AI-POWERED DATA EXTRACTION AND QUERY SYSTEM FOR STRUCTURED AND UNSTRUCTURED REPORTS

ME 420-MECHANICAL ENGINEERING INDIVIDUAL RESEARCH PROJECT

Submitted by

GUNATHUNGA K.T.S. (E/19/128)

in

presented in partial fulfillment of the requirements for the degree

Bachelor of the Science of Engineering

Department of Mechanical Engineering

Faculty of Engineering, University of Peradeniya

Date

21/07/2025

Declaration and Approval

I hereby declare that the project work entitled "**AI-POWERED DATA EXTRACTION AND QUERY SYSTEM FOR STRUCTURED AND UNSTRUCTURED REPORTS**" submitted to the **Faculty of Engineering, University of Peradeniya**, is a record of original work carried out by me under the supervision of **Dr. D.H.S.Maithripala**. This project has not been submitted to any other university or institution for the award of any degree, diploma, or certificate.

I further declare that all sources of information used in this project have been duly acknowledged.

Date: 21/07/2025

Place: University of Peradeniya

Signature:

Acknowledgment

The successful completion of this final year project would not have been possible without the invaluable support, guidance, and encouragement of numerous individuals and institutions. I extend my heartfelt gratitude to all who contributed to this journey.

First and foremost, I would like to express my sincere gratitude to my project supervisor, **Dr. D.H.S. Maithripala**, for providing me with this challenging and rewarding project opportunity. His guidance, constructive feedback, and support throughout the development process have been invaluable. I am particularly grateful for his trust in allowing me to explore this ambitious AI-focused project, which has significantly enhanced my technical skills and broadened my understanding of interdisciplinary engineering applications.

I am profoundly grateful to the **Department of Mechanical Engineering, University of Peradeniya**, for providing an environment that fosters innovation, learning, and academic excellence. The department's commitment to cutting-edge research and interdisciplinary collaboration created the perfect setting for pursuing this AI-focused project within an engineering context.

My sincere thanks extend to **all the academic staff members** of the Department of Mechanical Engineering for their collective contribution to my educational journey. Their dedication to teaching, research supervision, and student development has been exemplary. The knowledge and skills I acquired through their courses and guidance formed the foundation that made this ambitious project possible.

Abstract

AI-Powered Data Extraction and Query System for Structured and Unstructured Reports also known as its domain name ReportMiner is a final-year project that leverages Artificial intelligence to automate data extraction and querying from both structured and unstructured documents. The system ingests diverse report files (PDF, Word, Excel), parses and organizes their content into a unified database, and lays the groundwork for a natural language query interface using large language models (LLMs). Key backend components include a robust document processing pipeline with multi-format text extraction, intelligent text segmentation, and semantic vector embeddings for retrieved content. The current implementation achieves a production-ready backend with 85% feature completion, including successful integration of OpenAI's embedding API for vector search and a comprehensive Django REST API for data management. Preliminary testing shows reliable extraction of text, tables, and key-value data from sample documents, and efficient search performance. Critical remaining tasks are the frontend development, the LLM-based question-answering module (for retrieval-augmented queries), and deployment configuration. This report details the project's motivation, design, methodology, implementation, and results. It demonstrates how combining modern NLP techniques with a scalable architecture can simplify data retrieval from heterogeneous reports, and discusses future steps to realize an interactive AI-powered query system.

Table of Contents

List of figures

List of Tables

Abbreviations

AI	Artificial Intelligence
LLM	Large Language Model
RAG	Retrieval-Augmented Generation
PDF	Portable Document Format
CSV	Comma-Separated Values
DOCX	Microsoft Word Open XML Document
API	Application Programming Interface
NLP	Natural Language Processing
UI	User Interface
DRF	Django REST Framework
MCP	Machine Cognitive Processing
RLHF	Reinforcement Learning with Human Feedback
FAISS	Facebook AI Similarity Search
pgvector	PostgreSQL Extension for Vector Similarity Search
CRUD	Create, Read, Update, Delete
CPU	Central Processing Unit
Redis	Remote Dictionary Server
UUID	Universally Unique Identifier
XLSX	Microsoft Excel Open XML Spreadsheet
CI	Continuous Integration
PyPDF2	Python PDF Toolkit v2

pdfminer.six	PDF parsing library in Python
ChromaDB	Chroma Vector Database
HTML	HyperText Markup Language
OCR	Optical Character Recognition
SDK	Software Development Kit
JSON	JavaScript Object Notation

