AI-Powered Trading Reconciliation Agent

Overview

This project is an **AI-powered Trading Reconciliation Agent** that detects and analyzes anomalies in trading data. It utilizes **Retrieval-Augmented Generation (RAG)** to enhance AI-driven anomaly detection by incorporating historical trade data for improved accuracy and contextual reasoning.

Features

- **Automated Anomaly Detection**: Identifies mismatches in trade data using predefined tolerance levels.
- **Retrieval-Augmented Generation (RAG)**: Fetches similar past trades from a database to provide contextual insights.
- **AI-Driven Resolution Suggestions**: Uses an LLM (via **Ollama**) to classify anomalies and recommend resolutions.
- **Database Integration**: Stores trade history in **SQLite** for fast retrieval.
- **Modular & Scalable**: Built using **Python, Streamlit, and LangChain**.

Architecture

1. Data Storage & Retrieval

- **SQLite database (**`database.py`**)** stores all historical trade records.
- **Vector Embeddings** (using `all-MiniLM-L6-v2`) enable efficient similarity-based retrieval of past trades.

2. Anomaly Detection (`reconciliation.py`)

- Scans for **quantity mismatches** exceeding tolerance thresholds.
- Triggers the retrieval pipeline for similar past cases.

3. RAG-Enhanced AI Reasoning (`reasoning.py`)

- Retrieves relevant historical trades before AI processing.
- Constructs a **structured prompt** with past case data for the LLM.
- AI model (via Ollama) **analyzes anomalies and suggests resolutions**.

Key Components and How RAG is Applied

1. Database & Retrieval Module (`database.py`)

- "This module stores all trade records and supports embedding-based search.
- When an anomaly is detected, we **fetch similar historical trades using vector embeddings** before passing them to the LLM.
- We use **sentence embeddings (MiniLM)** to measure trade similarity and **fetch relevant cases dynamically.**

This retrieval step is what enables **RAG to enhance reasoning**—instead of the AI generating generic responses, it receives actual trade data for context-aware analysis."

---

2. Reconciliation Logic (`reconciliation.py`)

- "This module scans trade records and flags mismatches where actual quantities **breach predefined tolerance thresholds.**
- Once a break is detected, the system triggers the **retrieval pipeline**, fetching similar past trades for comparison."

---

3. AI Reasoning & RAG Integration (`reasoning.py`)

- "Once historical trades are retrieved, they are packaged into a structured prompt.
- The **RAG pipeline then feeds this contextual data into the AI model (running on Ollama with `gemma:latest`).**
- The model classifies the anomaly and suggests a resolution by leveraging both historical trade patterns and its trained knowledge."

Why RAG?

> This approach ensures that every AI-generated insight is **contextually relevant**, rather than a generic response.

Setup Instructions

1. Clone the Repository

git clone https://github.com/ewfx/sradg-ai-challengers.git
 cd sradg-ai-challengers/code/src

2. Install Dependencies

pip install -r requirements.txt

3. Run the Streamlit UI

streamlit run ui.py

Steps to Run the app

Step 1: Upload Trade Data

- “We start by uploading a CSV file containing trade data. The system reads and inserts it into our SQLite database.”

Step 2: Detect Breaks

- “Once the file is processed, the system detects mismatches using predefined tolerance levels and flags them in a table.”

Step 3: Apply RAG for AI-Driven Analysis

- “For each flagged trade, clicking ‘Analyze Trade’ retrieves the most **similar past cases** from the database.”
- “These retrieved trades are then **embedded into a structured AI prompt**, enabling the model to provide precise, data-backed recommendations.”

Step 4: AI-Generated Resolution

- “The model classifies the break (e.g., data entry error, late settlement, corporate action issue).”
- “It then suggests a **contextual resolution**, explaining how similar past cases were resolved.”

Usage

1. **Upload a trade data file** (CSV or JSON).
2. The system **detects anomalies** and flags mismatches.
3. **RAG fetches relevant past trades** from the database.
4. AI **analyzes the anomaly** and provides a **context-aware resolution.**

App Screen Shots