Redis - From Documents to Vectors and Natural Language APIs

Suyog Kale
Solution Architect Manager, Redis India | Organizer, Pune Developer's Community (PDC)

Ravi Joshi
Technologist Architect

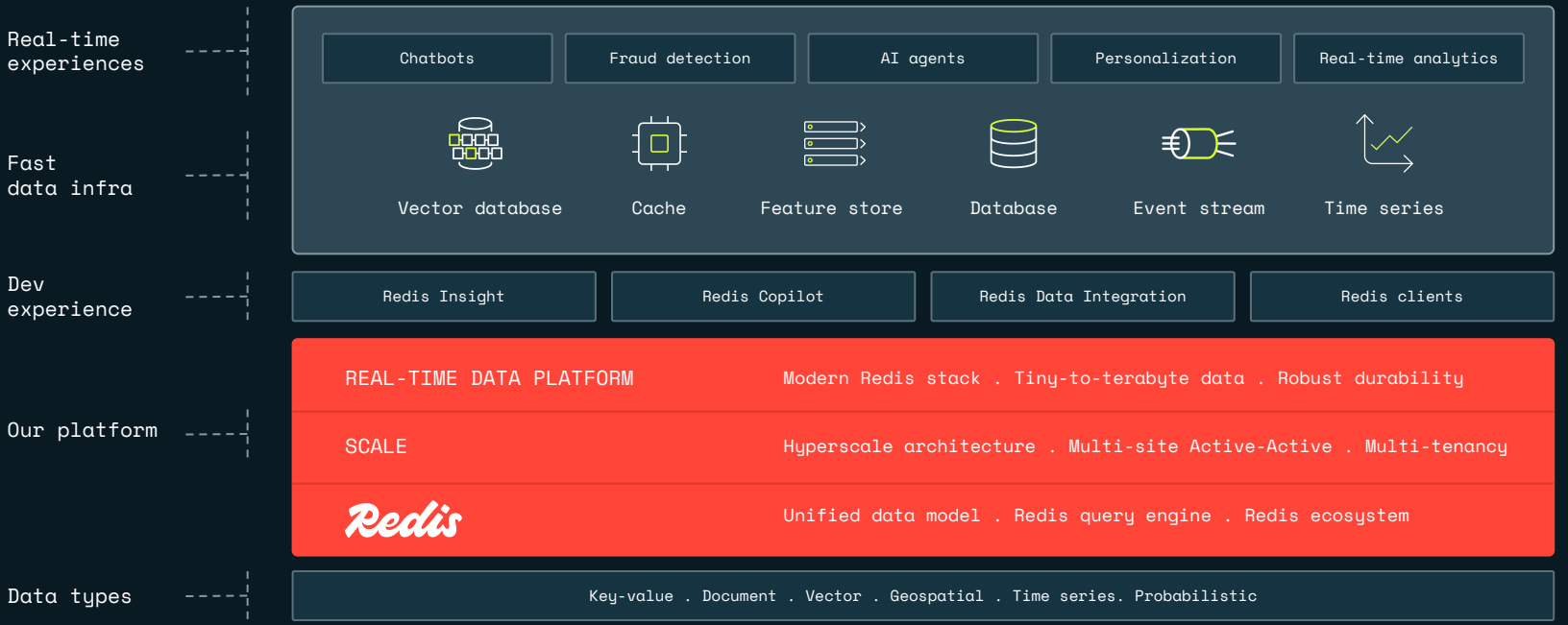
Agenda

- Introduction
- Redis
- SQL to NoSQL
- Redis JSON and Search
- Redis Vector database
- Demos
- Q&A
-