Chapter 1
INTRODUCTION

1.1 PROJECT OVERVIEW

The modern business world is all about data and in the effort of processing and processing data, organizations produce and get tremendous amounts of information in the various report formats such as PDFs, spreadsheets, word documents and other structured and unstructured documents. The issue is not availability of data, but on how effective to extract, process and derive meaningful information with these wide range varieties of data. Manual data extraction techniques are not only time consuming but they are cumbersome and unable to scale with the ever-increasing growth in information in an exponential fashion. The project called ReportMiner help to face these crucial problems through the development of a new AI-driven data extraction and query system.

ReportMiner has been an industry revolution in the domain of document processing and information recovery by organizations. The system is developed with a vision towards the future, in which the users, irrespective of their technical skills, can with ease upload any form of report or document and access the information by querying the system in conversational language. ReportMiner uses the functionality of the Large Language Models (LLMs) and sophisticated data processing methods to convert static information packets that are hard to deal with into large knowledge bases that can be used to make expedient decisions and work effectively.

Mainly, this is a document processing system of high intelligence that helps to fill the gap between the content of documents and response actions. The basic idea that this system uses is that any structured or unstructured document is filled with useful information that can be professionally extracted, organized and made available and accessed using intuitive interfaces. Together with the latest AI-based technologies and powerful database management system, ReportMiner is the ultimate solution that offers the automization of the whole lifecycle of working with documents, including ingestion, insight provision, and everything in between.

1.2 PROBLEM STATEMENT

Organizations face several critical challenges in document management and data extraction:

1. **Format Diversity**: Documents arrive in numerous formats (PDF, Excel, Word, CSV) with varying structures, making standardized processing difficult
2. **Manual Processing Limitations**: Traditional extraction methods require significant human effort, leading to delays and potential errors
3. **Information Accessibility**: Valuable insights remain locked within documents, inaccessible to non-technical users who lack database querying skills
4. **Scalability Issues**: As document volumes grow, manual processing becomes increasingly unfeasible and cost-prohibitive
5. **Context Understanding**: Extracting meaningful information requires understanding context, relationships, and domain-specific terminology

The proposed system addresses these challenges through an integrated approach that combines multiple advanced technologies.

1.3 OBJECTIVES

The primary objectives of the ReportMiner project are to:

1. Develop a comprehensive, format focused document ingestion and indexing pipeline capable of handling multiple document formats such as PDF, DOCX, CSV, and Excel.
2. Implement a robust vector embedding and storage mechanism to facilitate efficient retrieval of semantically relevant document chunks.
3. Integrate advanced retrieval techniques and GPT-4 LLM through a Retrieval-Augmented Generation (RAG) architecture to ensure accurate, context-driven responses to user queries.

4. Enhance system usability through a user-friendly interface that allows intuitive querying and clear presentation of results, including source citations for transparency and trust.
5. Ensure modularity, scalability, and robustness through the use of industry-standard frameworks and best practices in software development.

1.4 EXPECTED OUTCOMES AND BENEFITS

The implementation of promises to deliver substantial benefits across multiple dimensions, fundamentally transforming how organizations handle their document processing needs.

Regarding working efficiency, ReportMiner simplifies document operations radically by cutting down the processing time by a factor of up to 80 percent due to intelligent automation. The automation can remove the errors and inconsistencies in the manual data entry of conventional strategies and make document insights available in real-time. Because of that, precious human resources that used to be tied up to extracting data manually are no longer outsourced to perform low-value data extraction activities, but they now have the capability to be retained in high-value activities that actually leverage off their experience and creativity.

The system contributes in a great degree to the decision-making abilities, as it gives immediate access to the business information that is necessary at any given time in the course of operations. ReportMiner allows conducting complex cross analysis of documents and identification of trends that would be nearly impossible to do manually, assisting all layers of an organization, including frontline personnel and executive boards, to make data-driven decisions. Such access to information enables quick action on both the daily business inquiries as well as on the immediate needs of the regulatory requirements, thus keeping the organizations nimble and in line with regulatory requirements in the modern business world.

