ARCADE: A Real-Time Data System for Hybrid and Continuous Query Processing across Diverse Data Modalities

Jingyi Yang, Songsong Mo, Jiachen Shi, Zihao Yu, Kunhao Shi, Xuchen Ding, Gao Cong
{jyang028@e.,songsong.mo@,jiachen001@e.,zihao010@e.,kunhao002@e.,xuchen002@e.,gaocong@}ntu.edu.sg
Nanyang Technological University

ABSTRACT

The explosive growth of multimodal data—spanning text, spatial, vector, and relational modalities—coupled with the need for real-time semantic search and retrieval, has outpaced the capabilities of existing systems, which either lack efficient ingestion and continuous query capabilities, or fall short in supporting expressive hybrid analytics. We introduce ARCADE, a real-time data system that efficiently supports high-throughput ingestion and expressive hybrid and continuous query processing across diverse data types. ARCADE introduces a unified disk-based secondary index framework on LSM-based storage for vector, spatial, and text data modalities, a comprehensive cost-based query optimizer that jointly leverages multiple heterogeneous indexes for hybrid queries, and an incremental materialized view framework for efficient continuous queries. Our demonstration showcases ARCADE through interactive real-world scenarios of social media marketing and equity research, exposing system-level metrics, internal query optimization decisions, and side-by-side baseline comparisons to provide attendees with deep insights into how ARCADE's architectural choices facilitate real-time multimodal analytics.

PVLDB Reference Format:
Jingyi Yang, Songsong Mo, Jiachen Shi, Zihao Yu, Kunhao Shi, Xuchen Ding, Gao Cong. ARCADE: A Real-Time Data System for Hybrid and Continuous Query Processing across Diverse Data Modalities. PVLDB, 19(12): 4798 - 4801, 2026.
doi:10.14778/3827998.3828125

1 INTRODUCTION

Data of diverse modalities, *e.g.*, text, spatial, vector, and relational data, are continuously being generated at an unprecedented rate from social media platforms, urban mobility systems, and the financial sector. The increasing demand for semantic search and retrieval over such multimodal data, enabled by large language models (LLMs), imposes substantial challenges on modern database systems concerning real-time data ingestion and efficient execution of complex queries.

A crucial aspect of real-time multimodal data analytics is the efficient support of *hybrid queries*—joint ranking and filtering across vector, spatial, textual, and relational attributes [5, 13]—and *continuous queries* that automatically execute at fixed intervals or upon data updates for real-time monitoring. Since re-executed queries are frequently hybrid in nature, it is essential for modern systems to efficiently support both simultaneously.

Existing systems broadly fall into two categories. *Multimodal data systems*, such as PostgreSQL [2], DuckDB [11], and Snowflake [7], offer robust support for hybrid queries but typically lack high throughput ingestion and continuous query capabilities. *Real-time data systems* employing LSM-tree [10] architectures, such as AsterixDB [3] and Milvus [13], prioritize efficient ingestion but offer limited multimodal support. Even recent systems like SingleStore-V [5] only support hash indexes on non-vector attributes and lack the ability to jointly rank results based on both vector and non-vector attribute similarities. To address this gap, we introduce ARCADE [14], a real-time data system built on open-source RocksDB (storage) and MySQL (query engine) that addresses three core challenges:

- **Unified disk-based secondary index framework.** Unlike systems that maintain secondary indexes in separate LSM-trees [3, 4] or require fully loading per-segment indexes into memory [5], ARCADE embeds secondary index blocks for vector, spatial, and text data directly within SST files. This design enables fine-grained block-level access and cache reuse for fast queries, while its per-segment construction mechanism achieves 19× lower ingestion overhead compared to in-memory FAISS indexes.
- **Comprehensive cost-based query optimizer.** A unified `Next(.)` iterator interface across all index types enables composition of multiple heterogeneous indexes in a single hybrid query plan. For hybrid NN queries, an NRA-style [9] algorithm incrementally traverses indexes with early termination, achieving up to 7.4× speedup over the best baselines.
- **Incremental materialized view framework.** A view-based approach for continuous query optimization selects and maintains shared views based on benefit–cost analysis, supporting both SYNC (periodic) and ASYNC (event-driven) execution modes.