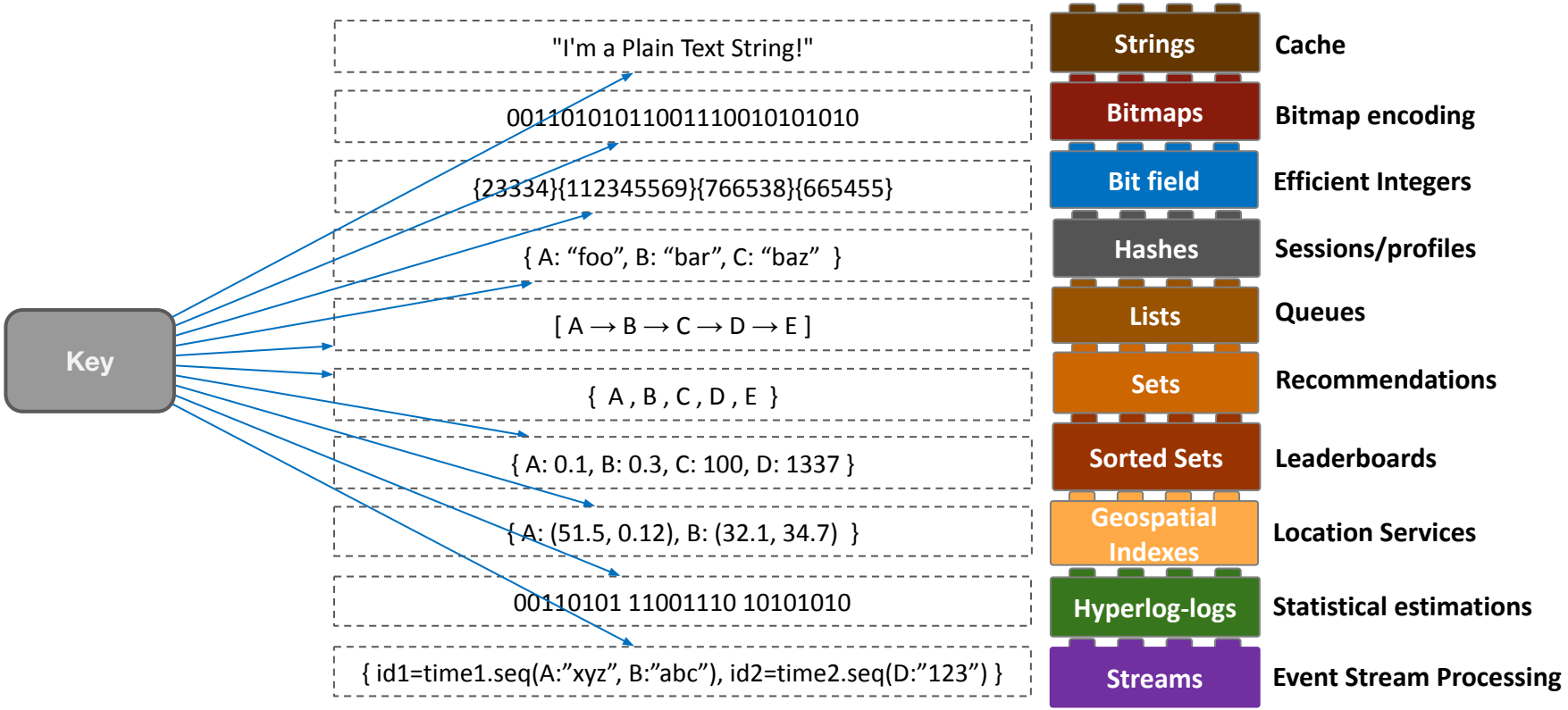
Introduction

Before, you knew us for caching.

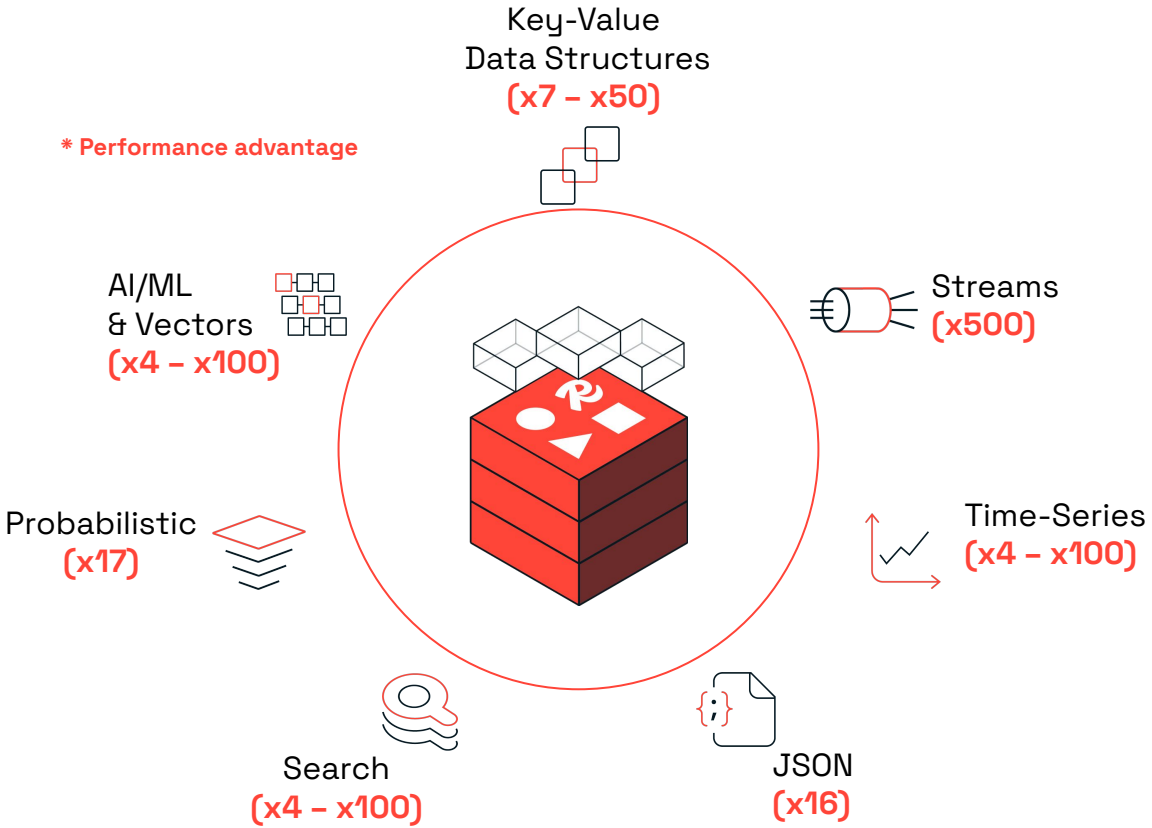
But we do a lot more.



