Data Collection and Text Generation Pipeline

1. Data Collection Process (Golang)

The system collects trending token data from two primary sources using Golang-based microservices:

- **Solana Tracker API**: Data is fetched every 30 minutes to track fast-moving Solana tokens.
- **CoinMarketCap API**: Data is fetched hourly to incorporate broader market trends from a mainstream crypto provider.

A Golang-based **ETL (Extract, Transform, Load) pipeline** handles the scheduled data collection, transformation into a standardized format, and loading into the database. This ensures data consistency across sources and manages source-specific formatting requirements.

2. Data Storage and Processing (pgAI and Ollama)

Once collected, the data undergoes the following processing steps:

- **pgAI Database**: Trending token data is stored in a PostgreSQL database with AI capabilities (pgAI), enabling structured data management and vector operations.
- **Data Concatenation**: Token attributes are combined into a text representation capturing key characteristics.
- **Embedding Generation**: The concatenated text is processed using the `nomic-embed-text` embedding model on **Ollama**, converting it into vector embeddings.
- **Vector Storage**: Generated embeddings are stored directly in **pgAI**, leveraging its vector search capabilities.

This stack efficiently combines traditional database functionalities with vector-based search operations.

3. API Layer (Golang)

The system provides two primary APIs, both implemented in Golang:

1. Embedding Query API

Enables vector similarity searches using **pgAI**, allowing users to find tokens with similar characteristics or market behaviors.

2. Token Search API

Employs a three-tiered lookup approach:

1. Checks **pgAI** for trending tokens.
2. If not found, queries the **Solana Tracker API** as a fallback.
3. If still not found, queries the **Dex Screener API** as a final fallback.

This tiered system ensures comprehensive coverage while prioritizing high-relevance tokens with optimized Golang API performance.

4. Query Classification

The user's query will be classified by the SLM model. The classification can be either `Token` or `General`.

1. If the query is classified as `Token`:
   - The `Token Search API` will be called.
2. If the query is classified as `General`:
   - The `Embedding Query API` will be called.
3. The retrieved data will then be fed to the `persona-trained LLM` as context.

5. RAG Integration and Response Generation (Python)

The system integrates a **Retrieval Augmented Generation (RAG)** pipeline using Python:

- Data retrieved from APIs is used as **context** for the RAG system.
- The **Python RAG pipeline** augments prompts with relevant token information.
- A **persona-trained LLM** (Large Language Model) processes the enriched context.
- The LLM generates responses using both general knowledge and specific token data.

This Python-based implementation provides flexible integration with modern LLM frameworks, enhancing natural language responses with contextualized crypto data.

This architecture ensures efficient, real-time tracking of trending crypto tokens, robust data processing, and AI-powered insights for enhanced user interactions.