The full system design, algorithms, and experiments are presented in our research paper at ICDE 2026 [14][1].

Demonstration. Our demonstration provides interactive exploration of ARCADE's internals through real-world scenarios based on dynamic Twitter data. Attendees can: (i) explore hybrid query optimization by creating/combining secondary indexes and observing how the optimizer selects plans, with internal metrics (*e.g.*, SST files accessed, cache hit rates); (ii) interact with continuous query management by registering SYNC/ASYNC queries and inspecting materialized view sharing; (iii) experience real-time RAG where ARCADE's vector index retrieves content for LLM answer generation; and (iv) compare with PostgreSQL (pgvector + PostGIS) side-by-side. Participants can also customize workload characteristics to explore their own scenarios.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828125

[1] ARCADE is open source at https://github.com/Jamesyang2333/ARCADE

2 SYSTEM OVERVIEW

System architecture. ARCADE adopts a three-layer architecture (Figure 1). The query interface accepts SQL queries and dispatches them via dedicated read/write threads. The query processing layer separates read and write paths and constructs optimized execution plans that jointly leverage secondary indexes and incremental materialized views. The storage layer is built on a partitioned LSM-tree with in-memory components (write buffer, block cache, global secondary indexes) and on-disk SST files that co-locate data blocks with unified secondary index blocks. For comprehensive details, we refer the reader to [14].

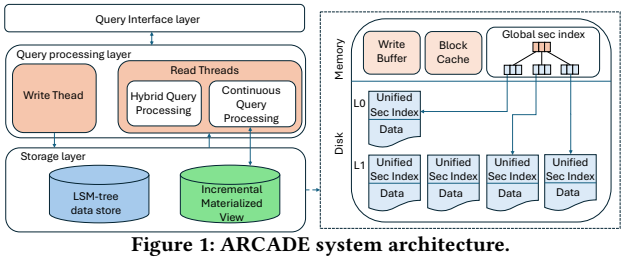


Figure 1: ARCADE system architecture.

2.1 Storage Layer

Unified disk-based secondary index. ARCADE implements an efficient and unified disk-based secondary index framework that extends the NEXT framework [12] to support multimodal data modalities including vector, spatial, and text data. Unlike existing LSM-based systems that maintain secondary-to-primary key mappings in a separate LSM-tree [3, 4], ARCADE integrates secondary index structures within the primary LSM table, eliminating index navigation overhead and cross-segment index maintenance overhead during compaction.

ARCADE adopts a two-level secondary index framework consisting of *per-segment* components residing in SST files and a *global index* component in RAM. Each per-segment component stores sorted mappings from secondary attribute values to data block handles, created during SST construction and remaining immutable. The global index, organized as a multi-level tree, maps secondary value ranges to SST index blocks, enabling efficient SST file pruning and direct query routing during lookups. When SST files undergo flush or compaction, the global index is updated at file granularity using batch operations in background threads, introducing minimal overhead on the ingestion path [14].

Vector and text IVF index. To support multimodal data, we extend the unified index framework to vector IVF index and text IVF index. The key distinction from existing per-segment vector indexes (*e.g.*, SingleStore-V [5]) is in access granularity: while existing approaches require each segment's entire vector index to be loaded into memory, ARCADE organizes the vector index into logical index blocks, enabling fine-grained reads and cached reads across queries. As illustrated in Figure 2, the Vector IVF index adopts a two-level approach: the first level consists of index metadata blocks storing IVF centroid-to-posting-list mappings; the second level consists of posting list blocks storing (vector, block handle) pairs. Both types of blocks are implemented as logical SST blocks—the basic I/O unit in RocksDB with block cache support [8].