Redis Data Structures - Use Case Driven Design



Multiple Data Models, Unprecedented Performance

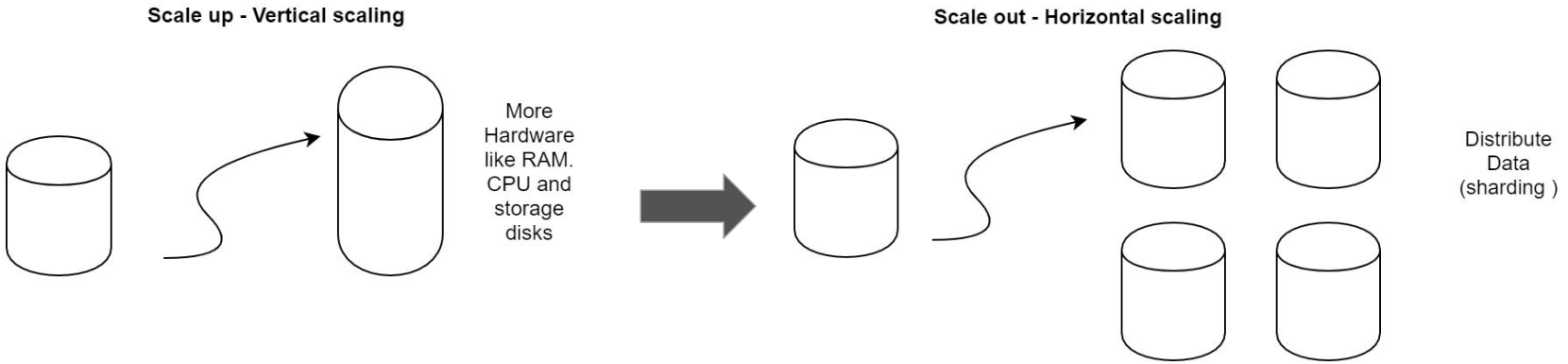


- Dedicated engine for each data model
- Models engines can be selectively loaded, according to use case
- All model engines access the same data, eliminating the need for transferring data between them

Relational vs NoSQL

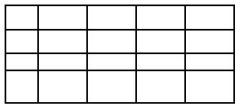
	Relational Database	**NoSQL Database**
Type	Relational data models	Nonrelational data models
Schema	Pre-defined schema	Schema is flexible
Data structure	Data is stored in tables, the schema is identified by column names.	No fixed structure, data can be stored as key-value, graph or documents.
Scalability	Limited scope to scale. Most of the time adding more infra is the only possible option. (vertical scaling)	Supports vertical as well as horizontal scaling
Property followed	ACID properties (Atomicity, Consistency, Isolation, and Durability)	CAP theorem (Eventual Consistency, Availability and Partition tolerance).
Support	Great support by the community as well as enterprises	Only few NoSQL databases have good support by community and enterprises
Example	PostgreSQL, MySQL, Oracle and Microsoft SQL Server	**Redis**, Cassandra, MongoDB, BigTable, HBase, Neo4j and CouchDB

Relational vs NoSQL

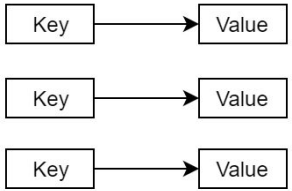


Relational vs NoSQL

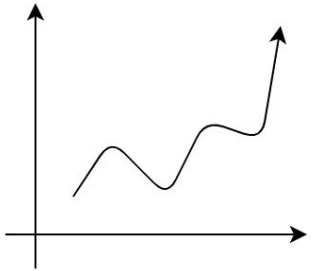
Relational Data



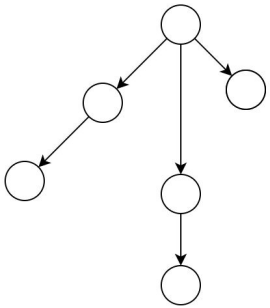
Key value pairs



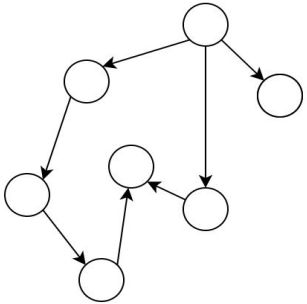
Time series



Documents



Graph



Why You Need SQL to RedisJSON and RediSearch

- Traditional RDBMS struggles with performance at scale.
- JSON document modeling matches modern app needs.
- RedisJSON stores rich objects; RediSearch provides fast querying.
- Enables schema-less, nested object storage with full-text search and filtering.

