effectively across semantic domains.

**Concurrency Validation**

Because production deployments rarely serve a single query at a time, a basic concurrency validation was conducted on a representative high-dimensional workload with N = 10^5 and D = 1536. Three execution patterns, the Baseline full scan, Hybrid Search, and PCA Two-Pass Search, were measured under 1, 4, and 8 concurrent clients. Each row in Table 8 corresponds to a single synchronised concurrent run, in which each client issued one query; the reported mean and p95 statistics are therefore computed from the client latencies observed within that run, and the aggregate throughput is computed from the full makespan of that same run.

**Table 8: Basic concurrency validation on a representative workload ($N = 10^5$, D = 1536; single synchronised run, one query per client).**

| Method | Clients | Mean Latency (s) | p95 Latency (s) | Aggregate QPS |
|---|---|---|---|---|
| Baseline | 1 | 9.02 | 9.02 | 0.11 |
| Baseline | 4 | 43.76 | 45.91 | 0.09 |
| Baseline | 8 | 229.52 | 246.76 | 0.03 |
| Hybrid | 1 | 0.83 | 0.83 | 1.13 |
| Hybrid | 4 | 0.98 | 1.09 | 3.60 |
| Hybrid | 8 | 1.27 | 1.61 | 4.82 |
| PCA | 1 | 1.11 | 1.11 | 0.89 |
| PCA | 4 | 2.84 | 3.17 | 1.26 |
| PCA | 8 | 8.16 | 9.67 | 0.83 |

Table 8 shows a clear separation among the methods. The Baseline execution pattern degrades sharply under contention: mean latency increases from 9.02s with one client to 43.76s with four clients and 229.52s with eight clients, while aggregate throughput falls

from 0.11 QPS to 0.03 QPS. This confirms that the client-side full-scan design does not merely remain slow under concurrency; it collapses as competing queries contend for the same buffer pool, network bandwidth, and client-side computation. Hybrid Search behaves much more favourably. Its mean latency rises only modestly from 0.83s at one client to 1.27s at eight clients, while aggregate throughput increases from 1.13 QPS to 4.82 QPS. PCA sits between these extremes: it is more concurrency-tolerant than the Baseline but still shows noticeable latency inflation at eight clients (8.16s mean, 9.67s p95), with aggregate throughput flattening at 0.83 QPS.

## DISCUSSION

The results demonstrate that MySQL 9.0's native VECTOR type represents a meaningful advancement that closes part of the gap between MySQL and systems with mature vector support, such as PostgreSQL with pgvector. The main contribution of the paper, however, is not the rediscovery of Hybrid Search or PCA as optimisation ideas. Rather, it is the quantitative identification of where MySQL 9.0 Community Edition actually spends time under its current client-side search architecture, and the resulting decision framework for choosing the least-bad deployment pattern under those constraints. Several implications follow from that characterisation.

First, the 600x performance gap observed against PostgreSQL with HNSW indexing on the SIFT1M dataset (Table 1) reflects the lack of server-side ANN indexing, not a deficiency in MySQL's storage format. When comparing the client-side data-preparation stage alone, the native VECTOR type remains substantially more efficient than JSON. This distinction is important: MySQL's limitation is algorithmic (O(N) brute-force vs O(log N) graph traversal), not architectural.

Second, the non-monotonic latency behaviour in the sensitivity analysis (Table 2) highlights the complex interaction between query selectivity and InnoDB's buffer pool management. At intermediate selectivities (5–20%), the working set exceeds the buffer pool's capacity to efficiently cache vector pages, leading to unpredictable eviction patterns. This observation reinforces the impedance mismatch discussed in Section 2 and suggests that careful buffer pool tuning, or the use of dedicated buffer pool instances for vector tables, could yield further improvements.

Third, the PCA sensitivity results in Table 5 clarify two previously implicit issues. First, the learned projection matrix is reasonably stable once the PCA training set reaches a few thousand examples: moving from 5,000 to 10,000 training vectors changes Recall@10 only from 0.905 to 0.902, yielding near-identical subspaces. This suggests that the original choice of 10,000 training vectors was conservative rather than precarious. Second, the same experiment shows a clear limit to generalisation across data distributions. When the PCA matrix is trained on one semantic domain and applied to another, Recall@10 drops from 0.931 to 0.775, indicating that the projection should be retrained or at least revalidated when the embedding population changes materially.