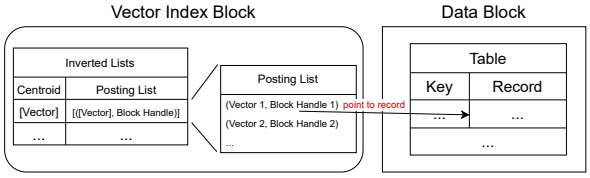


Figure 2: Vector IVF index structure.

2.2 Query Processing

Data types and queries. ARCADE supports relational, spatial (`GEOMETRY`), text (`TEXT` with inverted index), and vector (`JSON` with IVF index) data types. It handles both *snapshot queries* (one-time) and *continuous queries*, which include SYNC queries (periodic re-execution at fixed intervals) and ASYNC queries (triggered by relevant data changes).

Cost-based hybrid query optimization. Existing systems typically select a single index access path, failing to jointly leverage multiple indexes cooperatively [14]. ARCADE addresses this through a cost-based optimizer built upon MySQL's relational optimizer and extended with a unified `Next(.)` iterator interface across all secondary index types—vector IVF, spatial R-tree, and text inverted indexes. This enables the optimizer to: (i) enumerate feasible multi-index access plans, (ii) evaluate them using a cost model accounting for index access cost within the LSM hierarchy, candidate set sizes, and residual predicate overhead, and (iii) select the optimal index combination.

Hybrid NN query processing. For hybrid NN queries that rank results by combining vector similarity with non-vector attribute similarity (*e.g.*, spatial distance), existing systems typically resort to a costly two-stage heuristic: retrieving a large candidate set from one modality and re-ranking via random I/O lookups (Figure 3, left). ARCADE instead adopts an NRA [9]-style algorithm (Figure 3, right) that jointly traverses multiple indexes using the unified `Next(.)` interface, incrementally updating upper and lower bounds on candidate scores and terminating early once the top-k results are guaranteed, reducing index I/O by 93.8–98.9% [14].

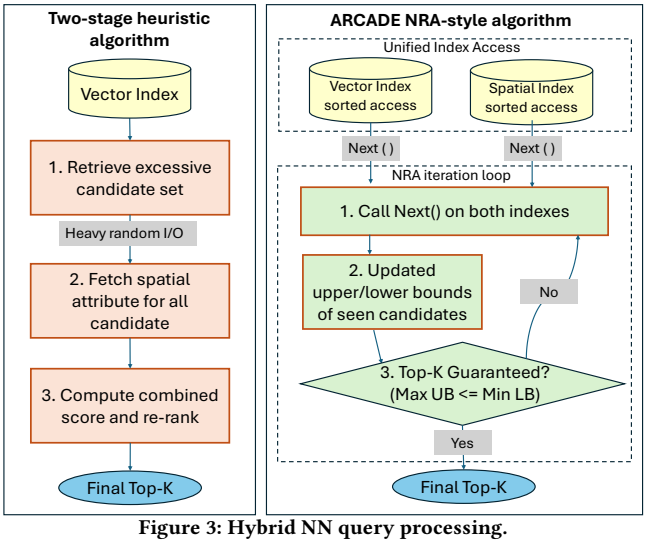


Figure 3: Hybrid NN query processing.

Continuous query optimization. Continuous query optimization traditionally relies on caching full query results [6], which

faces scalability and reusability challenges. ARCADE proposes an incremental materialized view-based framework (Figure 4) that generates shared views from issued continuous queries. The framework categorizes views into spatial range views (clustering overlapping query regions) and vector NN views (materializing expanded candidate sets), selecting views with the highest benefit-to-cost ratio under a storage budget. Views are incrementally maintained via delta-based updates [14].

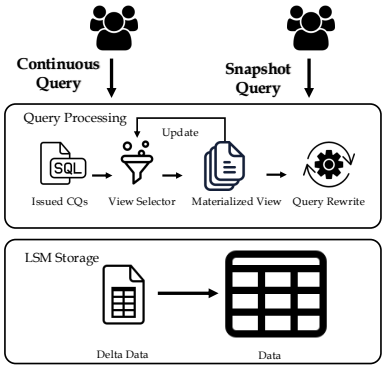


Figure 4: Continuous query processing framework.