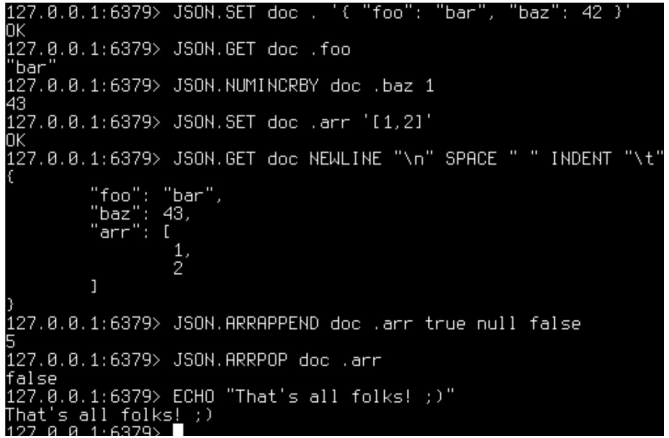
Feature	SQL (Relational)	RedisJSON + RediSearch (NoSQL)
Schema	Fixed	Dynamic / Flexible
Relationships	JOINs	Embedded documents
Performance	Disk-based, slower	In-memory, fast
Horizontal scaling	Complex	Built-in (Redis Enterprise)
Search capabilities	Limited full-text search	Advanced with RediSearch

RedisJSON

- It's a Redis Module that implements JSON as a native data structure
- JSON Path syntax for selecting fields within documents
- Documents are stored as binary data in a tree structure allowing fast access to sub-elements
- Typed atomic operations for all JSON value types.

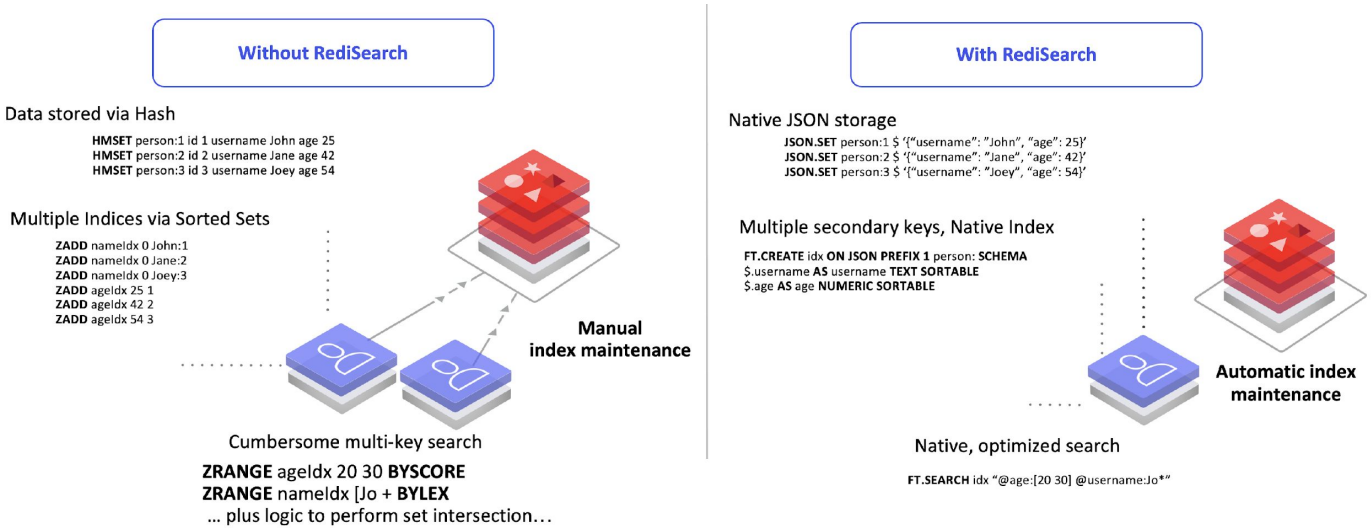
Advantages of RedisJSON:

- Nesting
- In-place updates
- Atomic Read operations
- Indexing, Querying and Full-text Search with RediSearch
- 12.7x faster than MongoDB



RediSearch Fast creation and automatic indexing of secondary keys

- Index any field in the database in real time for faster search results and unique data views
- Multi-field queries with no application code changes
- Once defined indexes are automatically updated, never manage indexes again



Redis Enterprise real time search capabilities

Indexing

Secondary index structures for numeric data, text values (tags), and geo locations

Indexing on multiple fields in docs via a single index

Declarative indexes

Incremental synchronous indexing

Document deletion and updating with index

Garbage collection

Querying

Multi fields queries

Numeric filters and range queries

Geo radius queries

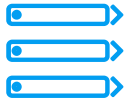
Complex Boolean queries with AND, OR, NOT operators between sub-queries