Fourth, the concurrency results in Table 8 add an important operational qualification. The single-client measurements already showed that the Baseline method is unsuitable for interactive use at scale, but the multi-client results demonstrate that this weakness compounds under contention rather than averaging out. The unindexed full scan

exhibits superlinear latency inflation and decreasing aggregate throughput as the client count rises, exactly the pattern one would expect when every request forces a large candidate transfer and client-side distance computation. By contrast, Hybrid Search scales comparatively well across 1, 4, and 8 clients because the scalar predicate bounds the working set before fetching vector payloads. PCA remains viable under moderate concurrency, but its two-pass design still incurs significant transfer and re-ranking work, resulting in throughput that flattens as client count increases. In practical terms, this means that Hybrid Search is not merely the fastest single-query workaround; it is also the most resilient of the evaluated application-side strategies under modest concurrent load.

Fifth, regarding the practical applicability of the evaluated optimisation techniques: Hybrid Search is highly effective but requires selective scalar predicates, which may not be available in all use cases (e.g., open-ended semantic search across an entire corpus). More importantly, its correctness is conditional on the predicate itself: the method performs an exact brute-force search only within the filtered subset, so if the true nearest neighbour belongs to a different category_id, it is excluded before any vector comparison takes place. Hybrid Search therefore preserves recall only when the scalar filter is part of the intended query semantics or is strongly correlated with the target neighbours. The PCA-based Two-Pass strategy is more general but introduces a non-trivial degradation in recall, particularly at aggressive compression ratios (Table 3). Moreover, this degradation is data-dependent: PCA preserves variance, not semantic neighbourhood structure per se, so recall can shift materially across embedding families. The sentence-transformer validation (Tables 4 and 5) is useful here because it shows both that the qualitative trend persists on semantic text embeddings and that training-set size and domain shift affect the exact outcome. For applications requiring both generality and high recall, a combination of the two techniques, applying PCA within a pre-filtered subset, could be explored.

This leads to a more concrete, practical decision rule that can be expressed as a cost model. Let the main latency terms be defined as follows.

| Symbol | Meaning | Units |
|---|---|---|
| N | Total corpus size | vectors |
| s | Scalar-filter selectivity, so N_filtered = sN | unitless |
| D | Original dimensionality | float/vector |
| D’ | Reduced PCA dimensionality | float/vector |
| kr | Number of full-precision candidates re-ranked in pass two | vectors |
| B | Effective end-to-end transfer bandwidth | bytes/s |
| Cp | Per-element parse cost | s/float |
| Cd | Per-element distance-computation cost | s/float |
| r | Fixed round-trip / statement overhead | s/query |

Under the client-side execution model studied here, the three strategies can be approximated as:

$$L_{baseline} \approx r + N\left(\frac{4D}{B} + D(c_p + c_d)\right) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

$$L_{hybrid} \approx r + sN\left(\frac{4D}{B} + D(c_p + c_d)\right) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$L_{pca} \approx 2r + N\left(\frac{4D'}{B} + D'c_d\right) + k_r\left(\frac{4D}{B} + D(c_p + c_d)\right) \ldots\ldots (3)$$

The first term in each expression captures fixed request overhead, the transfer term reflects the fact that each float occupies 4 bytes, and the remaining terms approximate client-side parse and distance-computation work. For PCA Two-Pass Search, the model assumes that reduced vectors are materialised in the database and that only the top kr candidates are re-fetched at full precision in the second pass.

This model is intentionally simple, but it captures the dominant terms observed in the measurements. Hybrid Search wins when s is small enough that shrinking N dominates all other effects. PCA wins over the Baseline when the first-pass savings exceed the extra second-round transfer, re-ranking, and round-trip terms. Practitioners can therefore estimate B and r from a small pilot run on their own hardware, plug in their expected N, D, s, D', and kr, and obtain a first-order prediction of which strategy should dominate before committing to a full benchmark campaign. The model also clarifies why dimensionality matters so strongly: as D or N grows, transfer and client-side arithmetic scale linearly, whereas the fixed round-trip overhead matters mainly at low D and small N. The first question is therefore not dimensionality but whether the query naturally includes a selective scalar predicate that is already part of the application semantics, for example, tenant_id, category, language, or time window. If the answer is yes, Hybrid Search is usually the best choice because it produces the strongest gains in both single-client and concurrent settings. In these experiments, once filtering reduced the effective candidate set to a small fraction of the corpus, typically a few per cent or less, Hybrid Search delivered 46x-54x throughput improvements (Table 7) and remained stable up to eight concurrent clients (Table 8). The second question is whether the workload is open-ended semantic retrieval over the full corpus. In that case, Hybrid Search is no longer a reliable substitute for ANN, because global recall becomes contingent on the predicate rather than on vector similarity.

