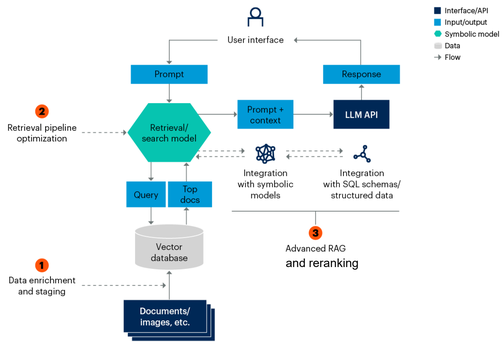
InterSystems IRIS: Achieving Speed and Usability in an Integrated Vector Database

David Van De Griek, Boya Song, Philip Miloslavsky, Yuchen Liu, Yiwen Huang, Mark Hanson, Jeff Fried, Dimitriy Bochkov – InterSystems

Abstract

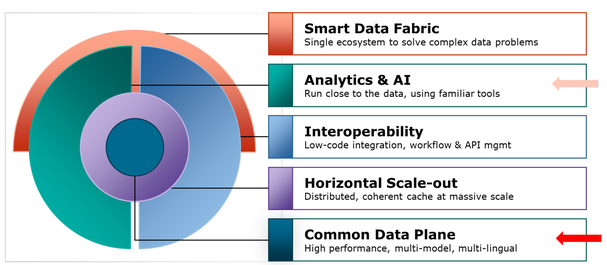
This paper introduces an advanced implementation of an integrated vector database grounded in a "Post-Relational" architectural approach. The resulting system is inherently multi-model and achieves superior scalability and performance metrics when compared to existing frameworks such as pgvector and Faiss. By integrating diverse functionalities into a single cohesive platform, it effectively mitigates data silos and lowers operational expenditures, thereby streamlining the overall data management lifecycle. Additionally, the system is engineered to facilitate ease of use through an intuitive SQL-based interface, coupled with comprehensive integration into the data engine. This design choice effectively abstracts the underlying pipeline complexities from the user, ensuring optimal efficiency in both the creation of embeddings and the retrieval process, thus broadening accessibility to sophisticated vector data operations without detracting from performance or scalability.

Vector Search in RAG Applications



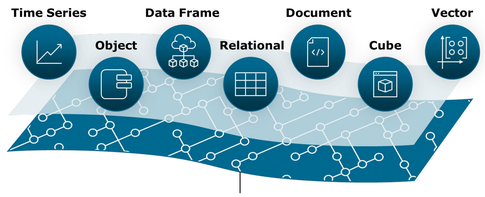
Criteria	Specialized Vector Databases	Integrated Vector Databases
Examples	Pinecone, Milvus, Weviate	PPASE, PostgreSQL+pgvector, ElasticSearch **InterSystems IRIS**
Ease of Use	Designed specifically for vector data	Integrated into existing databases, leveraging familiar interfaces and tools
Footprint	Typically requires additional infrastructure and resources	Utilizes existing database infrastructure, reducing the need for additional resources
Performance	Optimized for high-dimensional vector searches, often with advanced indexing techniques	Performance can vary based on the underlying database but benefits from integrated indexing

InterSystems IRIS Architectural Layers



Extending a native Multi-Model DB

The Common Data Plane



^global(<key1>,<key2>,...) = $encoding(<val1>,<val2>,...)

Design Goal: **Unified, versatile data engine that supports vector fields and vector indices** (e.g. HNSW index)

Unified storage for data and index vector data, columnar data and regular data
- Minimize data duplication
- Index/field size not limited by memory size

Unified SQL engine with great query processing capabilities, including
- Transactions
- Full SQL Support
- Filtered Vector Search
- Vector Range Search
- Vector Range Join
- Vector with Fulltext Search
- Semantic Join

Storage Design

Vector fields stored as a collection of chunked vectors, 64K values per chunk.
Metadata (Columnar Index Map and Columnar Data Map) support fast sorting and SIMD operations.

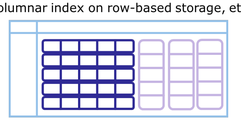
Special %Vector type: %Embedding

Flexible design, including ability to accommodate VERY long vectors
- Store long vectors in their own globals for performance and footprint
- Provision for managing space/precision tradeoffs
  - $vector can be integer, decimal FP; supports sparse encodings and different storage size magnitude for both sparse and dense encodings
- new DataDefinitionLocation property

Vector Storage Model evolved from columnar use cases

Storage model for SQL based on native $vector data type to deliver key analytical querying facilities needed for next-generation Data Warehouses, Lakes and Lakehouses

- Aligns physical table layout with typical analytical access patterns
- $vector language feature designed to support **translytical workloads**
- **Order of magnitude speedup** for analytical queries thanks to SIMD use and vectorized execution
- **Schema flexibility** - mixing row & column storage - is a key InterSystems IRIS differentiator
- **Indexing flexibility** – can use a columnar index on row-based storage, etc.