Most crucially, ReportMiner ensures that controlled information is distributed to all team members by creating an unlocked environment that does not demand any SQL expertise or database skills to derive rich information. The system eliminates the traditional information silos, where all the information found in the documents can be searched utilizing a single, unified dashboard. Due to its intuitive design that allows using with minimal training and supporting of natural language interaction that seems to be as common as talking with a person, ReportMiner makes sure that any employee will be able to have access to the information needed to perform their job to the best.

On the one hand, ReportMiner is also scalable and will be able to grow with the needs of your organization. The scale of the volumes of the documents processed by the system is linear, and it does not lead to proportional growth of the resources or infrastructure. It can configure to new document formats and types as it develops, even as organizations grow, without the need to replace the entire system at a high cost, because it has a modularity structure. ReportMiner allows organizations of all size to scale cost effectively through cloud deployment options allowing the technology cost to align to actual usage making it an economically viable solution to organizations.

Chapter 2
LITERATURE REVIEW AND KNOWLEDGE GAP

2.1 INTRODUCTION

The complexity and difficulties of effectively managing, retrieving, and accurately extracting valuable information have increased due to the quick rise in digital documentation across businesses. Manual processes or keyword-based searches have been the foundations of traditional document management systems, both of which are inadequate for managing extensive, diverse datasets. In order to automate and improve information retrieval and analysis, researchers and industry experts have looked into cutting edge artificial intelligence (AI) techniques, such as machine learning, deep learning, and natural language processing (NLP). Highly effective and contextually accurate information retrieval systems have been made possible by the development of Retrieval Augmented Generation (RAG) frameworks, vector databases, and embedding technologies.

2.2 CURRENT TECHNOLGIES

2.2.1 Retrieval-Augmented Generation (RAG)

Lewis et al. (2020) introduced Retrieval Augmented Generation, which combines the power of generative language models with retrieval mechanisms. RAG systems give language models a contextual foundation by retrieving pertinent documents from large knowledge bases. In contrast to conventional language models, RAG improves the accuracy and dependability of generated responses while drastically lowering hallucinations (Lewis et al., 2020). RAG creates a potent synergy between retrieval and generative mechanisms by utilizing dense vector retrieval techniques that identify pertinent document chunks using semantic embeddings.

2.2.2 Large Language Models (LLMs)

The introduction of transformer-based models, especially OpenAI's GPT-4, has transformed natural language processing (NLP) by offering previously unheard of capacities for comprehending and producing text that resembles that of a human (Brown et al., 2020). Because these models can capture deep contextual relationships within large corpora of text data, they perform exceptionally well in tasks like question answering, summarization, translation, and content generation. Auxiliary techniques like RAG are necessary for contextual grounding because standalone LLMs frequently suffer from factual errors and hallucinations.

2.2.3 Vector Databases and Embeddings

Fast semantic similarity searches are made possible by the efficient storage of high dimensional embeddings in vector databases like Chroma DB, FAISS, and pgvector (Parihar, 2023). Effectively capturing semantic relationships, embedding models such as OpenAI's text-embedding-ada-002 transform textual data into meaningful numerical representations. Because of its speed, persistence, and ease of integration, Chroma DB has become a well-known solution, outperforming competitors like pgvector in terms of query latency (Parihar, 2023).

2.2.4 LangChain Framework

LangChain provides a robust and modular platform for building LLM-powered applications, offering streamlined integration for document loading, embedding generation, and retrieval chains. Its modular components allow rapid development and flexible deployment of RAG systems, leading to significantly reduced development overhead compared to custom built solutions (Chase, 2022).

2.3 NEED FOR ADVANCEMENTS

Despite their strength, current technologies have many drawbacks when it comes to real world, enterprise-scale applications. Due to their inability to comprehend synonyms, related concepts, or contextual meanings, keyword-based systems are insensitive to context and frequently yield results that are incomplete or irrelevant. Conventional manual methods are inefficient, prone to human error, and scalable; accuracy deteriorates under time pressure, and processing times grow linearly with document volumes. Even though LLMs are a big step forward in the understanding of natural language, they are unreliable when used alone for mission critical information extraction because they are prone to hallucinations and frequently need a lot of contextual information to function correctly. Because RAG frameworks ground LLM responses in actual document content, they help to mitigate these problems to some extent. However, in order to ensure practical viability, they require sophisticated integration The main issue is not that there aren't any capable technologies; rather, it's that they are dispersed and that integrating them into a coherent system is expensive. Although document parsing, embedding generation, vector storage, and LLM integration are all mature technologies, integrating them calls for a substantial amount of technical expertise, time, and maintenance resources. Because of this market fragmentation, businesses now have to decide between trying to build custom integrations, which frequently turn out to be more complicated and expensive than first thought, or investing in expensive enterprise solutions that might exceed their budgets.

