ReportMiner:
AI-Powered Data Extraction and Query System for Structured and Unstructured Reports

Project Description

This project aims to develop an AI application that processes both structured and unstructured reports, such as PDFs, to extract and organize data using Large Language Models (LLMs) and databases. The system will ingest various reports, utilize LLMs to comprehend and extract relevant information, and populate a database with the extracted data. Users can then query the system using natural language, and the application will retrieve and present the required data efficiently. This solution is designed to simplify data management and retrieval, especially in environments where reports are diverse and users may not have technical expertise in data querying.

Objectives

1. Develop a Robust Data Ingestion Pipeline
Implement tools to process and extract text from structured and loosely structured reports (e.g., PDFs).
Ensure accurate extraction of information from various report formats, including handling of tables and layouts.

2. Implement AI-Based Data Extraction using LLMs
Utilize large language models to understand and extract key data points from unstructured text.
Fine-tune LLMs to improve accuracy in extracting domain-specific information.

3. Design and Populate a Scalable Database
Create an effective database schema for storing extracted data.

4. Enable Natural Language Query Processing
Develop a user interface for natural language queries and handle ambiguities by requesting clarifications when necessary.
Use LLMs to interpret queries and translate them into database operations.

5. Deliver User-Friendly Data Presentation
Present query results in a clear, organized, and accessible format.

Scope of the Project

Objective

To create an AI-powered system capable of:

1. Extracting relevant data from structured (e.g., spreadsheets, XML) and unstructured (e.g., PDFs, Word documents) reports.
2. Organizing extracted data into a database.
3. Allowing users to query data using natural language and retrieving results efficiently.

Technologies

- **Frontend:** Django Templates.
- **Backend:** Python, Django.
- **Database:** PostgreSQL or MongoDB.
- **AI Integration:** OpenAI APIs (e.g., GPT-4).
- **File Parsing:** Libraries like `PyPDF2`, `pdfminer.six`, or `PyMuPDF` for PDFs, and `python-docx` for Word files.
- **Containerization:** Docker (optional, for deployment).

Features

1. **File Upload:** Users can upload documents.
2. **Data Extraction:** Extract text and structured data using parsing tools.
3. **Data Organization:** Store data in a structured database schema.
4. **Query Interface:** Support natural language queries via OpenAI LLMs.
5. **Admin Panel:** For managing uploads, database records, and user permissions.

Roadmap

Phase 1: Planning and Requirements Gathering

1. **Understand Inputs:** Identify file formats (PDFs, Excel, Word) and their content structure.
2. **Define Outputs:** Clarify how users will query and what results they expect.
3. **Database Schema Design:**
    - Structured data (e.g., rows and columns from Excel).
    - Unstructured data (e.g., paragraphs from PDFs mapped to a context table).
4. **AI Needs:** Decide on specific tasks for OpenAI LLMs (e.g., summarization, key-value extraction).

Deliverables:

- Requirements Document.
- High-Level Architecture Diagram.

Phase 2: System Design

1. **Frontend:**
    - Develop a simple UI for uploading files and querying data.
2. **Backend:**
    - Set up Django with REST APIs for file upload, data retrieval, and query processing.
    - Define API endpoints (e.g., `/upload`, `/query`, `/results`).
3. **Database:**
    - Choose between PostgreSQL and MongoDB based on the data model.
    - Design tables/collections for structured and unstructured data.

Deliverables:

- Database Schema.
- API Documentation.

Phase 3: Development

3.1 File Ingestion

- Implement file upload feature in Django.
- Use libraries to extract text:
    - PDFs: `PyPDF2`, `pdfminer.six`.
    - Word Docs: `python-docx`.
    - Spreadsheets: `pandas`.

3.2 Data Extraction and Processing

- Integrate OpenAI API to:
    - Extract key-value pairs or important details.
    - Summarize unstructured data.
- Develop parsing pipelines for structured and unstructured formats.

3.3 Database Integration

- Store extracted data:
    - Relational: Tables for structured data.
    - NoSQL: Collections for unstructured text.

3.4 Query Interface

- Build a natural language query interface.
- Send queries to OpenAI API, process results, and return formatted responses.

Deliverables:

- Functional Backend APIs.
- Integrated AI Pipeline for data extraction.

Phase 4: Testing

1. **Unit Testing:** Test individual modules (e.g., file parsing, database operations).
2. **Integration Testing:** Ensure seamless integration between Django, OpenAI, and the database.
3. **User Testing:** Gather feedback on the user interface and query performance.

Deliverables:

- Test Cases Document.
- Bug Fixes and Performance Improvements.

Phase 5: Deployment

1. **Local Deployment:** Run the application locally to validate functionality.
2. **Dockerization (Optional):** Use Docker for a portable, containerized solution.
3. **Cloud Deployment (Optional):** Deploy on platforms like AWS, Heroku, or GCP.

Deliverables:

- Deployment Instructions.
- Live Application Link.

