Order Appetit Chatbot

Abstract

Order Appetit faced a critical challenge: enabling non-technical stakeholders to access actionable business insights from their complex MongoDB database. Traditional methods required technical expertise, leading to delays in decision-making and operational inefficiencies. To address this, we developed a conversational chatbot powered by large language models (LLMs) and a multi-agent system. The solution leverages Retrieval-Augmented Generation (RAG) for product name standardization and intelligent agents for schema analysis, query generation, and data analysis. This white paper outlines the problem, solution architecture, implementation details, and the transformative business impact of the Order Appetit Chatbot.

Introduction

In the competitive food delivery industry, timely and accurate data insights are essential for decision-making. Order Appetit's stakeholders faced several challenges:

- **Complex Database Structure**: The MongoDB database contained intricate relationships between orders, products, and restaurants. Querying this data required technical expertise, making it inaccessible to non-technical users.
- **Unstructured Nature of Data**: The database's unstructured nature led to inconsistencies in naming conventions for products and categories. For example:
    - Product names like "Mac n Cheese," "Macaroni & Cheese," and "Cheesy Mac" referred to the same item but were stored differently.
    - Categories lacked standardization, further complicating aggregation and analysis.These inconsistencies made it difficult to extract accurate insights about product performance or category trends.
- **Dependence on Technical Teams**: Non-technical stakeholders relied heavily on engineers to extract insights from the database. This dependency caused delays in accessing critical information needed for strategic planning.

These challenges hindered Order Appetit's ability to leverage its data for operational efficiency and strategic planning.

Founder’s Requirements

To address these challenges, the chatbot needed to provide insights into key metrics across three primary areas:

1. **Restaurant Analytics**:
    - Order volume by month and year.
    - Top-selling items.
    - Orders segmented by zip code.
2. **Order Analytics**:
    - Total order volume by month and year.
    - Orders segmented by zip code.
    - Orders analyzed by day of the week and time of day.
3. **User Analytics**:
    - New users by month and year.
    - Average value per user.
    - Predicted lifetime value of users.
    - User activity segmented by zip code.

These insights would help identify slow periods, underperforming restaurants, and general trends in user behavior and order history.

User Interface

The Order Appetit chatbot interface leverages Streamlit to provide an intuitive and responsive user experience. At its core, the interface implements a sophisticated conversation management system that enables multi-threaded discussions while maintaining context awareness.

Conversation Management

The interface incorporates a robust thread management system that allows users to create, switch between, and delete conversation threads. Each thread maintains its own isolated conversation history, preventing context bleed between different discussions. The system implements safety checks to prevent accidental deletion of active threads and provides visual feedback through warning messages when protected operations are attempted.

Memory Architecture

The conversation memory is built around the ConversationBufferWindow class, which implements a sliding window mechanism that maintains the last 5 messages of context. This design choice balances the need for contextual awareness with memory efficiency. The buffer uses a deque data structure for optimal performance in maintaining the conversation window.

Data Persistence

Currently implemented as a monolithic application, the system stores all conversation data locally using JSON-based persistence. The chat history is automatically saved and loaded, with comprehensive error handling for file operations. The system uses a custom JSON encoder to handle complex data types and ensures thread state preservation across sessions.

Technical Implementation

The conversation buffer is implemented through a deque-based system with a fixed window size:

```
class ConversationBufferWindow:
    def __init__(self, window_size=5):
        self.window_size = window_size
        self.buffer = deque(maxlen=window_size)
```

This implementation provides the foundation for future enhancements, including potential migration to a distributed architecture and cloud-based persistence while maintaining the current user-friendly interface and robust conversation management capabilities.

Proposed Solution

Overview

To address these challenges, we designed and implemented a conversational chatbot that empowers stakeholders to query business data in natural language. The system uses advanced AI techniques to bridge the gap between unstructured data and actionable insights.

Key Features

1. **Multi-Agent System**:
    - A modular architecture with specialized agents for schema analysis, query generation, and data analysis.
    - Agents work collaboratively to process user queries and deliver accurate results.
2. **RAG System for Product Name Standardization**:
    - Resolves inconsistencies in product names and categories using semantic similarity matching.
    - Ensures accurate aggregation of sales data across variations.
3. **Natural Language Interface**:
    - Built using Streamlit for intuitive user interaction.
    - Provides real-time responses with textual insights.