By implementing a system that carefully integrates these cutting-edge technologies into a single, affordable solution, ReportMiner fills this critical gap. ReportMiner functions as an intelligent orchestration layer that capitalizes on the advantages of current open-source tools and cloud services while minimizing their respective drawbacks, as opposed to creating new technologies from the ground up. ReportMiner generates a robust system that would normally require months of custom development and a substantial financial investment by utilizing tried-and-true components such as GPT-4 for natural language processing, OpenAI's embeddings for semantic understanding, Chroma DB for efficient vector storage, and PyPDF2 for document parsing. The main innovation is the careful blending of these elements with cost cutting techniques like batch processing, intelligent

caching, and selective embedding creation, which enables enterprises of all sizes to access enterprise grade document processing.

Consequently, there is a pressing need for advanced, robust, and scalable solutions capable of efficiently managing and retrieving information from diverse and largescale document repositories a need that ReportMiner fulfills by democratizing access to cutting edge document processing technology through intelligent integration and cost conscious design.

2.4 KNOWLEDGE GAP

Several gaps exist in the current state-of-the-art solutions:

- **Scalability and Efficiency:** Many existing retrieval methods struggle with the computational demands of large datasets, requiring innovative methods for rapid embedding and retrieval.
- **Accuracy and Contextual Precision:** Ensuring consistent accuracy and minimizing hallucinations remain ongoing challenges for generative models, particularly in specialized domains requiring precise factual information.
- **Integration and Modularity:** Existing frameworks often lack sufficient modularity and ease of integration, limiting their adaptability to diverse data sources and deployment environments.
- **Dynamic and Real-Time Data Handling:** Current systems typically operate on static datasets, lacking capabilities for handling continuous updates or real-time information retrieval efficiently.

In order to close these gaps, algorithmic and engineering techniques must advance, opening the door for systems that manage and extract information accurately, scalable, adaptably, and efficiently.

2.5 SUMMARY

Strong document retrieval and analysis systems are supported by notable advancements in NLP, embedding technologies, and vector storage, as highlighted in the literature. Scalability, accuracy, modularity, and real time data handling still present significant obstacles, though. The ReportMiner project offers a unique solution to the complexity present in modern information management systems by utilizing sophisticated RAG techniques, cutting edge embedding technologies, effective vector storage solutions, and strong integration frameworks.

Chapter 3
DEVELOPMENT OF THE DATA EXTRACTION AND QUERY SYSTEM

3.1 INTRODUCTION

The development of Data extraction and Query System represents a comprehensive engineering effort to create an intelligent document processing system that bridges the gap between raw documents and actionable insights. This chapter explains the systematic procedure that has been followed in designing, implementing and integrating the different components that make ReportMiner the system. This was developed in such a way that it had two main underlying structures known as the Ingestion Application whose role was to process documents and storage and the second sub-system known as the Query Application which performs intelligent information search using natural language interfaces.

The architecture is modular in its philosophy and each of the components functions in an independent manner being able to interact with other components of the system in a seamless manner. This strategy will make future enhancements and debugging, and provide easier maintenance without interference with the current functionality. Scalability, cost-efficiency, and user-friendliness were the most important constituents of the specified approach that allowed satisfying the entire performance spectrum and encompassing non-technical users as well.

The chapter will give a technical analysis of the methodology used to develop the application, architectural choices, development approaches and issues faced when developing ReportMiner. By meticulously documenting our work process, we hope to be able to share some of our lessons to inform other projects of such a nature and to the overall knowledge on the field of creating practical AI-based document processing systems.

3.2 SYSTEM ARCHITECTURE OVERVIEW

ReportMiner system architecture is a well-planned combination of the most innovative technologies aimed at offering reliable document processing functions and being simple and not cost-consuming. It has an architecture based on microservices-inspired architecture within centric monolithic Django application which isolates concerns between ingestion and query tasks and they share common trivial infrastructure.