When no selective predicate is available, dimensionality and corpus size become the dominant signals. For moderate corpus sizes up to roughly N=10^5, PCA is the more practical workaround when vectors are high-dimensional, especially D=768 or D=1536, because the transfer bottleneck dominates and compression yields clear latency savings. This is exactly where the measured crossover appears in the results: at D = 1536, PCA improved throughput by 15.5x over the baseline (Table 7), while remaining workable under modest concurrency even though it scaled less well than Hybrid Search (Table 8). For lower-dimensional workloads such as D = 128, the advantage is weaker because the two-pass protocol adds a fixed extra round-trip while the raw vector payload is already relatively small. In that regime, PCA should be treated as an optional optimisation with a dataset-specific recall cost rather than as a default design.

If the workload requires open-ended search over large corpora, especially for N> 10^6, or if high global recall and predictable latency are both hard requirements, the evidence in this paper points toward migrating to a system with native ANN execution, such as PostgreSQL with pgvector. The SIFT1M comparison is decisive here: MySQL's unindexed full scan remained in the multi-second range even with native VECTOR storage, whereas PostgreSQL with HNSW operated in the tens of milliseconds (Table 1). That gap is too large

to plausibly close with application-side workarounds alone. The practical implication is that Hybrid Search and PCA are best viewed as deployment strategies for MySQL when migration is undesirable or premature, not as full substitutes for server-side ANN indexing.

Finally, several threats to validity should be acknowledged. Most of the benchmark suite remains single-threaded, and the added concurrency validation in Table 8 is intentionally basic: it covers only one representative workload (N = 10^5, D = 1536), one host configuration, and up to eight concurrent clients. Production systems serving heterogeneous queries or mixed read-write workloads could therefore exhibit different contention patterns. The use of Docker introduces a thin virtualisation layer [37] that may marginally affect I/O performance. A more serious limitation concerns data realism. The synthetic datasets follow a uniform distribution, and in high-dimensional settings, such vectors exhibit distance concentration, meaning that many pairwise distances become nearly indistinguishable. This makes the synthetic workload suitable for measuring storage overhead, transfer cost, and brute-force latency scaling, but much less reliable for drawing general conclusions about retrieval quality. For that reason, the headline figures of 5.9x, 54x, and 15.5x should be interpreted primarily as measurements of storage-format, candidate-set, and transfer-volume reductions under MySQL's client-side architecture, not as universal claims about retrieval effectiveness across arbitrary embedding distributions. The SIFT1M evaluation improves realism for large-scale nearest-neighbour search and supports the storage-overhead interpretation by showing the same qualitative JSON-versus-VECTOR advantage with one million vectors, yet SIFT descriptors are not for semantic retrieval, unlike sentence-transformer embeddings. Sentence-transformer validations were therefore added in Tables 4 and 5, which reduce but do not eliminate this concern: the corpora are still modest in size, only one embedding model is tested, and the transfer experiment covers only one domain split. Accordingly, the PCA results should be interpreted as converging evidence across multiple data regimes rather than as a universal law.

## CONCLUSION

This paper provided a detailed empirical characterisation of the native VECTOR data type in MySQL 9.0 Community Edition. The central finding is not that B-Tree pre-filtering or PCA compression are novel techniques, but that MySQL's current open-source vector stack is governed by a clear bottleneck structure: binary VECTOR storage removes substantial serialisation overhead, yet the absence of server-side indexing and distance functions leaves large workloads dominated by data transfer, client-side parsing, and O(N) execution. In that sense, the paper's main contribution is a component-level account of where the time goes, coupled with an evidence-based framework for deciding which workaround is appropriate under which workload conditions.