Initial Setup

- Conducted stakeholder meetings to understand business requirements.
- Analyzed MongoDB architecture to identify key collections (orders, products, restaurants).
- Selected CrewAI as the framework for orchestrating intelligent agents due to its robust communication capabilities.

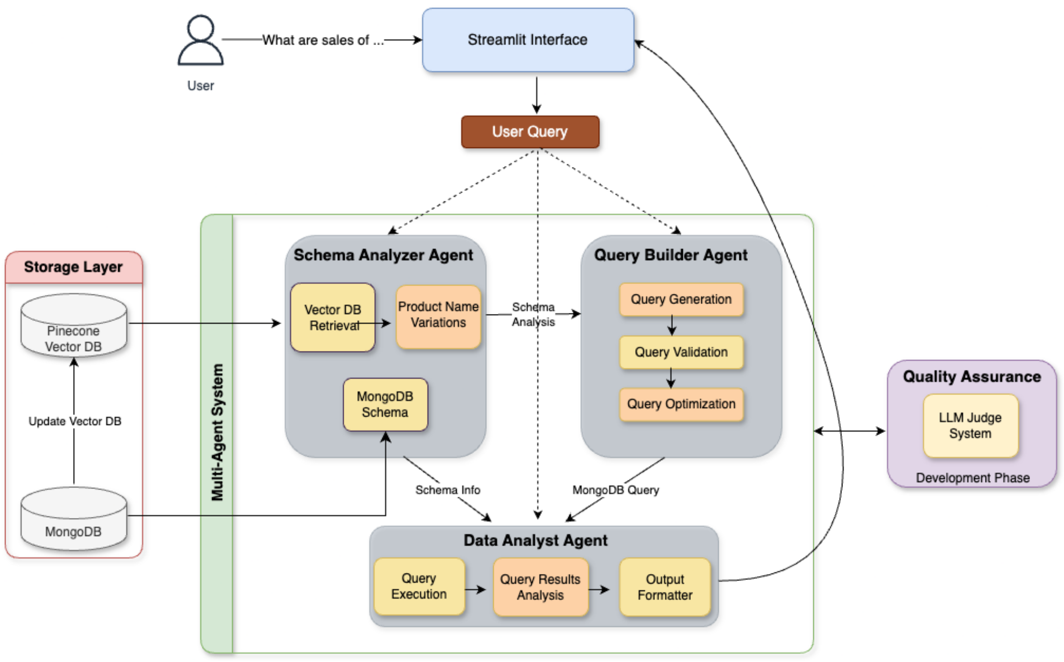
Key Challenges

1. **Database Complexity**: Querying required knowledge of schemas and relationships across collections.
2. **Product Name Standardization**: Variability in naming conventions hindered accurate aggregation of sales data.
3. **Query Optimization**: Ensuring real-time responses without overloading the database.

System Architecture

The chatbot is built on a multi-agent system with three core agents:

1. **Schema Analyzer Agent**:
    - Maps user queries to relevant database schemas.
    - Resolves product name inconsistencies using a RAG system.
2. **Query Builder Agent**:
    - Generates optimized MongoDB aggregation pipelines based on user intent.
    - Validates queries to ensure accuracy and efficiency.
3. **Data Analyst Agent**:
    - Analyzes query results to generate insights.
    - Recommends visualizations based on data characteristics.



Schema Analyzer Agent

The Schema Analyzer Agent is responsible for mapping natural language queries to relevant database schemas. It ensures accurate query generation by understanding the structure of collections, resolving inconsistencies in product naming, and optimizing schema usage to fit within LLM context windows. To achieve this, we developed three specialized tools:

1. **Pre-Optimized Schema Tool**: Analyzes pre-configured schemas stored locally.
2. **Real-Time MongoDB Schema Extraction Tool**: Connects directly to MongoDB to extract schema details dynamically.
3. **Food Items Finder Tool**: Resolves product naming inconsistencies using semantic similarity matching powered by SentenceTransformer and Pinecone.

Tools Developed for Schema Analysis

We developed two tools for schema analysis to support the agent’s functionality:

1. Pre-Optimized Schema Tool

This tool analyzes pre-configured schemas stored locally. These schemas are carefully optimized to include only meaningful fields required for querying while excluding irrelevant data to reduce token usage. Each field in the schema is annotated with detailed descriptions to provide additional context during query processing.