Consistency semantics. ARCADE provides snapshot isolation for reads and follows RocksDB's WAL protocol for durability [14].

3 DEMONSTRATION OVERVIEW

Overview. The demo interface consists of five main panels: Query Panel, Parameter Setting Panel, Index Management Panel, Continuous Query Management Panel, and System Metrics Panel.

Query Panel. This panel (Figure 5) allows users to execute SQL queries with customized index and view strategies. It displays the optimized query plan, execution time, internal metrics (SST files accessed, index blocks read, cache hit rate), and result table. It also supports side-by-side comparison against multiple baselines, and alternative execution strategies—so attendees can isolate the speedups attributed to ARCADE's unified optimizer.

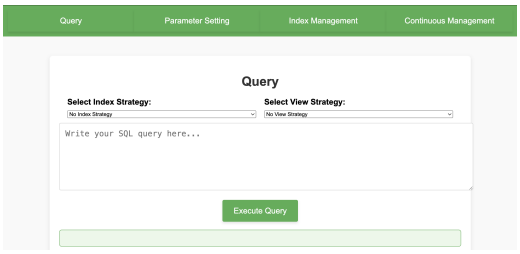


Figure 5: Query execution panel with SQL input, query plan, metrics, and results.

Parameter Setting Panel. Users configure system parameters such as n_{probe} for ANN processing and the materialized view budget.

Index Management Panel. This panel (Figure 6) lets users create vector IVF, spatial, and text indexes via SQL commands, and compose them into reusable *index strategies* for query execution.

Continuous Query Management Panel. This panel (Figure 7) lets users define SYNC or ASYNC queries by specifying SQL, query type, and interval. It lists active queries with their results, statistics, and associated materialized views.

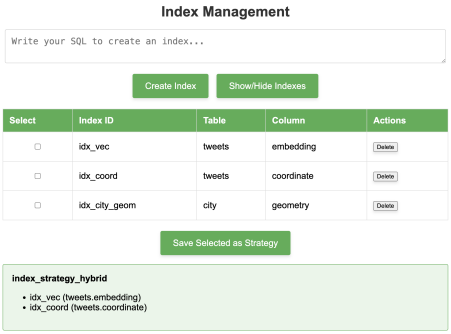


Figure 6: Index management panel. Users create, delete, and combine indexes into reusable strategies.

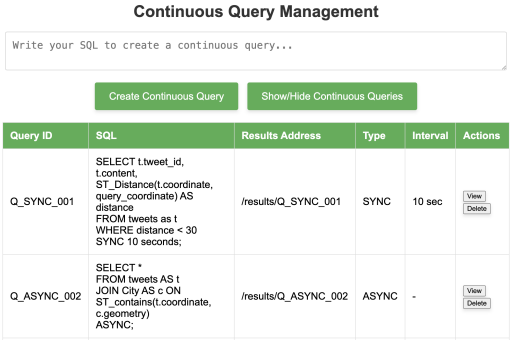


Figure 7: Continuous query management panel.

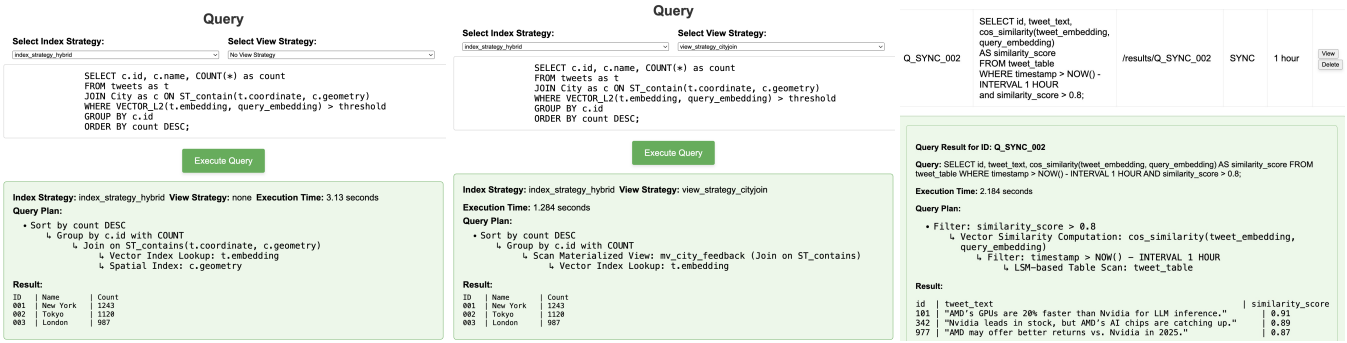
System Metrics Panel. This panel provides real-time visibility into ARCADE's internals. It displays: (i) LSM-tree statistics including levels, SST file counts, and compaction events; (ii) per-query index metrics such as SST files pruned, index blocks read, and centroids probed; (iii) block cache statistics including hit rates and eviction counts; and (iv) ingestion throughput. These metrics update in real time as queries execute, allowing attendees to observe how index creation, view materialization, and parameter changes affect internal system behavior.