Processing Flexibility & Accuracy

Fast vector operations

- **Numeric Operations (scalar and vector-wise)**
  - $VECTOROP("+" | "-" | "/" | ... | "cosine" | "dot-product" , vector, vector | scalar, bitmap) returns vector
- **String Operations (scalar and vector-wise)**
  - $VECTOROP("_" | "lower" | "substring" | ... , vector, vector | scalar, bitmap) returns vector
- **Filter Operations (scalar and vector-wise)**
  - $VECTOROP("=" | ">" | "<" | ... | "defined" | "undefined" , vector, vector | scalar, bitmap) returns bitmap
- **Aggregate Operations**
  - $VECTOROP("count" | "max" | "min" | "sum", vector, bitmap) returns scalar
- **Grouping Operations**
  - $VECTOROP("group", "count" | "max" | "min" | "sum", vector, bitmap, list) modifies list
- **Miscellaneous Operations**
  - $VECTOROP("convert", vector)
  - $VECTOROP("mask", vector, scalar)
  - $VECTOROP("positions", vector, bitmap)
  - $VECTOROP("bytesize", vector)
  - ...

Query processing implementation: Unified SQL Engine

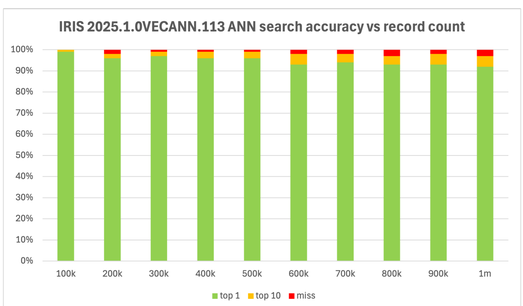
- Unified Storage Model
  - comparable access cost
    - for any storage type
    - for any index type
- Universal Query Optimizer
  - pre-optimizer query rewrite
  - awareness of vector algorithms
  - multi-index plans
- Adaptive Parallel Execution
  - data agnostic

- Efficient integration of vector/embedding data type
- Example: Semantic Join
  - join with a similarity condition on word embeddings [1]
  - tolerates misspellings and different formats to deliver more join results

[1] Dong Y, Xiao C, Nozawa T, Enomoto M, and Oyamada M DeepJoin: joinable table discovery with pre-trained language models Proc. VLDB Endow. Digital Library

Semantic Join in SQL

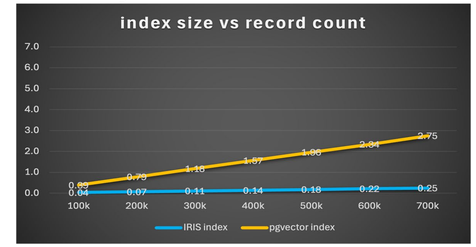
```
select
    FilmA.title Film1,
    FilmB.title Film2,
    ReviewA.star_rating * ReviewB.star_rating CombinedRating
from  Cinema.Film FilmA
join  Cinema.Film FilmB
    on (vector_cosine(FilmA.overview_embedding, FilmB.overview_embedding)>.55)
join  Cinema.Review ReviewA on (FilmA.imdb_id=ReviewA.imdb_id)
join  Cinema.Review ReviewB on (FilmB.imdb_id=ReviewB.imdb_id)
where FilmA.imdb_id<FilmB.imdb_id
    and FilmA.release_year>1950
    and FilmB.length>60
order by CombinedRating desc, FilmA.imdb_id, FilmB.imdb_id
```



Performance and Footprint

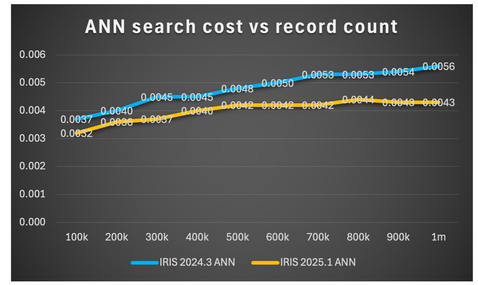
Query processing implementation: HNSW index

- Implemented HNSW index according to Malkov 2016
- No need to store the vectors in the index as the SQL engine can access the original vector field



- Challenge: needs to be compatible with parallel INSERT

We continue to improve performance release on release

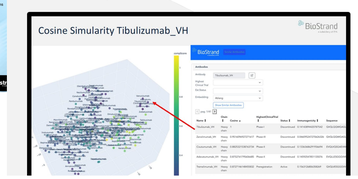


Commercial Deployment

Integrated Vector Search has been adopted and deployed by many customers since the initial release in March 2024. Examples:

SerenityGPT – built on InterSystems IRIS & Vector Search

Biostrand: Complex Data Analysis With IRIS Vector Search



References

[1] Yunan Zhang, Shige Liu, Jianguo Wang.
Are There Fundamental Limitations in Supporting Vector Data Management in Relational Databases? A Case Study of PostgreSQL.
Proceedings of International Conference on Data Engineering (ICDE), 2024.

[2] Benjamin De Boe, Tom Woodfin, Thomas Dyar, Dave McCaldon, Aleks Djakovic, Alex MacLeod and Don Woodlock, 2020.
IntegratedML: Every SQL Developer is a Data Scientist.
Proceedings of 4th Workshop on Data Management for End-to-End Machine Learning (DEEM) 2020.

[3] Jianguo Wang, Eric Hanson, Guoliang Li, Yannis Papakonstantinou, Harsha Simhadri, Charles Xie.
Vector Databases: What's Really New and What's Next?.
Proceedings of Very Large Data Bases Conference (VLDB), 2024.

Try genAI development with InterSystems IRIS –get hands on for free at:

InterSystems.com/TryIRIS

Contact:
Jeff.Fried@InterSystems.com
@jefffried