The resulting decision rule is straightforward. When queries naturally include selective metadata constraints, Hybrid Search is the best practical strategy because it bounds the candidate set before vector transfer, thereby delivering the strongest gains in both latency and concurrency, with 46x-54x speedups in the evaluated workloads. When no such predicate exists, PCA-based Two-Pass Search becomes the more relevant workaround for moderate corpus sizes and high-dimensional vectors, because compression directly attacks the measured transfer bottleneck; at D = 1536 it delivered a 15.5x speedup, though with a dataset-dependent recall trade-off. The added sentence-transformer validation

shows that this recall-latency pattern is not confined to SIFT descriptors, while the new sensitivity study indicates that the PCA projection is stable once trained on several thousand in-domain examples but transfers materially worse across domains. The simple cost model in Section 5 makes this rule more portable by allowing practitioners to substitute their own bandwidth, selectivity, and dimensionality values rather than extrapolating directly from the benchmark host used here. The added concurrency validation sharpens the same point operationally: under 4-8 concurrent clients, the Baseline full scan degrades severely, PCA remains workable but begins to flatten, and Hybrid Search offers the best combination of latency stability and aggregate throughput.

These results indicate that, while MySQL 9.0 is not yet a direct substitute for specialised vector databases in high-scale scenarios, it now presents a practical option for small-to-medium datasets ($N \leq 10^5$) or highly filtered workloads, especially when concurrent demand is modest and selective metadata predicates are available. Just as importantly, the main speedup claims in this paper should be read with the right scope: they quantify how much overhead can be removed from MySQL's current client-side execution path by using binary storage, scalar pre-filtering, or compressed first-pass retrieval. Even if Oracle later backports ANN indexing into the Community Edition, the analytical contribution of this paper remains useful: the component-level breakdown provides a reusable method for determining whether a vector workload is primarily limited by storage layout, transfer, parsing, or similarity computation. The practical decision framework is likewise not specific to one product release. It generalises to any RDBMS with partial vector support, where engineers still need to decide whether scalar pre-filtering, compressed two-pass retrieval, or migration to a system with stronger native execution is the more appropriate response to the observed bottleneck. Several promising avenues for future research emerge from these findings. First, developing server-side User Defined Functions (UDFs) for common distance metrics (such as cosine similarity and Euclidean distance) could eliminate network bottlenecks by transferring computation to the database core. A UDF-based method could enable server-side WHERE clauses based on distance thresholds, removing the need to transfer entire datasets to the client.

Second, exploring the integration of external Approximate Nearest Neighbour (ANN) libraries (e.g., FAISS [39] and HNSWlib) via the MySQL plugin interface could provide a route to true server-side indexing in the open-source edition. Such integration would need to address the challenges of WAL and MVCC discussed in Section 2 to preserve ACID properties.

Third, more extensive concurrency experiments, including mixed read-write workloads and higher client counts, would provide a more realistic view of production performance. Fourth, larger-scale validation across multiple semantic embedding models and domain shifts would help determine how stable the PCA trade-off and learned projection matrix remain across retrieval domains. Finally, it remains an open question whether Oracle will backport vector indexing capabilities from MySQL HeatWave to the Community Edition. Competitive pressure from PostgreSQL's thriving pgvector ecosystem may accelerate this process, but the present evaluation would still remain informative as a template for diagnosing bottlenecks and selecting workarounds in partially featured vector-capable RDBMSs.

## REFERENCES


[1] A. Vaswani *et al.*, 'Attention is all you need', *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[2] P. Lewis *et al.*, 'Retrieval-augmented generation for knowledge-intensive nlp tasks', *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 9459–9474, 2020.

[3] W. Fan *et al.*, 'A survey on rag meeting llms: Towards retrieval-augmented large language models', in *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, 2024, pp. 6491–6501.

[4] Z. Jing, Y. Su, and Y. Han, 'When large language models meet vector databases: A survey', in *2025 Conference on Artificial Intelligence x Multimedia (AIxMM)*, IEEE, 2025, pp. 7–13.

[5] N. Reimers and I. Gurevych, 'Sentence-bert: Sentence embeddings using siamese bert-networks', *ArXiv Prepr. ArXiv190810084*, 2019.

[6] J. Wang *et al.*, 'Milvus: A purpose-built vector data management system', in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 2614–2627.

[7] Y. Han, C. Liu, and P. Wang, 'A comprehensive survey on vector database: Storage and retrieval technique, challenge', *ArXiv Prepr. ArXiv231011703*, 2023.

[8] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[9] A. Kane, 'pgvector: Open-source vector similarity search for PostgreSQL'. 2024. [Online]. Available: https://github.com/pgvector/pgvector

[10] PlanetScale Inc., 'Vector search and storage'. 2024. [Online]. Available: https://planetscale.com/docs/vitess/vectors

[11] C. Wei *et al.*, 'Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data', *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3152–3165, 2020.