Key Features

1. **Schema Optimization**:
    - Retains only essential fields from each collection.
    - Reduces token usage by limiting schema size while maintaining context richness.
2. **Field Descriptions**:
    - Adds detailed descriptions to each field, improving LLM understanding of their purpose.
    - Enables accurate mapping of user queries to database fields.
3. **Error Handling**:
    - Handles missing schema paths or invalid collection names gracefully.

Optimized Schema Example

Here’s an example of a pre-optimized schema for the `orders` collection:

```
{
    "collection_name": "orders",
    "fields": {
        "createdAt": {
            "type": ["datetime"],
            "description": "Timestamp when the customer submitted their order in the system"
        },
        "store_id": {
            "type": ["ObjectId"],
            "description": "Reference ID to the store/restaurant document where the order was placed"
        },
        "user_id": {
            "type": ["ObjectId"],
            "description": "Reference ID to the customer's user document who placed this order"
        },
        ...
    }
}
```

2. Real-Time MongoDB Schema Extraction Tool

This tool connects directly to MongoDB in real time to extract the schema of any specified collection. It samples documents from the collection, analyzes their structure, and identifies field types dynamically. This tool is particularly useful when dealing with collections that are not pre-configured or when new fields are added dynamically.

Key Features

1. **Real-Time Analysis**:
    - Connects directly to MongoDB using a URI.
    - Samples documents from specified collections for analysis.
2. **Nested Field Handling**:
    - Recursively analyzes nested documents and arrays.
    - Identifies field types at all levels of document hierarchy.
3. **Dynamic Updates**:
    - Adapts to changes in collection structure without requiring manual updates.

Comparison of These Two Tools

Feature	Pre-Optimized Schema Tool	Real-Time Schema Extraction Tool
Data Source	Pre-configured JSON files	Direct connection to MongoDB
Field Descriptions	Detailed annotations included	Not included but can be added manually
Adaptability	Requires manual updates	Automatically adapts to new fields
Use Case	Fixed collections with known fields	Dynamic or evolving collections

---

3. Food Items Finder Tool

This tool resolves inconsistencies in product names by finding semantically equivalent food names using embeddings generated by SentenceTransformer and stored in Pinecone’s vector database.

Key Features

- Semantic similarity matching: Identifies similar food items based on embeddings.
- Dynamic updates: Automatically incorporates new food items into the vector database.
- Scalability: Handles large datasets efficiently using Pinecone's indexing capabilities.

The Schema Analyzer Agent combines these three tools—Pre-Optimized Schema Tool, Real-Time MongoDB Schema Extraction Tool, and Food Items Finder Tool—to deliver accurate query-to-schema mapping while resolving naming inconsistencies effectively. Together, these tools empower non-technical stakeholders with seamless access to actionable insights from Order Appetit’s complex database.

Schema Analyzer Agent Output Summary

- **Relevant Collections and Fields**:
    - Collection: `orders`
    - Fields:
        - `details.name`, `details.qty`, `details.price`, `total_amount`

- **Naming Variations for Food Items**:
    - Example: For “Mac and Cheese,” equivalent names include “Mac n Cheese,” “Macaroni & Cheese,” and “Cheesy Mac.”
- **Schema Mapping**:
    - Detailed JSON structure provided above.
- **Explanations and Assumptions**:
    - Focused on names over IDs.
    - Case-insensitive matching will be used for product names.

This detailed output from the Schema Analyzer Agent ensures that subsequent agents can generate accurate MongoDB queries tailored to user requirements while addressing challenges such as naming inconsistencies and schema complexity.

Query Builder Agent

The Query Builder Agent is designed to transform user intent into optimized MongoDB aggregation pipelines. It operates independently, leveraging schema mappings provided by the Schema Analyzer Agent to construct accurate and efficient queries. This agent ensures that the generated queries meet user requirements while maintaining performance and scalability.

Responsibilities

1. **Query Generation**:
    - Constructs MongoDB aggregation pipelines based on schema mappings and user queries.
    - Incorporates strategies for handling naming inconsistencies.
2. **Validation**:
    - Ensures the generated queries adhere to MongoDB syntax and produce accurate results.
3. **Optimization**:
    - Implements best practices for handling large datasets efficiently, such as indexing, projection, and memory optimization.
4. **Python Integration**:
    - Produces executable Python code using PyMongo to execute the query and return results in JSON format.

