**CISC 691: A06: Building the AI Agent of Your Choice!**

1. **Agent Purpose, Use Case, and Problem Solved**

The project develops a sophisticated virtual agent that helps e-commerce consumers quickly obtain trustworthy insights from Amazon product reviews. This system addresses the issue of information overload that users face when trying to evaluate products based on a plethora of unstructured reviews. The agent combines customer feedback with structured data, such as ratings and prices, to provide users with clear summaries that help them make informed purchasing choices. Interactive dialogue features in the solution display confidence scores along with follow-up questions, enhancing trust and maintaining user engagement throughout multi-turn conversations. The agent proves to be a valuable resource for both shoppers, customer service personnel, and personal assistants operating in e-commerce fields.

1. **Design Pattern, Architecture, Workflow and Framework Selection**

Our system structure uses the Retrieval-Augmented Generation (RAG) design pattern to combine document retrieval with generative model capabilities. Our system utilizes ChromaDB as the vector database for storing and retrieving preprocessed Amazon review embeddings. OpenAI’s GPT-4 generates the final answer based on review text data and structured product information that the system retrieves.

The overall architecture is hybrid and layered. The raw Amazon CSV data starts its journey through ingestion before being sent to preprocessing and segmentation into manageable pieces. Transformer-based models available from Hugging Face Transformers convert text segments into vector embeddings that are stored in ChromaDB. The retrieval engine handles FastAPI endpoint queries by retrieving the most relevant documents and determining composite confidence scores. Retrieved text and Metadata are used by the system to form a GPT-4 prompt which subsequently produces the final answer. The system architecture achieves robust error management by combining asynchronous tasks with retry mechanisms and customizable configurations that address varying demands and real-world unpredictability.

*Workflow Design:*

A diagram of a process

AI-generated content may be incorrect.

*SVG Link for Workflow:* [*https://www.mermaidchart.com/raw/cd0dc186-4222-40dd-a729-0c28a8bb35e9?theme=light&version=v0.1&format=svg*](https://www.mermaidchart.com/raw/cd0dc186-4222-40dd-a729-0c28a8bb35e9?theme=light&version=v0.1&format=svg)

Below is an excerpt of the core code used to initialize the configuration, load the OpenAI API key, and define the FastAPI application along with request/response models:

***Code Snippet:***

A screen shot of a computer

AI-generated content may be incorrect.

This segment of code initiates the foundational setup for the agent's operations. The ConfigManager manages configuration details while FastAPI creates the web server and its endpoints. By implementing asynchronous libraries such as asyncio with proper logging practices, we achieve both system responsiveness and debuggability during high-load situations.

1. **Workflow Design**

The agent follows a workflow that consists of modular steps which execute sequentially. The agent begins its operation by loading raw data from an Amazon CSV file before processing it. The text receives tokenization before breaking into segments to comply with model size limits. Text chunks receive an embedding process, and their resulting data gets stored in ChromaDB which enables quick similarity searches.

The retrieval function activates during query processing to find and obtain the documents that show the best relevance score. The updated retrieval function demonstrated by the code snippet now makes use of existing RAG pipeline functions instead of dummy data.

A computer screen shot of text

AI-generated content may be incorrect.

The function initializes a ChromaDBRetriever object using the supplied configuration parameters then searches the vector database to retrieve relevant document segments. Asynchronous operations in the design approach minimize blocking time when waiting for I/O processes.

A composite confidence score is produced by the system through aggregation of similarity scores from retrieved documents. The function calculates a score based on the average values of document scores and converts them to percentages without applying penalty-based variation in the code snippet.A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

We combine the structured data from the retrieved documents into one context string before sending it to GPT-4. Each document must contain product\_name field as well as review information rating and price to be processed correctly within this function. Missing fields trigger the error handler during processing:A computer code on a black background

AI-generated content may be incorrect.

For generating the final answer, an asynchronous function calls GPT-4 with a retry mechanism incorporating exponential backoff. This ensures reliability when external API calls experience intermittent failures:

A computer screen shot of text

AI-generated content may be incorrect.

Additionally, the system generates follow-up questions to maintain ongoing dialogue and uses a helper function to format the answer for improved readability:

A screen shot of a computer program

AI-generated content may be incorrect.

Finally, the FastAPI endpoint brings together all these functions in a robust, error-handled workflow. When a query is received, the endpoint retrieves documents, calculates the confidence score, integrates data, generates the response via GPT-4, formats the answer, and generates follow-up questions:

A screen shot of a computer program

AI-generated content may be incorrect.

The entire system is bootstrapped by a main function that runs the server using Uvicorn with auto-reload enabled during development:

A black screen with white text

AI-generated content may be incorrect.

This code demonstrates how each component of the agent’s workflow is implemented, from configuration and asynchronous processing to error handling and response formatting.

**Postman App API:**

This section demonstrates how the POSTMAN app utilizes its built AI Agent to obtain necessary answers.