4 DEMONSTRATION SCENARIOS

Our demonstration involves three scenarios designed to exercise complementary capabilities and query types: Scenario 1 stresses *hybrid-search* multi-index optimization, Scenario 2 stresses *continuous* queries with incremental view maintenance, and Scenario 3 stresses *hybrid-NN* retrieval driving real-time RAG. Throughout all scenarios, users can inspect system-level metrics and compare performance through a side-by-side panel that runs the same queries on multiple baselines. We include PostgreSQL as a representative feature-complete multimodal database and AsterixDB as a representative LSM-based real-time system.

Dataset. The demo uses our TRACY (**T**weet hyb**R**id **A**nd **C**ontinuous quer**Y**) benchmark [14], with three tables: Tweet (33M geo-tagged tweets with 128-dim embeddings), POI (7M Yelp points-of-interest with 128-dim embeddings), and City (186K cities from OSM [1]).

Query types. The demo uses two hybrid query types: *(1) Hybrid Search Queries* enable selection with multiple predicates across relational and non-relational (*e.g.*, vector, spatial) attributes. *(2) Hybrid NN Queries* rank results based on a combination of similarity measures over multiple modalities, potentially with additional filters.



(a) With unified secondary indexes (b) With incremental materialized view (c) Continuous query for RAG

Figure 8: Query execution and continuous query in ARCADE. (a) Multi-index plan to prune SST files. (b) Query rewriting with cached materialized views to eliminate costly joins. (c) Continuous query for RAG that retrieves semantically relevant tweets at regular intervals.

Scenario 1: Hybrid query optimization for social media marketing. A marketing manager examines the popularity of "Vision X" across cities by finding all semantically relevant tweets and aggregating results by city—a hybrid search query involving a spatial join and vector similarity filter (Figure 8).

The attendee first executes the query without any secondary indexes, observing the baseline full-scan plan. The System Metrics Panel reveals the number of SST files accessed and the high I/O cost. The attendee then creates three indexes using the Index Management Panel: a vector IVF index on the tweet embedding, and spatial indexes on the tweet coordinates and city geometry. After saving these as an *index strategy*, re-executing the query reveals a different query plan (Figure 8a), where the optimizer jointly leverages vector and spatial indexes to prune SST files and reduce I/O. The System Metrics Panel displays the improvement in SST files pruned, index blocks read, and cache hit rates, allowing direct comparison of query plans with and without multi-index optimization.

Scenario 2: Continuous query for equity research. An investment analyst continuously monitors nearby tweets to detect emerging market sentiment about a semiconductor maker. The user registers a SYNC continuous query retrieving semantically relevant tweets within a spatial range every 10 seconds:

```
SELECT t.tweet_id, t.content,
 ST_Distance(t.coordinates, @location) AS dist,
 VECTOR_L2(t.embedding, @query_emb) AS sim
FROM tweets t
WHERE dist < @max_distance
ORDER BY sim DESC SYNC 10 seconds;
```

Alternatively, the user can switch to ASYNC mode, which triggers re-execution upon relevant data changes rather than at fixed intervals. Once registered, continuous queries are maintained using incremental materialized views. As shown in Figure 9, ARCADE groups overlapping spatial queries and materializes shared regions (*e.g.*, the `v_dist` view) to avoid redundant computation across queries.

