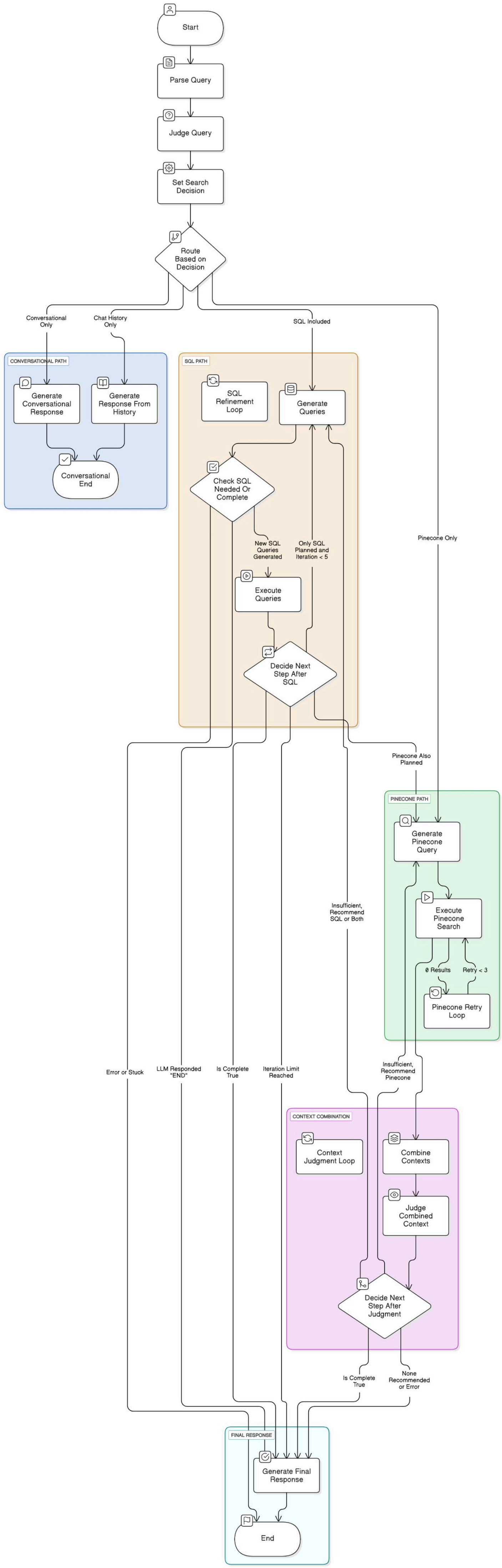
Technical Documentation

Name- Pradipta Sundar Sahoo
Enroll. No. - 22118054
Year- 3rd year,
Branch- Chemical Engg
College- IIT Roorkee
email- pradipta.ss2004@gmail.com
ph. no.- 7327873628

SYSTEM ARCHITECTURE



Workflow Explanation

1. **Start & Parse Query (run_agent)**
    - Receives user query.
    - Separates question from chat history (if present).
    - Initializes the agent's memory (AgentState).
2. **Judge Query (LLM)**
    - Analyzes query + history.
    - Decides initial strategy: SQL, Pinecone, History, Conversational, or Combo.
    - Stores decision (state.search_decision).
3. **Route Based on Decision**
    - Directs flow based on the judge's decision.
4. **Path A: Conversational / History Only**
    - generate_conversational_response: LLM crafts friendly reply for greetings/small talk → END.
    - generate_response_from_history: LLM uses only chat history to answer follow-ups → END.
5. **Path B: SQL Path (Data Fetching)**
    - generate_queries: LLM generates SQL based on query, schema, context, reasoning.
    - Check: If LLM says "END", go to Final Response. Else, continue.
    - execute_queries: Runs SQL against PostgreSQL DB. Stores results.
    - Decide:
        - If Pinecone also planned → Go to Pinecone Path.
        - If SQL only & needs more → Loop back to generate_queries (respect iteration limit).
        - If complete → Go to Final Response.
6. **Path C: Pinecone Path (Data Fetching)**
    - generate_pinecone_query: LLM creates vector search text + filters.
    - execute_pinecone_search:
        - Embeds text (SentenceTransformer).
        - Queries Pinecone index.
        - Retry Loop: If 0 results, LLM refines query (broader), retries (max 3).
    - Stores results.
7. **Context Combination & Judgment (If SQL and/or Pinecone used)**
    - combine_contexts: Merges results from SQL & Pinecone into formatted text.
    - judge_combined_context: LLM evaluates combined info for sufficiency.
    - Decide:
        - If sufficient OR max iterations → Go to Final Response.
        - If insufficient → Loop back to generate_queries (SQL needed) or generate_pinecone_query (Pinecone needed), providing reasoning.
8. **Final Response**
    - generate_final_response: LLM synthesizes a comprehensive answer using all gathered context (SQL, Pinecone, history, judgment notes).
9. **End**
    - Agent finishes, returns the final state (including the response).

Challenges Faced	Solutions Implemented
1. Scraping Data - Was unable to get indian restaurants listed and get details about food n all	Scrapped Zomato's Website to get consistent dataset
first thought: user_query → NL2SQL → end	
2. Single query may not handle fetch all contexts required for the response	Realised Simple NL2SQL won't work , Agentic framework came to picture.
2. User may not ask exact "category/ restaurant's name" .	Use Tri-Grams in postgres for fuzzy matching.
3. The features like "spicy" was not explicity mentioned dataset, but we need to understand that.	Generated description_clean , used Pinecone with metadata.(for item attributes) Tried setting up pgvector extension. (Windows error) wasted 3-4hrs

Future Improvements

1. If we are successful in **pg-vector** then we won't require, pinecone at all. (So a big architecture component will be removed.
2. Tried **Caching** the queries and results in Redis , and ask users about (using **vector similarity between queries**) if they want similar query's reply. (Time constraint)
3. **Real-time data** - use some API (if present, should be), or Cron-Jobs.
4. It is still giving structured output at each, but we should rather use structured outputs by openai -- using python class (claims 100% accuracy)
5. **Action-based Responses:** When possible, enable the agent to perform actions, like making reservations or placing an order, directly through external APIs.
6. **Proactive Assistance & Suggestions:** Offer suggestions or follow-up questions to guide the user towards more specific or helpful information.
7. Could try with **Knowledge Graph Integration:** If a knowledge graph of restaurant data is available or can be built, integrate it for richer semantic understanding and more accurate results.
8. Multi-Modal Capabilities.
9. Use **Hyper-personalization , user's buying attributes.** (Suppose a guy usually buy biryani from 'Chicken Biryani from Biryani by kilo' , Then if that guy query about "Biryani" then it should show 'chicken biryanis" "biryani by kilo"/ randomisation for variety. --- (Handled by scoring techniques , weights n all)

Just read that postgres 17 has some setup issues with pgvector.