**Server Setup**

You need to ensure your FastAPI server is active before you can test it with Postman. This is achieved by starting the server with Uvicorn using a command like uvicorn agent: To start your FastAPI server for testing with Postman use the terminal command uvicorn agent:app --host 127.0.0.1 --port 8000 --reload. This command launches the server on your local machine at 127.0.0.1.1: The backend system becomes ready to process incoming requests when you launch the server at 127.0.0.1:8000.

**Creating a New Request in Postman**

Once the server starts running you can proceed to open Postman to make a new API request. Launch Postman to initiate a new request and choose POST as the HTTP method. You then provide the URL http: Enter http://127.0.0.1:8000/query in the address field to direct your request to the appropriate query processing endpoint of your API.

**Setting Request Headers**

Go to the "Headers" tab within Postman to add a header with Content-Type as the key and application/Json as the value. The header plays a critical role by directing the server to recognize the JSON format of the incoming payload so that FastAPI handles data parsing correctly.

**Defining the Request Body**

Open the "Body" tab in Postman and pick the "raw" choice while ensuring you select JSON from the format dropdown. You should enter the JSON payload here based on the QueryRequest model specification from the code. Here is an example of a valid payload:

A screenshot of a computer

AI-generated content may be incorrect.

The JSON format requires a mandatory "query" field with an optional "session\_id" that helps maintain conversation context across multiple turns.

**Sending the Request:**

You must click the Send button in Postman once you have properly set up the URL headers and the request body. The action sends your request to the currently active FastAPI server. After validating the JSON payload the server initiates its internal operations which involve document retrieval and confidence score calculation as well as structured data integration to generate an answer.

**Processing the Request on the Server:**

The server API uses the ChromaDBRetriever class to asynchronously retrieve relevant documents from a vector database when processing incoming queries. Document scores are used to calculate a composite confidence score which the system integrates with structured product details into one unified context string. Before sending it to OpenAI API with GPT-4 model for response generation the system combines the query with the constructed context string. The server tracks fundamental processing stages and manages errors effectively during its operational period.

**Reviewing the Response:**

After processing the API generates a JSON response that follows the QueryResponse model. The JSON response in Postman displays both the generated answer and confidence scores for document relevance as well as follow-up questions. The review process shows proper execution of all system tasks starting from document collection up to the final stage. response creation. Postman shows the correct HTTP error codes defined by the API when errors happen.

**Response Input & Output:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Challenges and Their Resolutions**

The key development challenge was merging freeform review content with organized product details. The ingestion process required enhancements because product names did not appear in the initial dummy data and needed extraction from the CSV file. The solution involved updating ingestion and integration functions to use a new key "product\_name" while implementing try-except blocks to avoid errors when keys were missing. Our external API requests GPT-4 require improved resilience. The generate\_gpt4\_response function addressed issues by implementing a retry loop with exponential backoff. The code successfully integrated asynchronous operations for I/O processes to avoid blocking while improving system responsiveness. Modular coding and detailed logging along with various improvements made the agent function more reliably and robustly.

1. **Reflection and Areas for Future Improvement**

The system effectively extracts product reviews from Amazon data and turns them into valuable insights. However, further improvements are possible. The introduction of advanced domain-specific embedding models will lead to better retrieval accuracy and increased confidence levels. Advanced cross-encoder re-ranking techniques improve answer accuracy by supplying enhanced contextual information. Reinforcement learning enables the dialogue management system to develop complex multi-turn conversation abilities through real-time user feedback which helps refine responses. The application of performance optimizations, which include data caching alongside batch processing multiple queries, helps decrease latency while simultaneously enhancing the user experience during production.

**Conclusion:**

The report analysis introduces an advanced RAG-enabled virtual assistant designed to evaluate e-commerce data. The modular FastAPI framework integrates advanced retrieval methods and structured data processing with asynchronous operations alongside robust error management to generate real-time insights with substantial context from Amazon reviews. The solution eliminates information overload while establishing foundational elements for increased precision and enhancements in dialogue management together with system performance optimization. Continual development work will enable the agent to provide users with increasingly accurate and captivating personalized experiences.

**GitHub Repository Link:** [**https://github.com/VVPPower/CISC-691\_A06\_AI-Agent**](https://github.com/VVPPower/CISC-691_A06_AI-Agent)

**YouTube Link:** [**https://youtu.be/v9vB0Wpmeik**](https://youtu.be/v9vB0Wpmeik)

**References:**

* 1. Professor. Don Ohara assignment of RAG pipeline is used for building AI Agent.
  2. Professor. Majid Shaalan is input from assignment 05.
  3. ChatGPT API.
  4. <https://www.postman.com/>
  5. <https://www.mermaidchart.com/>
  6. Jeong, C. (2024). A Graph-Agent-Based Approach to Enhancing Knowledge-Based QA with Advanced RAG. *Knowledge Management Research*, 99-119.