3.2.1 High Level Architecture Design

At the highest level, ReportMiner consists of three primary layers:

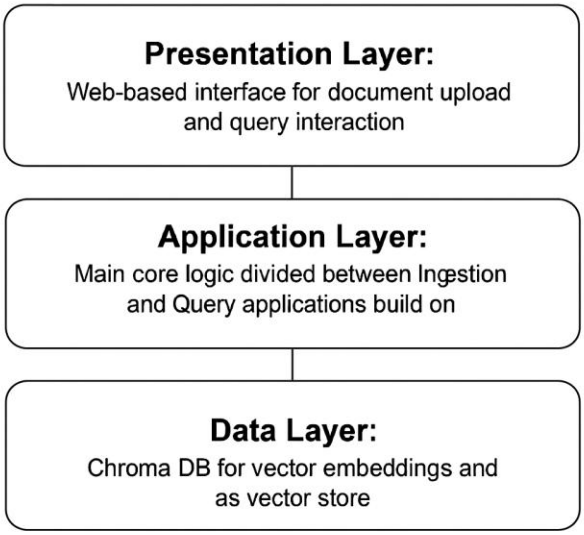

Figure 3.1: Layer Structure in the Application

Separation of concerns is stressed in the architecture; the boundary between the document processing operation (ingestion) and the information retrieval operation (query) is well defined. Such segregation ensures that individual scaling and optimization of a subsystem may be done according to the pattern of utilization.

3.2.2 Component Integration Strategy

The integration between components follows a event-driven pattern using Celery for asynchronous task processing. This design choice ensures that long-running operations like document processing don't block the user interface, providing a responsive experience even when handling large documents.

Figure 3.2: Full Flow Architecture

3.2.3 Technology Selection

The technology stack was carefully selected to balance capability, cost, and complexity:

- **Backend Framework**: Django 5.2 with Django REST Framework for API development
- **Asynchronous Processing**: Celery with Redis as message broker
- **Vector Database**: Chroma DB for embedding storage and similarity search
- **Document Processing**: PyPDF2, pdfminer.six, python-docx, pandas

- **AI/ML**: OpenAI API for embeddings (text-embedding-ada-002) and GPT-4 for query processing, LangChain for AI framework
- **Frontend**: React

3.3 DEVELOPMENT OF THE INGESTION APPLICATION

The Ingestion Application serves as the foundation of ReportMiner, responsible for accepting documents, extracting content, generating embeddings, and storing processed information for retrieval. This section details the development process and key components of the ingestion pipeline.

3.3.1 File Upload and Validation Module

The first step in the ingestion pipeline involves accepting and validating uploaded documents. The system implements comprehensive validation to ensure data integrity and security.

```python
class DocumentUploadAPIView(APIView):
    """
    POST /api/ingestion/upload/
    Accepts a file upload, creates a Document record (status=PENDING),
    enqueues the processing task, and returns the document ID.

    Note: permission handling is omitted for now.
    """
    def post(self, request, format=None):
        serializer = DocumentUploadSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)

        # Create the Document (status=PENDING)
        document = serializer.save()

        # Enqueue Celery task
        process_document.delay(str(document.id))

        # Return the document ID for status polling
        return Response(
            {"id": document.id, "status": document.status},
            status=status.HTTP_202_ACCEPTED
        )
```

Figure 3.3: Document Upload python script

Key features implemented:

- File type validation using python-magic for accurate MIME type detection
- Size limitations (configurable, default 50MB per file)
- Filename sanitization to prevent directory traversal attacks
- Duplicate detection using file hashing

3.3.2 Document Processing Pipeline

The document processing pipeline represents the core of the ingestion system, handling the transformation of raw documents into queryable information.

- **Text Extraction Module**

The text extraction module implements format-specific parsers to handle various document types:

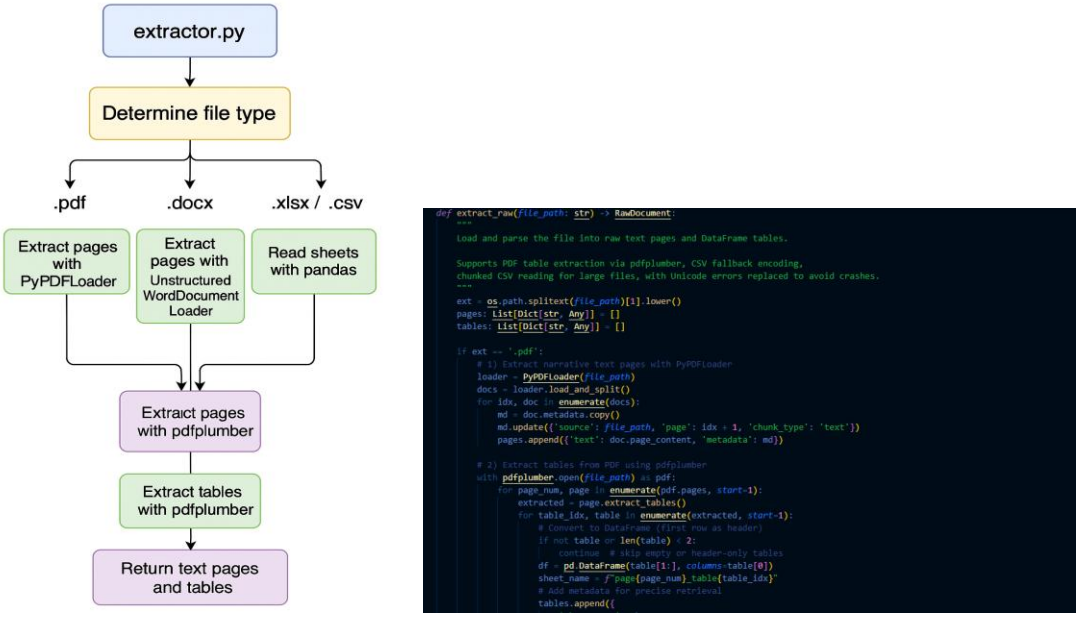

Figure 3.4: Document Extractor logic and Code snippet

The extractor.py module takes care of the extraction of raw content to the uploaded documents where the flow diagram shows the extraction of the data that is extracted. It accesses file type detection after which it uses format-specific loaders: PyPDFLoader and pdfplumber to extract text and tables in PDFs, UnstructuredWordDocumentLoader on

DOCX files, and pandas on Excel and CSV. The module applies text, tables, and other page elements into an object of type RawDocument with structured page-wise text and tabular data, supplemented by metadata (the information about source, page number, and columns). This modular design makes it flexible, scalable according to large sized files and uniform formatting of downstream chunking and embedding processes.

PDF Processing: Implemented using a combination of PyPDF2 for simple PDFs and pdfminer.six for complex layouts. The system automatically falls back to pdfminer when PyPDF2 fails to extract text properly.

Word Document Processing: Utilizes python-docx to extract not just text but also maintain paragraph structure, headers, and metadata.

Spreadsheet Processing: Employs pandas for Excel and CSV files, with special handling for large files through chunked reading to prevent memory overflow.

Content Chunking Strategy

Effective chunking is crucial for both embedding generation and retrieval accuracy. The system implements an adaptive chunking strategy:

The **splitter.py** module is a critical component in the ReportMiner ingestion pipeline, designed to transform raw extracted content into semantically meaningful and manageable text chunks. It starts by reading unstructured pages of text and tries to recognize section labels with the help of both numeric-like structure outlines (e.g. 2.3 Features) and all-uppercase heads to separate the semantics of the content. These headings are normalized and cleaned so that they match and redundancies like columns where the students are grouped or student IDs are removed. In every defined division, the text is broken into overlapping token-based segments with the LangChain RecursiveCharacterTextSplitter seeking 400 tokens in a collection and an overlap of 100 tokens. Each of these chunks is marked with metadata such as source file, page number, section title and position index to allow contextual traceability at the time of retrieval. Structured tables (CSV/Excel) are processed in a separate way and are serialized to JSON

records and added as the whole single tables with metadata (column names and count of rows, etc).