[12] Google Cloud, 'Cloud SQL for MySQL: Vector Search'. 2024. [Online]. Available: https://cloud.google.com/sql/docs/mysql/vector-search

[13] Oracle Corporation, 'MySQL 9.0 Reference Manual: The VECTOR Type'. 2024. [Online]. Available: https://dev.mysql.com/doc/refman/9.0/en/vector.html

[14] Oracle, 'MySQL HeatWave Vector: Integrated Vector Store for Generative AI'. 2024. [Online]. Available: https://www.oracle.com/a/ocom/docs/mysql-heatwave-technical-brief.pdf

[15] A. Guttman, 'R-trees: A dynamic index structure for spatial searching', in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984, pp. 47–57.

[16] J. L. Bentley, 'Multidimensional binary search trees used for associative searching', *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[17] P. Indyk and R. Motwani, 'Approximate nearest neighbors: towards removing the curse of dimensionality', in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 604–613.

[18] R. Weber, H.-J. Schek, and S. Blott, 'A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces', in *VLDB*, 1998, pp. 194–205.

[19] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, 'When is "nearest neighbor" meaningful?', in *International conference on database theory*, Springer, 1999, pp. 217–235.

[20] A. Gionis, P. Indyk, R. Motwani, and others, 'Similarity search in high dimensions via hashing', in *Vldb*, 1999, pp. 518–529.

[21] H. Jegou, M. Douze, and C. Schmid, 'Product quantization for nearest neighbor search', *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, 2010.

[22] R. Guo *et al.*, 'Accelerating large-scale inference with anisotropic vector quantization', in *International Conference on Machine Learning*, PMLR, 2020, pp. 3887–3896.

[23] Y. A. Malkov and D. A. Yashunin, 'Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs', *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, 2018.

[24] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, 'Approximate nearest neighbor algorithm based on navigable small world graphs', *Inf. Syst.*, vol. 45, pp. 61–68, 2014.

[25] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi, 'Diskann: Fast accurate billion-point nearest neighbor search on a single node', *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[26] Q. Chen *et al.*, 'SPANN: Highly-efficient Billionscale Approximate Nearest Neighbor Search. CoRR abs/2111.08566 (2021)', *ArXiv Prepr. ArXiv211108566*, 2021.

[27] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, 'ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging', *ACM Trans. Database Syst. TODS*, vol. 17, no. 1, pp. 94–162, 1992.

[28] C. Zhan *et al.*, 'AnalyticDB: real-time OLAP database system at Alibaba cloud', *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 2059–2070, 2019.

[29] R. Ramakrishnan, J. Gehrke, and J. Gehrke, *Database management systems*, vol. 3. McGraw-Hill New York, 2003.

[30] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High performance MySQL: optimization, backups, and replication*. O'Reilly Media, Inc., 2012.

[31] E. J. O'neil, P. E. O'neil, and G. Weikum, 'The LRU-K page replacement algorithm for database disk buffering', *Acm Sigmod Rec.*, vol. 22, no. 2, pp. 297–306, 1993.

[32] M. Stonebraker, 'SQL databases v. NoSQL databases', *Commun. ACM*, vol. 53, no. 4, pp. 10–11, 2010.

[33] I. Jolliffe, 'Principal component analysis', in *International encyclopedia of statistical science*, Springer, 2011, pp. 1094–1096.

[34] S. Dasgupta and A. Gupta, 'An elementary proof of a theorem of Johnson and Lindenstrauss', *Random Struct. Algorithms*, vol. 22, no. 1, pp. 60–65, 2003.

[35] A. Kusupati *et al.*, 'Matryoshka representation learning', *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 30233–30249, 2022.

[36] M. Aumüller, E. Bernhardsson, and A. Faithfull, 'ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms', *Inf. Syst.*, vol. 87, p. 101374, 2020.

[37] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, 'An updated performance comparison of virtual machines and linux containers', in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, IEEE, 2015, pp. 171–172.

[38] J. Johnson, M. Douze, and H. Jégou, 'Billion-scale similarity search with GPUs', *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[39] M. Douze *et al.*, 'The faiss library', *IEEE Trans. Big Data*, 2025.

[40] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding', in *Proceedings of the 2019 conference of the*

*North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[41] A. Neelakantan *et al.*, 'Text and code embeddings by contrastive pre-training', *ArXiv Prepr. ArXiv220110005*, 2022.

[42] R. Bayer and E. McCreight, 'Organization and maintenance of large ordered indices', in *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, 1970, pp. 107–141.