AI-Driven Multi-Vector Semantic Analysis Architecture

1. Project Overview

This project proposes a semantic query architecture leveraging AI to analyze both database field values and their contextual usage in source code. These semantics are encoded as embeddings and stored across multiple vector databases. The system enables intelligent query generation based on natural language input by combining vector search and LLM-based reasoning, thus supporting automation in querying structured data sources.

---

2. System Architecture

The architecture comprises five distinct semantic vector stores, each capturing a different facet of field-level meaning:

2.1 Vector DB A – Field Content Semantics

- Extract representative samples (e.g., first 1000 records) from each database column.
- Convert value distributions into descriptive natural language.
- Embed the descriptions using a model such as `bge-base-zh-v1.5`.
- Store the vectors in Vector DB A to represent the field’s intrinsic data semantics.

Code Example:

```
from sentence_transformers import SentenceTransformer

sample_values = ['2024-01-01', '2024-01-15', '2024-02-01']
description = f"This column contains dates like: {',
'.join(sample_values[:3])}"

model = SentenceTransformer("bge-base-zh-v1.5")
vector = model.encode(description)
```

Semantic Relation Evaluation

To determine whether two fields are semantically related based on their embeddings, the system applies a multi-strategy similarity assessment. These methods support downstream tasks such as synonym detection, schema mapping, and prompt enrichment.

Strategy 1: Cosine Similarity

- Compute cosine similarity between each pair of field embeddings.
- If similarity exceeds a threshold (e.g., 0.85), fields are considered semantically related.

```
from sklearn.metrics.pairwise import cosine_similarity

similarity = cosine_similarity([vector_field_A], [vector_field_B])[0][0]
if similarity > 0.85:
    print("Potential semantic relation identified")
```

Strategy 2: Clustering

- Apply unsupervised clustering (e.g., KMeans, DBSCAN) to group similar fields.
- Fields within the same cluster are treated as semantically aligned.

```
from sklearn.cluster import DBSCAN

clusters = DBSCAN(eps=0.3, min_samples=2).fit(all_field_vectors)
labels = clusters.labels_
```

Strategy 3: Semantic + Rule-Based Hybrid

Combine vector similarity with lightweight heuristics:

Heuristic Rule	Impact
Column name similarity (e.g., Jaccard)	Boost score
Same data type	Boost score
Description text similarity	Boost score
Value distribution overlap	Boost score
Same schema/table context	Boost score

```
def score_relation(vec1, vec2, name1, name2, type1, type2):
    score = cosine_similarity([vec1], [vec2])[0][0]
    if jaccard_similarity(name1, name2) > 0.6:
        score += 0.05
    if type1 == type2:
        score += 0.05
    return score
```

Optional: LLM-Assisted Verification

Use a local LLM to assess whether two columns are semantically equivalent based on their natural language descriptions.

```
prompt = f"""
Field 1: "Column storing order creation timestamps"
Field 2: "Date the transaction was initialized"

Are these fields semantically related? Answer Yes/No and explain briefly.
"""
result = local_llm(prompt)
```

2.2 Vector DB B – Code Usage Semantics

- Statistically analyze all CRUD operations in source code.
- Automatically extract context for each field (e.g., filter conditions, update targets).
- Convert contextual logic into natural language and generate embeddings.
- Store results in Vector DB B to reflect functional semantics from application code.

Code Example:

```
code_snippet = "order = db.query(Order).filter(Order.created_at >=
'2024-01-01')"
prompt = f"Describe the semantic intent of the fields in the following
code:\n{code_snippet}"
summary = local_llm(prompt)
embedding = model.encode(summary)
```

2.3 Vector DB C – Semantic Integration & Reasoning Layer

- Perform similarity and alignment reasoning between vectors in DB A, B, D, and E.

- Establish high-level relations such as synonym fields, entity equivalence, or data lineage.
- Store integrated relationships in Vector DB C for downstream LLM prompt enrichment.

Code Example:

```
from sklearn.metrics.pairwise import cosine_similarity

similarity = cosine_similarity([vector_a], [vector_b])[0][0]
if similarity > 0.85:
    print("Strong semantic relation found")
```

2.4 Vector DB D – View-Level Semantics (Optional Extension)

- Parse and document metadata of SQL Views including joins and aggregations.
- Generate embeddings for view descriptions and store in Vector DB D.

Code Example:

```
view_description = "This view aggregates sales data by region and
month."
vector_view = model.encode(view_description)
```

2.5 Vector DB E – Stored Procedure Semantics (Optional Extension)

- Analyze stored procedure logic and parameterization intent.
- Generate descriptive vectors and store in Vector DB E.

Code Example:

```
sp_description = "Stored procedure calculates quarterly revenue for a
given region."
vector_sp = model.encode(sp_description)
```

---

3. Query Execution Pipeline

3.1 Natural Language Input

User prompt:

“Analyze the order volume from Q1 to Q4 in 2024.”

3.2 Semantic Vectorization

```
query_vec = model.encode("Analyze the order volume from Q1 to Q4 in 2024")
```

3.3 Multi-Vector Database Retrieval

```
results_A = vector_db_A.search(query_vec)
results_B = vector_db_B.search(query_vec)
results_D = vector_db_D.search(query_vec)
results_E = vector_db_E.search(query_vec)
```

3.4 Prompt Assembly for LLM Inference

```
prompt = f"""
User Question: "Analyze the order volume from Q1 to Q4 in 2024"

[Vector DB A]:
{results_A}

[Vector DB B]:
{results_B}

[Vector DB D]:
{results_D}

[Vector DB E]:
{results_E}

Tasks:
1. Identify relevant fields.
2. Determine time range filtering logic.
3. Define aggregation method for order volume.
4. Generate the appropriate SQL query.
"""
response = local_llm(prompt)
```

3.5 Auto-Generated SQL

```
SELECT DATE_TRUNC('quarter', created_at) AS quarter,
       SUM(amount) AS total_orders
FROM orders
WHERE created_at BETWEEN '2024-01-01' AND '2024-12-31'
  AND status = 'completed'
```

```
GROUP BY quarter
ORDER BY quarter;
```

---

4. Technical Components

Component	Description
Semantic Embedding Model	BGE embedding model (`bge-base-zh-v1.5`) for vector representation
Large Language Model (LLM)	Local inference with LLaMA3, Mistral or similar open-source models
Vector Database Engines	FAISS (local), Qdrant, Weaviate (remote or cloud-native options)
Code Semantic Tools	AST-based static analysis, LLM summarizers, Regex-based extractors

---

5. Application Scenarios

- **AI SQL Copilot:** Assist users in querying databases with natural language.
- **Cross-System Field Mapping:** Discover equivalent or semantically related fields.
- **Zero-Documentation Schema Exploration:** Understand structure and logic of undocumented databases.
- **Data Governance:** Semantic lineage tracing for compliance and auditing.
- **Automated Report Generation:** Dynamically generate analytics scripts and dashboards.

---

6. Innovation Highlights

- **Multifaceted Semantic Representation:** Combines runtime data, code logic, and DB object metadata.
- **Natural Language Query Interface:** Empowers non-technical users to access data directly.
- **Automated Query Generation:** Reduces time spent on writing and debugging queries.
- **Extensibility:** Easily support additional database object types (e.g., Views, Procedures).
- **Semantic Lineage Tracking:** Enhances metadata transparency and traceability.

---

7. Future Development

- Integration with Knowledge Graphs and Ontologies
- Multilingual Model Support (e.g., Chinese, English, Vietnamese)
- Support for Heterogeneous and Unstructured Data Sources
- Incorporation of User Feedback with RLHF Fine-Tuning