Optional query clauses

Ask for full document content or just ids

Aggregations across shards

Full-text search



Inverted index structure for full-text

Document ranking & field weights

Expansion and scoring

Prefix/Infix/Suffix based searches

Exact phrase search, or slop based search

Stemming based query expansion in many languages

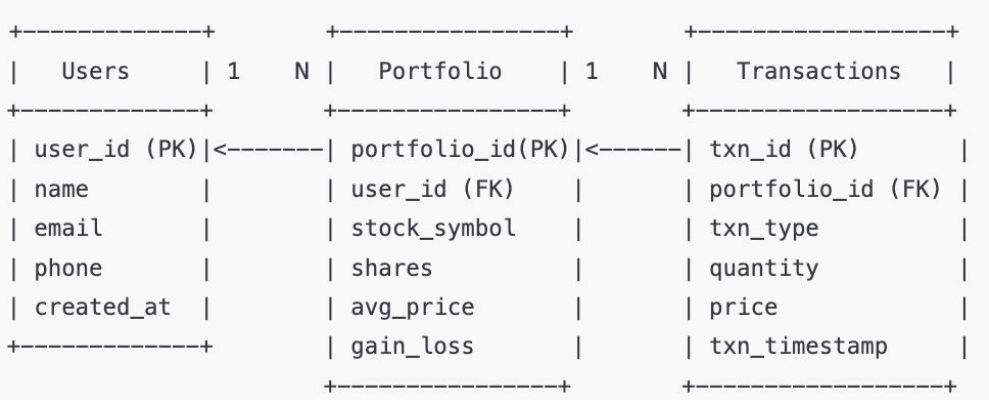
Support for custom functions for query

Limiting searches to specific doc fields

Spell-checking and auto-completion dictionaries

Demo time

SQL Entity Design - User, Portfolio, Transactions




```
SELECT
   U.user_id, u.name, u.email, u.phone, u.created_at,
   P.portfolio_id, p.stock_symbol, p.shares, p.avg_price, p.gain_loss

FROM Users u
JOIN Portfolio p ON u.user_id = p.user_id
WHERE u.user_id = 'u1';
```

Challenges in SQL Model at Scale

- Complex JOINs across tables.
- Slower queries with increasing users and transactions.
- Multiple queries needed to retrieve full user context.
- Difficult to scale horizontally.

RedisJSON Document Model

```
{
  "userId": "u1",
  "name": "User 1",
  "portfolio": [
    {"stock": "AAPL", "quantity": 10, "gainLoss": 15.5},
    {"stock": "TSLA", "quantity": 5, "gainLoss": -3.2}
  ],
  "transactions": [
    {"txnId": "t1", "type": "BUY", "amount": 2000, "timestamp": "2024-01-01T10:00:00Z"},
    {"txnId": "t2", "type": "SELL", "amount": 3000, "timestamp": "2024-01-03T10:00:00Z"}
  ]
}
```

- Index Definition:
  - @userId, @portfolio[*].stock, @portfolio[*].gainLoss, @transactions[*].timestamp

- Search Query:
  - List user portfolio by userId ordered by gain/loss:

```
FT.SEARCH idx:users "@userId:{u1}" SORTBY gainLoss DESC
```

  - List recent transactions by userId ordered by timestamp:

```
FT.SEARCH idx:users "@userId:{u1}" SORTBY timestamp DESC
```

Redis as vector database

What is a Vector Database?

What is a Vector Database?

- A database for storing high-dimensional vectors.
- Supports KNN (k-nearest neighbors) search using cosine, Euclidean, IP distance.
- Used in AI/ML, recommendation, semantic search, NLP.

What are Dimensions in a Vector Database?

In a **vector database**, **dimensions** refer to the number of numerical values (features) in a **vector** that represents a piece of data (like text, image, audio, or user behavior). Each vector is a point in an *n-dimensional space*, and **n** is the number of dimensions.

- **Example:** A 3-dimensional vector might look like [0.2, 0.8, 0.1].
- A **768-dimensional vector** might be generated by a large language model like BERT for a sentence.

Importance: The number of dimensions impacts **accuracy**, **storage**, **performance**, and the **relevance** of vector similarity search.

Why Dimensions Matter?

Dimension Count	Pros	Cons	Example Usage
Low (e.g. 3–10)	Fast search, simple math	Poor representation of complex data	IoT sensor data, GPS coordinates
Medium (e.g. 128–300)	Good for small NLP models or image embeddings	Trade-off between speed & quality	Face recognition, basic semantic search
High (e.g. 512–1536+)	Rich, accurate semantic understanding	Higher compute, needs optimized ANN search	LLM embeddings, RAG, GenAI apps

Example Use Cases by Vector Dimension

Use Case	Description	Typical Vector Dimension	Notes
Face Recognition	Each face image is embedded into a vector	128 or 512	Used in mobile devices and surveillance
Document Semantic Search	Each sentence or paragraph encoded	384 (MiniLM), 768 (BERT), 1536 (OpenAI)	Higher dimensions offer better semantic quality
Product Recommendations	User behavior encoded into vectors	50–200	Used in retail and e-commerce
Audio Fingerprinting	Snippets of audio converted into embeddings	128–1024	Used for identifying songs or speakers
Code Embeddings	Code snippets embedded by models like CodeBERT	768+	Used in code search and developer tools
Multimodal Search	Combined embeddings	512–1024+	Needs aligned embeddings from CLIP or similar models
Financial Fraud Detection	Transaction patterns encoded	30–100	Combines numerical, categorical, and temporal data

Redis as Vector Database

Key Capabilities

Vector Search

INDEX TYPES

Flat (KNN)

HSNW (ANN)

QUERY TYPES

Hybrid

Pre-filtering

DISTANCE METRICS

Euclidean

Cosine

Inner product

VECTOR TYPES

FP32

FP16 / BF16

Int8

Full-text Search

SEARCH CAPABILITIES

Stemming

Stop Words

Spell-checking

Fuzzy

Synonyms

Proximity

SCORING ALGORITHMS

BM25

TF-IDF

DISMAX

Hamming

Other Search Types

Numeric

Geo-Spatial

Polygon

Tags

Redis powers a multitude of AI use cases.