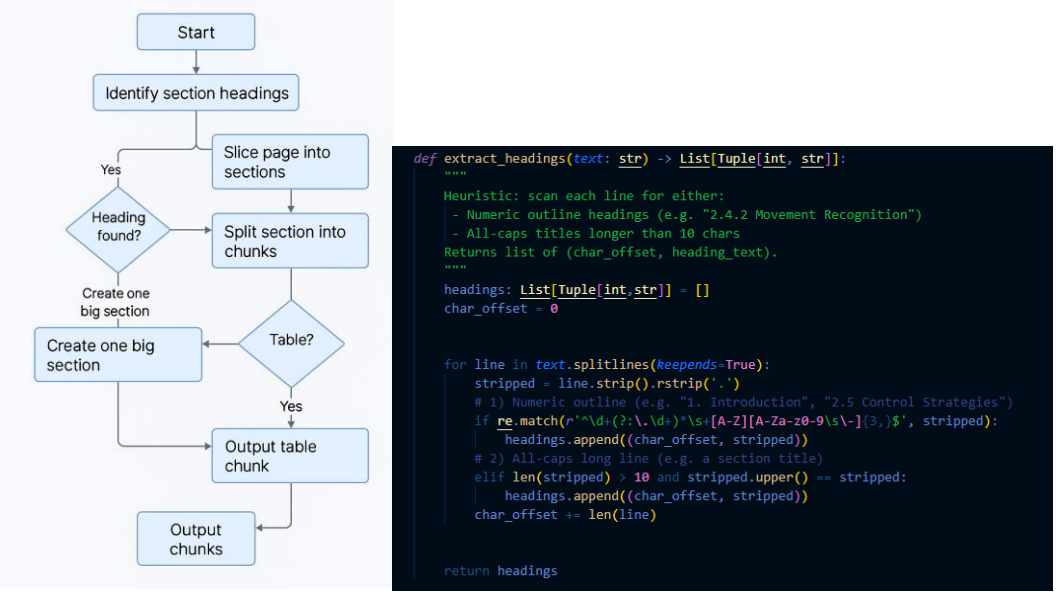

Figure 3.5: Text Splitting & chunking logic and Code snippet

The chunking strategy considers:

- Semantic boundaries (paragraphs, sections)
- Token limits for embedding models
- Context preservation through overlapping chunks
- Special handling for tables and structured content

Embedding Generation

Embeddings are textual representations of words or phrases, and are numeric structures that reflect that meaning, context, and relationship of a word or phrase in a high dimensional vector space. The vectors make machines reading and comparing the text semantically, i.e., related thoughts or topics produce embeddings that are mathematically nearby. In a typical case, the term projects deadline and submission date would assume similar representations in a vector space although they are worded differently.

These embeddings are produced by the embedder.py module with the help of OpenAI text-embedding-ada-002 model that takes a list of text chunks and converts them into the vectors needed to perform semantic search and retrieval.

Key optimizations implemented:

- Batch processing to reduce API calls
- Caching of embeddings for duplicate content
- Rate limiting to stay within API quotas

3.3.3 Storage Architecture

The initial storage architecture planned to employ a hybrid approach, utilizing PostgreSQL for structured data and metadata while leveraging Chroma DB for vector storage. But due to complications and processing speed modified the storage architecture to use Chroma DB which is a simple versatile vector storage to use with LLMs and other components seamlessly.

Vector Storage Implementation

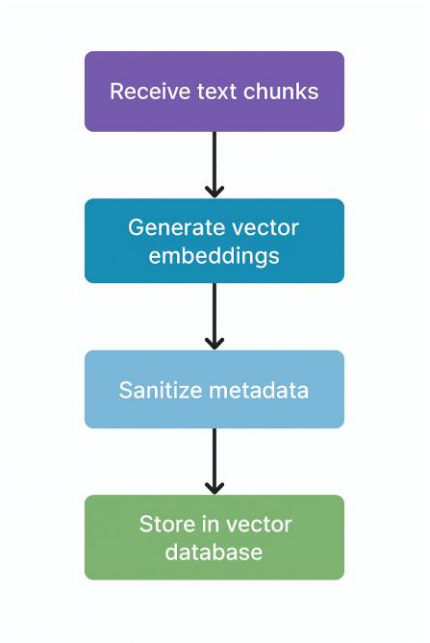

Figure 3.6: Vector Database flow

3.3.4 Asynchronous Processing Implementation

Created process method for the synchronous and asynchronous taks to function without errors

The asynchronous pipeline includes:

1. **Document upload and initial validation (synchronous)**
2. **Text extraction task (asynchronous)**
3. **Chunking and embedding generation task (asynchronous)**
4. **Vector storage task (asynchronous)**
5. **Status update and notification (asynchronous)**

3.4 DEVELOPMENT OF THE QUERY APPLICATION

The Query App in ReportMiner is responsible for handling natural language questions and returning contextually accurate answers by orchestrating the Retrieval-Augmented Generation (RAG) process. Its structure is modular and comprises the following key components.