Expected Output

The Query Builder Agent is responsible for generating an optimized MongoDB query or aggregation pipeline based on user intent. The output of this agent must adhere to the following structure:

The agent’s output will be a JSON object containing two key components:

a. `python_code`

- **Description**: This field contains Python code that connects to MongoDB, executes the query, and returns the results as a JSON object.
- **Requirements**:
    - The code must be executable.
    - It should use PyMongo to connect to the MongoDB database.

    - The query must include all necessary steps (e.g., `$match`, `$group`, `$project`, `$unwind`) to address the user query.
    - The code should handle errors gracefully and ensure efficient execution.

b. `query_output_structure`

- **Description**: This field describes the expected format of the query results, including all fields and their data types.
- **Requirements**:
    - Provide a clear description of each field in the output.
    - Specify data types for every field (e.g., string, integer, float).
    - Include any nested structures if applicable.

Data Analyst Agent

The Data Analyst Agent is responsible for executing the MongoDB query generated by the Query Builder Agent and presenting the results in a structured, granular, and tabular format. This agent ensures that the query output is transformed into an easily interpretable format, using names instead of IDs and aligning with schema mappings provided by the Schema Analyzer Agent. The agent focuses solely on presenting data without additional analysis or interpretation.

Responsibilities

1. **Query Execution**:
    - Validates and executes the Python code generated by the Query Builder Agent.
    - Ensures proper connection to MongoDB and captures query results in JSON format.
2. **Data Transformation**:
    - Converts raw JSON output into a structured format.
    - Verifies data completeness and correctness against schema mapping.
3. **Result Formatting**:
    - Presents results in a granular tabular format with clear column headers.
    - Ensures consistency in naming conventions and alignment with schema mapping.
4. **Output Presentation**:
    - Displays results in an organized manner, sorted appropriately if needed.
    - Avoids additional analysis or interpretation beyond presenting the data.

Expected Output

The Data Analyst Agent must produce an output adhering to the following structure:

The agent's output will be a JSON object containing two key components:

a. `query_execution_status`

- **Description**: Indicates whether the query execution was successful or encountered errors.
- **Requirements**:
    - If successful, include a success message.
    - If unsuccessful, provide an apology message without hallucinating results.

b. `tabular_results`

- **Description**: A detailed, granular table of query results formatted for clarity and consistency.
- **Requirements**:
    - Use names (not IDs) for fields such as product names or restaurant names.
    - Include all necessary fields (e.g., product names, quantities, sales values) with clear column labels.
    - Maintain consistent formatting aligned with schema mapping outputs.

Quality Assurance Framework

To ensure reliability:

- Integrated LangSmith for performance monitoring of LLM agents.
- Developed automated test cases with structured evaluation criteria (accuracy, completeness, format compliance).
- Established a feedback loop using LLM Judge prompts to evaluate responses against ground truth data.

This framework focuses on evaluating the performance of LLM agents, validating query results against real-time database outputs, and maintaining compliance with structured response formats. The QA process ensures that the chatbot delivers consistent and actionable insights to stakeholders.

Key Components of the QA Framework

1. LangSmith Integration for Performance Monitoring

- **Objective**: Monitor the performance of LLM agents in real-time and track their effectiveness in generating accurate responses.
- **Implementation**:
    - Set up testing environments for individual agents (Schema Analyzer, Query Builder, Data Analyst).
    - Developed performance monitoring dashboards to track metrics such as response accuracy, latency, and error rates.
    - Implemented automated testing pipelines to validate agent outputs against predefined criteria.

---

2. Automated Test Cases

- **Objective**: Validate agent outputs using structured evaluation criteria.
- **Evaluation Criteria**:
    1. **Accuracy**: Ensure factual correctness by comparing AI-generated responses with ground truth data from the database.
    2. **Completeness**: Verify that all requested metrics are covered in the response.
    3. **Format Compliance**: Ensure adherence to JSON structure and proper formatting of fields.