Use cases

Ready-to-deploy solutions

Flexible dev tooling

Fastest benchmarked vector DB

RAG | Agents | Gateways | Search / Recommender Systems | Fraud/ Anomaly Detection


RAG Context


RAG Memory


Agent State


Semantic caching


Semantic routing


Feature Stores


Rate limiter


RedisVL


Langchain

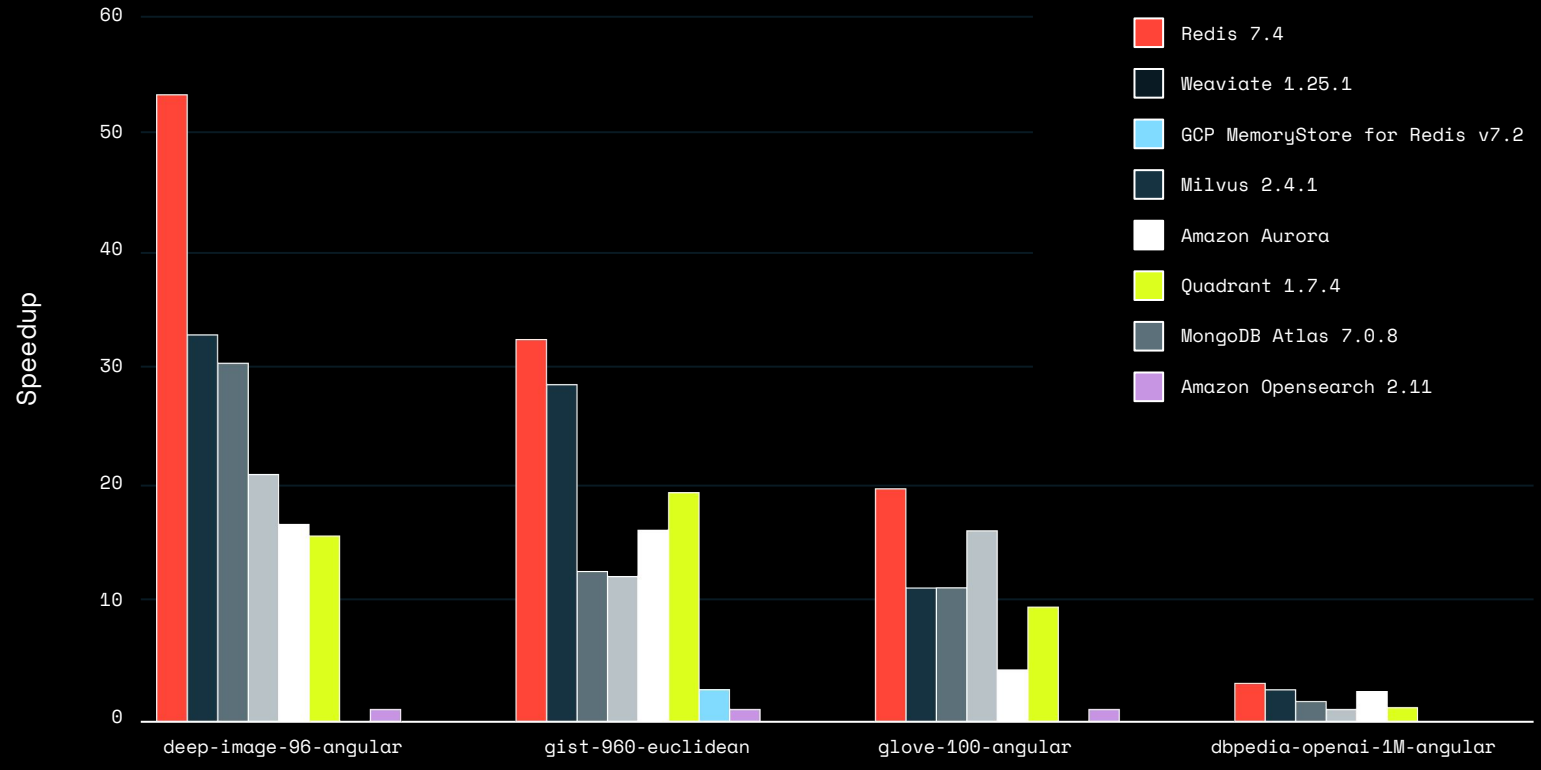

Spring AI


LlamaIndex

Redis Query Engine

Redis

Faster than every other vector database. Period.



Redis Vector Library

What is RedisVL?

- **The AI-native Python client for Redis.**
- Designed for realtime AI applications utilizing Redis' data structures and powerful capabilities.

Links

- GitHub: https://github.com/redis/redis-vl-python
- Documentation: https://redisvl.com
- PyPI: https://pypi.org/project/redisvl/

Key Features

- Native **schema** design and ergonomic **query** building.
- Lightning-fast information retrieval and **vector similarity search**.
- Built-in ecosystem integrations and utilities like common **vectorizers** and **rerankers**.
- Out-of-the-box **extensions** for common use cases like **semantic caching**, **LLM memory**, and more.