Users can save cached views as a *view strategy* to accelerate snapshot queries. As shown in Figure 8b, the optimizer rewrites the query to leverage the cached spatial join view, eliminating the costly join operation. The System Metrics Panel shows incremental update overhead and view reuse statistics, enabling attendees to compare execution with and without materialized view optimization.

Scenario 3: Real-time RAG with LSM vector index. ARCADE supports Retrieval-Augmented Generation (RAG) on real-time data. The investment analyst queries: "Summarize the latest one-hour

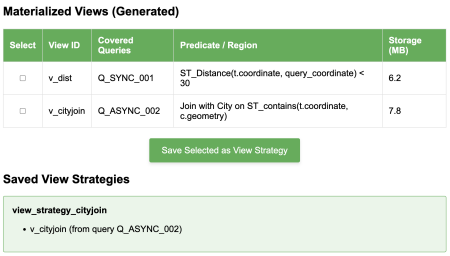


Figure 9: Materialized view management. ARCADE groups overlapping spatial queries and materializes shared regions for reuse.

posts comparing products and investment returns of Nvidia and AMD." Tweet contents are retrieved from ARCADE in real time and fed to an LLM as context for answer generation. A continuous query performs hourly ANN search, retrieving semantically relevant tweets (similarity score > 0.8), as shown in Figure 8c. Users can adjust n_{probe} to tune the recall–latency trade-off and observe how different settings affect both retrieval quality and query performance in the System Metrics Panel.

REFERENCES

[1] [n.d.]. OpenStreetMap. https://www.openstreetmap.org.
[2] [n.d.]. pgvector. https://github.com/pgvector/pgvector.
[3] Sattam Alsubaiee et al. 2014. AsterixDB: A Scalable, Open Source BDMS. *Proc. VLDB Endow.* 7, 14 (2014), 1905–1916.
[4] Fay Chang et al. 2008. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* 26, 2 (2008), 1–26.
[5] Cheng Chen, Chenzhe Jin, Yunan Zhang, et al. 2024. Singlestore-v: An integrated vector database system in singlestore. *PVLDB* 17, 12 (2024), 3772–3785.
[6] Zhida Chen et al. 2023. STAR: A Cache-based Stream Warehouse System for Spatial Data. *ACM Trans. Spatial Algorithms Syst.* 9, 4 (2023), 28:1–28:27.
[7] Benoit Dageville et al. 2016. The snowflake elastic data warehouse. In *SIGMOD*. 215–226.
[8] Facebook. 2023. RocksDB. https://github.com/facebook/rocksdb.
[9] Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences* 66, 4 (2003), 614–656.
[10] Chen Luo and Michael J. Carey. 2020. LSM-based storage techniques: a survey. *VLDB J.* 29, 1 (2020), 393–418.
[11] Mark Raasveldt and Hannes Mühleisen. 2019. Duckdb: an embeddable analytical database. In *SIGMOD*. 1981–1984.
[12] Jiachen Shi, Jingyi Yang, Gao Cong, and Xiao-Li Li. 2025. NEXT: A New Secondary Index Framework for LSM-based Data Storage. *SIGMOD* (2025). https://github.com/JC-Shi/NEXT.
[13] Jianguo Wang et al. 2021. Milvus: A purpose-built vector data management system. In *SIGMOD*. 2614–2627.
[14] Jingyi Yang, Songsong Mo, Jiachen Shi, Zihao Yu, Kunhao Shi, Xuchen Ding, and Gao Cong. 2026. ARCADE: A Real-Time Data System for Hybrid and Continuous Query Processing across Diverse Data Modalities. In *Proceedings of the 42nd IEEE International Conference on Data Engineering (ICDE)*. https://github.com/Jamesyang2333/ARCADE/blob/main/ARCADE_ICDE_camera_ready.pdf.