Example Test Case Format:

```
test_case = {
    "inputs": {
        "query": "Show least-performing restaurants in terms of order volume for
2024",
        "context": "Analysis period: Last 180 days"
    },
    "outputs": {
        "ground_truth": get_least_performing_restaurants(db),
        "expected_format": {
            "restaurant_breakdown": [
                {
                    "store_name": str,
                    "metrics": {
                        "total_orders": int,
                        "total_value": float,
                        "average_order_value": float,
                        "volume_percentage": float,
                        "value_percentage": float
                    },
                    "monthly_performance": [
                        {"month": str, "orders": int, "value": float}
                    ]
                }
            ],
            "aggregated_metrics": {
                "total_volume": int,
                "total_value": float,
                "average_order_value": float,
                "analysis_period": {"start_date": str, "end_date": str},
                "restaurants_analyzed": int
            }
        }
    }
}
```

3. Feedback Loop with LLM Judge

- **Objective**: Evaluate AI-generated responses against ground truth data using a feedback mechanism.
- **Implementation**:
    - Designed prompts for LLM Judge to assess responses on accuracy, completeness, and format compliance.
    - Established a scoring system (1–5 scale) for each evaluation criterion.
    - Provided detailed analysis of discrepancies and suggestions for improvement.

Example Evaluation Prompt:

```
evaluation_prompt = """
You are an expert evaluator assessing the accuracy of AI-generated analytics res
ponses against actual database results.

Query: {user_query}
Ground Truth (Database Results): {database_results}
AI Generated Response: {ai_response}

Evaluate the response on the following criteria:

1. Factual Accuracy:
   - Are all numerical values exactly matching the database results?
   - Are all trends and patterns correctly identified?
   - Are there any hallucinations or incorrect statements?

2. Completeness:
   - Does the response cover all metrics present in the database results?
   - Are all time periods and categories properly addressed?
   - Is any relevant information omitted?

3. Format Compliance:
   - Does the response maintain proper JSON structure?
   - Are all required fields present and correctly formatted?
   - Is the data properly nested and organized?

Provide your evaluation in the following format:

## Evaluation ##
Accuracy Score (1-5):
Completeness Score (1-5):
Format Compliance Score (1-5):
Detailed Analysis:
- Factual Accuracy Issues:
- Missing Information:
- Format Discrepancies:
Overall Score (1-5):
Explanation:
"""
```

Benefits of QA Framework

1. Ensures high reliability and accuracy of agent outputs.
2. Provides actionable feedback for continuous improvement through automated evaluation mechanisms.
3. Maintains consistency in output formatting for seamless integration with downstream systems.
4. Validates real-time data queries against live database states, ensuring up-to-date results.

Results

Business Impact

1. Reduces time-to-insight, enabling faster decision-making.
2. Improved data accessibility for non-technical stakeholders through natural language querying.
3. Enhanced operational efficiency by automating repetitive data analysis tasks.

Technical Achievements

1. Designed a production-grade natural language analytics system using CrewAI.
2. Developed a scalable RAG system for resolving product name inconsistencies across diverse datasets.
3. Built an intuitive Streamlit-based interface with real-time visualization capabilities.

Example Use Cases

1. **Revenue Analysis**:
   Query:
   "Which restaurants had the highest revenue growth in the past three months?"
   Output: A ranked list of restaurants with revenue growth percentages and visual comparisons across periods.
2. **Product Performance**:
   Query:
   "Show me the performance of all pizza varieties across restaurants."
   Output: Aggregated metrics for pizza sales with breakdowns by restaurant and time trends.
3. **Operational Insights**:
   Query:
   "What are the least-performing restaurants in terms of order volume?"
   Output: A detailed report highlighting underperforming locations with monthly trends

---

Future Enhancements

To further improve the capabilities of the chatbot, the following enhancements are planned:

1. **Integrate Visualization in Chat**: Enable users to view data visualizations directly within the chat interface, enhancing the interpretability of results.
2. **Include Repetitive Queries as Suggestions**: Implement a feature that suggests frequently asked queries to users, streamlining the query process and improving user experience.
3. **Improve Retrieval of Names**: Enhance the system's ability to accurately retrieve food or restaurant names even when they are not spelled correctly, thereby increasing the robustness of query responses.

Conclusion

The Order Appetit Chatbot demonstrates how conversational AI can transform business analytics by making compnlex data accessible to non-technical users. Through innovative use of LLM agents, RAG systems, and intuitive design, we have empowered stakeholders with real-time insights that drive informed decision-making.