Sample dataset

- Fake dataset of users/customers including name, age, job, credit_score category, and a user embedding (vector).
- List of Python dictionaries.
- Numpy used to create vector embeddings and cast to bytes.

```python
import numpy as np

data = [
    {
        'user': 'john',
        'age': 30,
        'job': 'engineer',
        'credit_score': 'high',
        'location': '-122.4194,37.7749',
        'user_embedding': np.array([0.1,0.2,0.3],dtype=np.float32).tobytes()
    },
    {
        'user': 'mary',
        'age': 28,
        'job': 'doctor',
        'credit_score': 'low',
        'location': '-122.4194,37.7749',
        'user_embedding': np.array([0.2,0.1,0.4],dtype=np.float32).tobytes()
    },

# Additional records…

]
```

RedisVL: define a schema

What is a schema in Redis?

- Defines field types, definitions, and index configuration.
- Required to enable search in Redis.

Methods to define schema:

- **YAML**: Easy to maintain and manage. Human-readable.
- **Python Dictionary**: Directly in code for dynamic schemas.

```yaml
version: '0.1.0'

index:
  name: customers
  prefix: customer

fields:
  - name: user
    type: tag
  - name: credit_score
    type: tag
  - name: job
    type: text
  - name: age
    type: numeric
  - name: location
    type: geo
  - name: user_embedding
    type: vector
    attrs:
      algorithm: flat
      dims: 3
      distance_metric: cosine
      datatype: float32
```

```python
schema = {
    "index": {
        "name": "customers",
        "prefix": "customer",
    },
    "fields": [
        {"name": "user", "type": "tag"},
        {"name": "credit_score", "type": "tag"},
        {"name": "job", "type": "text"},
        {"name": "age", "type": "numeric"},
        {"name": "location", "type": "geo"},
        {
            "name": "user_embedding",
            "type": "vector",
            "attrs": {
                "dims": 3,
                "distance_metric": "cosine",
                "algorithm": "flat",
                "datatype": "float32",
            },
        },
    ],
}
```

RedisVL: create an index

What is a search index?

- **Secondary index** to enable efficient search across objects in Redis.
- *Optionally* overwrite existing index or delete existing data.

Methods to create:

- Init from schema **YAML or dict.**
- Init with your own **Redis client object.**
- Init with a **connection string.**

```python
from redis import Redis
from redisvl.index import SearchIndex

# bring your own client
client = Redis.from_url("redis://localhost:6379")
index = SearchIndex(schema, client)

# OR bring your connection string
index = SearchIndex(schema, redis_url="redis://localhost:6379")
```

Create the search index

```python
index.create(overwrite=True, drop=True)
```

Load data to Redis

```python
index.load(data)
```

RedisVL: query types

VectorQuery

- Standard KNN-style vector query.
- Uses vector index to compute K nearest neighbors based on the chosen distance metric.

```
from redisvl.query import VectorQuery

vector_query = VectorQuery(
    vector=[0.1, 0.2, 0.3],
    vector_field_name="user_embedding",
    return_fields=["user", "age"],
    num_results=3
)

results = index.query(vector_query)
```

VectorRangeQuery

- Yields search results within the semantic distance threshold. Not guaranteed to return anything.

```
from redisvl.query import VectorRangeQuery

range_query = VectorRangeQuery(
    vector=[0.1, 0.2, 0.3],
    vector_field_name="user_embedding",
    return_fields=["user", "age"],
    distance_threshold=0.2
)

results = index.query(range_query)
```

RedisVL: query types

FilterQuery

- Standard search + query capabilities like tag-based filters, full-text search, numeric or geospatial search.

```
from redisvl.query import FilterQuery
from redisvl.query.filter import Tag

has_low_credit = Tag("credit_score") == "low"

filter_query = FilterQuery(
    return_fields=["user", "credit_score",
"age"],
    filter_expression=has_low_credit
)

results = index.query(filter_query)
```

CountQuery

- Count the number of records in the index that match a particular filter expression.

```
from redisvl.query import CountQuery

has_low_credit = Tag("credit_score") == "low"

filter_query = CountQuery(
    filter_expression=has_low_credit
)

count = index.query(filter_query)
```

RedisVL: adding filters

All query types accept filter expressions

```
from redisvl.query import VectorQuery
from redisvl.query.filter import Tag, Text, Num, Geo, GeoRadius

has_low_credit = Tag("credit_score") == "low"
is_engineer = Text("job") % "engine*"
is_atleast_25 = Num("age") >= 25
geo_filter = Geo("location") == GeoRadius(-122.4194, 37.7749, 10, "mi")

filters = (has_low_credit & is_engineer & is_atleast_25 & geo_filter)

vector_query = VectorQuery(
    vector=[0.1, 0.2, 0.3],
    vector_field_name="user_embedding",
    return_fields=["user", "age"],
    num_results=3,
    filter_expression=filters
)

results = index.query(vector_query)
```

