Document For: Real Estate QnA System

1.Introduction

The objective of this assignment is to design and implement a scalable QnA system that fetches accurate information related to real estate projects from a given dataset.
We aim to leverage modern techniques such as Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and other AI frameworks.
The system should be implemented as an API, showcasing feasibility, scalability, and low latency.

2**. Design Overview**

2.1 System Architecture

The system architecture consists of the following key components:

Data Ingestion and Preprocessing:

Load the dataset and preprocess it for efficient querying.

Indexing:

Use LlamaIndex to create a searchable index of the real estate projects.

Query Handling:

Implement a QnA system using a combination of LLMs (like GPT-4) and retrieval techniques (RAG) to handle user queries.

API Layer:

Flask-based API to handle incoming requests and return responses.

Validation and Hallucination Prevention:

Implement mechanisms to validate responses and prevent hallucination.

2.2 **Technology Stack**

Language Model:

GPT-4 for natural language understanding and generation.

Retrieval Framework:

RAG for combining retrieval and generation capabilities.

Indexing:

LlamaIndex for efficient indexing and searching

API Framework:

Flask for building the API

Deployment:

Docker for containerization and easy deployment.

3.Data Ingestion and Preprocessing

3.1 Loading the Dataset

The dataset is loaded from the provided Google Drive link.
The data is read into a pandas DataFrame for easy manipulation and preprocessing.

3.2 Preprocessing

Clean and normalize text data.
Extract relevant features such as project names, locations, descriptions, amenities, etc.
Handle missing values and data inconsistencies.

4**. Indexing**

4.1 LlamaIndex

LlamaIndex will be used to create a searchable index of the real estate projects.

This index will allow us to efficiently retrieve relevant documents based on user queries.

4.2 Index Construction

Tokenize and vectorize the textual data.

Create an inverted index for fast lookup.

Store metadata for each project to aid in contextual understanding.

5.**Query Handling**

5.1 Retrieval-Augmented Generation (RAG)

RAG combines the strengths of retrieval-based and generation-based approaches:

Retrieval:

Use the LlamaIndex to fetch relevant documents based on the query.

Generation:

Use GPT-3.5 to generate a coherent and contextually accurate response based on the retrieved documents.

5.2 Query Processing

Parse and understand the user query.

Retrieve relevant documents from the index.

Generate a response using the retrieved documents as context.

6. **API Layer**

6.1 Flask API

The API will be built using Flask and will expose endpoints for querying the QnA system

.**6.2 Endpoints**

/query:

Accepts user queries and returns the generated response

/validate:

Validates the accuracy of the response (optional for validation purposes)

6.3 Request Handling

Handle incoming HTTP requests.

Invoke the query handling component

Return the response in JSON format.

7.Validation and Hallucination Prevention

7.1 Response Validation

Cross-check generated responses with the original dataset.

Implement a confidence scoring mechanism to assess response reliability.

7.2 Hallucination Prevention

Use context from retrieved documents to ground the generated response.

Apply post-processing checks to ensure factual accuracy.

8. Scalability and Performance

8.1 Scalability

DesignUse efficient data structures for indexing and retrieval.

Optimize Flask application for concurrency

Implement caching mechanisms to speed up repeated queries.

8.2 Performance Testing

Conduct load testing with multiple concurrent requests.

Measure and report latency metrics.

9. Implementation

9.1 Working Solution

The code will be pushed to a Git repository.

The following sections outline the core components

9.1.1 Data Ingestion and Preprocessing

```
import pandas as pd

# Load the dataset
url = 'path_to_dataset'
data = pd.read_csv(url)

# Preprocess the data
def preprocess_data(data):

 # Implement preprocessing steps
   return data
preprocessed_data = preprocess_data(data)
```

9.1.2 Indexing with LlamaIndex

```
from llamaindex import LlamaIndex

index = LlamaIndex()
index.build(preprocessed_data)
```

9.1.3 Query Handling with RAG

```
from transformers import pipeline

retriever = index.get_retriever()
generator = pipeline('text-generation', model='gpt-3.5')

def handle_query(query):
   documents = retriever.retrieve(query)
   context = ' '.join([doc['text'] for doc in documents])
   response = generator(context + query)
   return response
```

9.1.4 Flask API

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/query', methods=['POST'])
def query():
   user_query = request.json['query']
   response = handle_query(user_query)
   return jsonify({'response': response})
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

10. Testing and Validation

10.1 Functionality Testing

Test various queries to ensure accurate and relevant responses.

Validate responses against the original dataset.

10.2 Performance Testing

Conduct load testing with tools like JMeter or Locust.

Report latency and throughput metrics.