Unilever POC - Architecture Overview

Unilever Procurement GPT POC - Architecture Overview

System Overview

Goal: Evaluate and monitor AI agents (Spend & Demand) with ≥90% accuracy

Key Components: 1. Evaluation Framework - Validates agent response correctness 2. Monitoring Framework - Detects drift and classifies errors

---

1. Architecture Components

Existing Layer (Unilever)

- **Spend Agent** - Handles spend queries, generates SQL
- **Demand Agent** - Handles demand queries, generates SQL

Data Collection Module

- Captures: Query, Response, Logs, Timestamp
- Format: JSON

Evaluation Framework (Feature 1)

6-Step Process:

Step 1: Pre-Processing - Clean data, extract SQL, normalize format

Step 2: Structural Validation - Check SQL syntax, schema, data types

Step 3: Semantic Check - Compare with ground truth, calculate similarity (0.0-1.0)

Step 4: LLM Judge - Model: Ollama Llama 3.1 (free, local) - Input: Query + Response + Ground Truth - Output: PASS/FAIL + Confidence + Reasoning

Step 5: Scoring - Formula: `(0.3 × Structural) + (0.3 × Semantic) + (0.4 × LLM)` - Decision: PASS if score ≥ 0.7

Step 6: Store Result - Save to Evaluation DB with all scores

Monitoring Framework (Feature 2)

Layer 1: Query Monitoring

- **Purpose:** Track all incoming queries
- **Process:** Capture query patterns, frequency, complexity
- **Output:** Query statistics for baseline comparison

Layer 2: Drift Detection

- **Purpose:** Identify when queries deviate from expected patterns
- **Process:**
  1. Convert query to 384-dim vector (embedding)
  2. Compare with baseline (1000 training queries centroid)
  3. Calculate drift score: `1 - cosine_similarity`
  4. Classify: Low (0.1-0.3), Medium (0.3-0.5), High (>0.5)
- **Trigger:** Alert if High drift detected

Layer 3: Error Detection

- **Purpose:** Identify failures in agent responses
- **Triggers:**
  - Evaluation FAIL result
  - API exceptions/timeouts
  - SQL execution errors
- **Output:** Error event with full context

Layer 4: Error Classification

- **Purpose:** Categorize errors for root cause analysis

- **Categories:**
  1. SQL Generation - syntax, schema, logic errors
  2. Context Retrieval - missing or wrong context
  3. Data Errors - missing, quality, format issues
  4. Integration - API, timeout, authentication
  5. Agent Logic - wrong reasoning, out of scope
- **Process:**
  - Rule-based: Pattern matching for common errors
  - LLM-based: Complex error analysis with Llama 3.1
- **Output:** Error category, severity, suggested fix

Layer 5: Metrics Aggregation

- **Purpose:** Calculate KPIs from all monitoring data
- **Metrics:**
  - Evaluation accuracy (overall, per-agent)
  - Drift trends over time
  - Error distribution by category
  - Error frequency and severity patterns
- **Storage:** PostgreSQL for historical analysis

---

2. Storage Layer

Database: PostgreSQL 15+ with pgvector

Table 1: queries

Purpose: Central table for all queries - query_id, query_text, agent_type - agent_response, generated_sql - status, timestamp, user_id

Table 2: evaluations

Purpose: Store evaluation results - evaluation_id, query_id, agent_type - structural_score, semantic_score, llm_score, final_score - evaluation_result (PASS/FAIL), confidence, reasoning - timestamp

Table 3: errors

Purpose: Classified errors - error_id, query_id, evaluation_id - error_category, error_subcategory - error_message, stack_trace, severity - frequency_count, first_seen, last_seen

Table 4: drift_monitoring

Purpose: Track query drift - drift_id, query_id - query_embedding (VECTOR 384) - drift_score, drift_classification - similarity_to_baseline, is_anomaly - timestamp

Table 5: baseline

Purpose: Reference for drift detection - baseline_id, agent_type - centroid_embedding (VECTOR 384) - num_queries, avg_query_length - common_keywords (JSONB)

Relationships:

```
queries (1) → (N) evaluations
queries (1) → (N) errors
queries (1) → (N) drift_monitoring
evaluations (1) → (N) errors
```

---

3. Data Flow

```
┌─────────────────────────────────────────────────────────┐
│ STEP 1: USER QUERY                                      │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ STEP 2: AGENT PROCESSING (Existing)                    │
│  • Spend Agent or Demand Agent                          │
│  • Generate SQL → Execute → Return Response             │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ STEP 3: DATA COLLECTION                                 │
│  • Capture: Query + Response + Logs + Timestamp         │
│  • Store in queries table                               │
└─────────────────────────────────────────────────────────┘
                          ↓
         ┌────────────────┴────────────────┐
         ↓                                 ↓
┌──────────────────────┐    ┌──────────────────────────────┐
│ PATH A: EVALUATION   │    │ PATH B: DRIFT DETECTION      │
│                      │    │                              │
│ 1. Pre-process       │    │ 1. Generate embedding        │
│ 2. Structural check  │    │ 2. Compare with baseline     │
│ 3. Semantic check    │    │ 3. Calculate drift score     │
│ 4. LLM judge         │    │ 4. Classify drift level      │
│ 5. Score & decide    │    │ 5. Store in drift_monitoring │
│ 6. Store result      │    │ 6. Alert if HIGH drift       │
│                      │    │                              │
│ Result:              │    │ Result:                      │
│ PASS → evaluations   │    │ Drift data in DB             │
│ FAIL → evaluations   │    │                              │
│        + errors      │    │                              │
└──────────────────────┘    └──────────────────────────────┘
         ↓                                 ↓
         └────────────────┬────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ STEP 4: ERROR CLASSIFICATION (if FAIL)                  │
│  1. Collect error context                               │
│  2. Rule-based check → Quick patterns                   │
│  3. LLM classify → Complex cases                        │
│  4. Enrich with metadata                                │
│  5. Store in errors table                               │
│  6. Alert if CRITICAL                                   │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ STEP 5: METRICS CALCULATION                             │
│  • Query all 3 tables: evaluations, errors, drift       │
│  • Calculate: Accuracy, Pass Rate, Error Distribution   │
│  • Aggregate: Drift trends, Error frequency             │
│  • Generate KPIs                                        │
└─────────────────────────────────────────────────────────┘
```

```
                         ↓
┌─────────────────────────────────────────────────────────┐
│ STEP 6: STORAGE & METRICS                               │
│  • Data stored in PostgreSQL tables                     │
│  • Metrics calculated from stored data                  │
│  • Results available for monitoring & analysis          │
└─────────────────────────────────────────────────────────┘
```