RedisVL: utilities

Convenience utils to help build complete workflows

Vectorizers

- Built-in vectorizers to simplify data embedding workflow.
- Support for **Cohere, OpenAI, AzureOpenAI, VertexAI, MistralAI,** and **HuggingFace** out of the box.
- Customizable vectorizer base class also available.

```
from redisvl.utils.vectorize import \
OpenAITextVectorizer

oai = OpenAITextVectorizer()

embedding = oai.embed("Sample text")
dmbeddings = oai.embed_many(
    ["Sample text", "More sample text"]
)
```

Rerankers

- Built-in rerankers to simplify search result reranking process.
- Support for **Cohere** and **HuggingFace** out of the box.
- Customizable reranker base class also available.

```
from redisvl.utils.rerank import \
HFCrossEncoderReranker

cross_encoder_reranker = HFCrossEncoderReranker(
    "BAAI/bge-reranker-base"
)
```

RedisVL: Semantic Cache

What is a semantic cache?

- A cache for natural language questions/phrases. Semantic search is used to generate cache hits.
- Typically used in an NLP search OR RAG application where the inputs are human questions.
- Save on LLM costs and improve latency/responsiveness for redundant questions.

```python
from redisvl.extensions.llmcache import SemanticCache

# init cache with TTL and semantic distance threshold
llmcache = SemanticCache(
    name="llmcache",
    ttl=360,
    redis_url="redis://localhost:6379",
    distance_threshold=0.1
)

# store user queries and LLM responses in the semantic cache
llmcache.store(
    prompt="What is the capital city of France?",
    response="Paris"
)

# check the cache with a slightly different prompt
response = llmcache.check(
    prompt="What is France's capital city?"
)
```

https://github.com/redis-developer/redis-ai-resources/tree/main/python-recipes/semantic-cache

RedisVL: LLM Short-Term Memory

What is LLM Memory?

- **LLMs are stateless.**
- Your apps host multiple sessions for multiple users in production.
- Redis provides a fast, distributed memory layer for LLMs to recall conversation history associated with a user session.
- Sometimes called “LLM session management”.

```python
from redisvl.extensions.message_history import SemanticMessageHistory


session = SemanticSessionManager(
    name="user-session",
    redis_url="redis://localhost:6379",
    distance_threshold=0.5
)

session.add_messages([
    {"role": "user", "content": "hello, how are you?"},
    {"role": "assistant", "content": "I'm doing fine, thanks."},
    {"role": "user", "content": "what is the weather going to be today?"},
    {"role": "assistant", "content": "I don't know"}
])
```

Fetch recent conversation history

```python
session.get_recent(top_k=1)
```

Fetch conversation history relevant to the term “weather”

```python
session.get_relevant(“weather”, top_k=1)
```

https://github.com/redis-developer/redis-ai-resources/tree/main/python-recipes/llm-session-manager

RedisVL: Semantic Router

What is a semantic router?

Assign incoming queries to proper handlers based on semantics:

- **Topic classification**
- **LLM selection:** choose the right LLM for the given task
- **Guardrails:** Prevent access to undesirable topic areas
- **Data segregation**: route queries to appropriate data sources

```python
from redisvl.extensions.router import Route, SemanticRouter

routes = [
    Route(
        name="greeting",
        references=["hello", "hi"],
        metadata={"type": "greeting"},
        distance_threshold=0.3,
    ),
    Route(
        name="farewell",
        references=["bye", "goodbye"],
        metadata={"type": "farewell"},
        distance_threshold=0.3,
    ),
]

# build semantic router from routes
router = SemanticRouter(
    name="topic-router",
    routes=routes,
    redis_url="redis://localhost:6379",
)


router("Hi, good morning")
```

https://www.redisvl.com/user_guide/semantic_router_08.html

When to use RedisVL?

Simplicity

User needs easy-to-use, AI-native capabilities out of the box while keeping dependencies light.

Configurability

User wants more configurability than popular AI ecosystem integrations can offer.

Persona

User has a data science or machine learning background rather than software or data engineering.

Supplemental resources

Redis AI Resource Repo

https://github.com/redis-developer/redis-ai-resources

Look here to find AI example code, recipes, and demos.

Redis - SQL to NoSQL sample code

https://github.com/suyogdilipkale/redis-document-data-to-vectorsearch

Step by step Redis JSON, Search and Vector database code samples

RedisVL

https://github.com/redis/redis-vl-python

The Redis Vector Library codifies best practices and makes it easier to get going in Python.

Join us.

A WORLDWIDE IN-PERSON EVENT SERIES

REDIS RELEASED

Sign up here:

Bengaluru

Sept 19, 2025 at 10 am

Sheraton Grand

Bengaluru

Sign up here:

Delhi

November 20, 2025 at 10 am

The Leela Ambience Gurugram

Sign up here:

Mumbai

November 13, 2025 at 10 am

JW Marriott

Mumbai Sahar

Join us.

A WORLDWIDE IN-PERSON EVENT SERIES

REDIS RELEASED

Mumbai

Sign up here:

November 13, 2025 at 10:00 am

JW Marriott Mumbai Sahar

Join us.

A WORLDWIDE IN-PERSON EVENT SERIES

REDIS RELEASED

Delhi

November 20, 2025 at 10:00 am

The Leela Ambience Gurugram

Sign up here:

thank you.