Recommendation

1. **AI Integration:**
    - Use OpenAI's `text-davinci` or `gpt-4` models for language understanding.
    - Implement rate limiting to optimize cost.
2. **Database Choice:**
    - **PostgreSQL**: If the majority of data is structured.
    - **MongoDB**: If there's significant unstructured data.
3. **Scalability:**
    - Use Celery with Django for background tasks like parsing large files.
    - Integrate caching (e.g., Redis) for frequently queried data.
4. **Security:**
    - Add authentication and authorization using Django's inbuilt system or OAuth.
    - Sanitize user inputs to avoid injection attacks.
5. **Documentation:**
    - Maintain clear documentation for APIs and the system.
    - Add inline comments to your code.

Next Steps

1. Start with the requirements gathering phase.
2. Install Django, set up a basic project structure, and design your database schema.
3. Implement file upload and processing as the first functional module.

1. Requirement Analysis

- **Objective:** Understand and document the functional and non-functional requirements.
- **Tasks:**
    - Identify input file types (e.g., PDFs, Excel, Word).
    - Specify expected outputs (e.g., tabular data, key-value pairs, summarized data).
    - Define user roles (e.g., admin, regular user).
    - List potential queries and desired result formats.
    - Document constraints like file size limits, response time, and cost optimization for LLM usage.

2. Technology Selection

- **Objective:** Choose the right tools and frameworks for each component.
- **Tasks:**
    - **Frontend:** Django Templates (or React if you need a dynamic frontend).
    - **Backend:** Django REST Framework for APIs.
    - **Database:** PostgreSQL for structured data or MongoDB for unstructured and hierarchical data.
    - **AI/ML:** OpenAI APIs (GPT-4) for natural language processing and data extraction.
    - **File Parsing:** Use `PyPDF2`, `pdfminer.six`, `PyMuPDF`, `pandas`, or `python-docx` as needed.
    - **Deployment:** Docker for containerization, and cloud platforms like AWS or Heroku for hosting.

3. System Design

- **Objective:** Create an architectural blueprint for the system.
- **Tasks:**
    - **High-Level Architecture:**
        1. User uploads a file.
        2. Backend extracts data and stores it in a database.
        3. Users query the system in natural language.
        4. Backend retrieves data, processes the query using LLMs, and returns results.

- **Modules to Design:**
    1. **File Ingestion Pipeline:**
        - Upload functionality with validation.
        - Text extraction from files.
    2. **Data Extraction Module:**
        - Process text using LLMs to extract relevant details.
    3. **Database:**
        - Design schema for structured (tables) and unstructured (key-value or JSON) data.
    4. **Query Module:**
        - Process natural language queries using LLMs.
        - Translate queries into database operations.
    5. **Data Presentation:**
        - Format results into readable tables, JSON, or graphs.
- **Flow Diagram:** Draft a flowchart or sequence diagram to visualize the data flow.

4. Development Phase

4.1 File Upload and Validation

- **Steps:**
    - Create an upload endpoint in Django.
    - Validate file formats, size, and metadata.
    - Save the file temporarily for processing.

4.2 Text Extraction and Preprocessing

- **Steps:**
    - Implement parsing tools to extract raw text from the uploaded files.
    - Preprocess the text (e.g., clean, tokenize, and handle special characters).

4.3 AI-Based Data Extraction

- **Steps:**
    - Use OpenAI API to extract key data points.
    - Fine-tune queries to extract domain-specific information (e.g., tables, summaries).
    - Test accuracy and refine prompts for LLMs.

4.4 Database Design and Integration

- **Steps:**
    - Design tables/collections to store structured and unstructured data.
    - Write Django ORM models or use SQLAlchemy for database operations.
    - Implement data insertion scripts after LLM processing.

4.5 Query Module

- **Steps:**
    - Design an endpoint to accept natural language queries.
    - Use LLMs to interpret the query and map it to database operations.
    - Fetch results and format them appropriately.

4.6 User Interface

- **Steps:**
    - Build forms for file upload and query submission.
    - Add a results page to display query outputs (tables, summaries, etc.).

5. Testing

- **Objective:** Ensure the system works as intended and is user-friendly.
- **Tasks:**
    - **Unit Testing:** Test individual components like file upload, data extraction, and query processing.
    - **Integration Testing:** Test the flow between file upload, LLM processing, database storage, and query results.
    - **Performance Testing:** Measure response times for large files and complex queries.
    - **User Testing:** Collect feedback on usability and functionality.

6. Deployment

- **Objective:** Make the system available for use.
- **Tasks:**
    - **Local Deployment:** Test the complete system locally.
    - **Dockerization:** Containerize the application for portability.
    - **Cloud Hosting:** Deploy the system to a cloud provider (e.g., AWS, Heroku).
    - **Monitoring:** Set up logging and monitoring tools to track system health and usage.

7. Maintenance and Optimization

- **Objective:** Keep the system updated and optimized.
- **Tasks:**
    - Optimize database queries for faster retrieval.
    - Regularly update LLM fine-tuning for better domain relevance.
    - Add features based on user feedback (e.g., support for new file formats).

Flow Summary

1. **Input:** The user uploads a file (PDF, Word, etc.).
2. **Processing:**
    - Extract raw text using parsing tools.
    - Use LLMs to extract relevant information and populate the database.
3. **Storage:** Save structured data in a relational database or NoSQL database for unstructured content.
4. **Query:** User submits a natural language query.
    - LLM interprets the query and retrieves results from the database.
5. **Output:** Present the results in a user-friendly format (e.g., tables, graphs).