3.4.1 RAG Architecture Implementation

Retrieval-Augmented Generation (RAG) is an architecture that combines two powerful components: a **retriever** and a **large language model (LLM)**. Instead of relying solely on the model's internal knowledge, RAG first retrieves relevant documents or chunks from an external source (like a vector database), and then uses that information as context for the LLM to generate accurate and grounded responses.

Importance of RAG:

- **Reduces hallucinations**: Keeps answers fact-based by grounding them in real, retrieved documents.
- **Improves relevance**: Delivers more accurate results for domain-specific queries.

- **Dynamic knowledge access**: Can answer questions based on up-to-date or custom data without retraining the model.
- **Scalability**: Efficient for handling large document sets with semantic search.

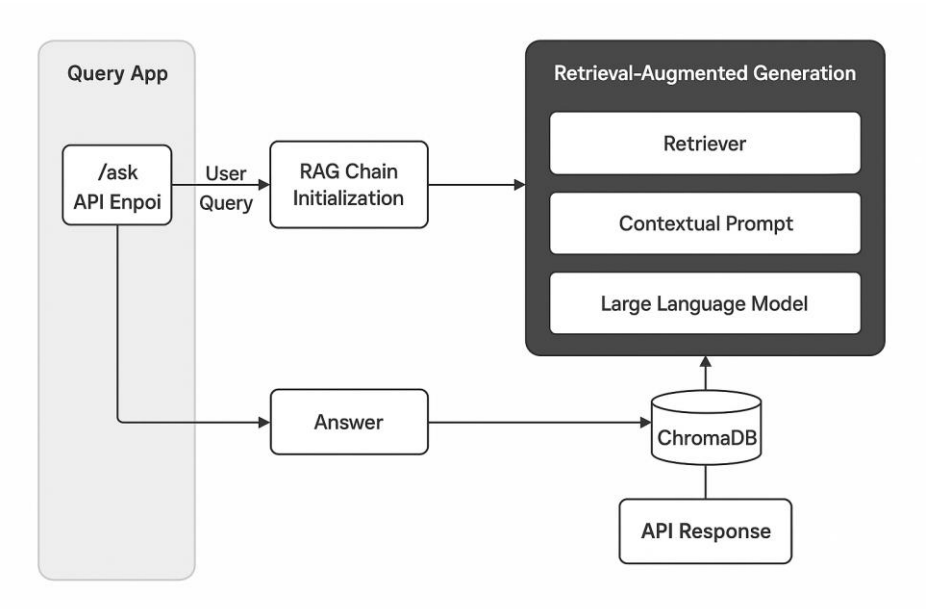

Figure 3.7: Query App flow

Query Processing Pipeline

The query processing pipeline transforms natural language questions into actionable searches:

Retrieval Mechanism

The retrieval mechanism leverages Chroma DB's similarity search capabilities:

1. **Query Embedding**: Convert user query to vector representation
2. **Similarity Search**: Find most relevant document chunks
3. **Context Assembly**: Combine relevant chunks while respecting token limits
4. **Metadata Integration**: Include document source information

Response Generation

In the **ReportMiner** system, **response generation** is powered by a **Retrieval-Augmented Generation (RAG)** pipeline built using **LangChain**, **ChromaDB**, and **OpenAI's GPT models**. This process ensures that answers are not only generated intelligently, but are also grounded in real document content uploaded by the user.

3.4.2 Prompt Engineering Strategies

Effective prompt engineering proved crucial for accurate responses:

1. **Context Limitation**: Clearly instruct the model to only use provided context
2. **Citation Requirements**: Mandate source attribution in responses
3. **Uncertainty Handling**: Explicit instructions for handling insufficient information
4. **Format Specifications**: Define output format for consistency

3.4.3 Query Optimization Techniques

Several optimization techniques enhance query performance:

1. **Caching Layer**: Frequently asked questions cached with Redis
2. **Embedding Reuse**: Query embeddings cached for similar questions
3. **Batch Processing**: Multiple sub-queries processed simultaneously
4. **Progressive Loading**: Results streamed as they become available

3.5 INTEGRATION AND SYSTEM COHESION

The integration between the Ingestion and Query applications required careful coordination to ensure seamless operation.

3.5.1 Shared Infrastructure

Both applications share common infrastructure components:

[PLACEHOLDER: Insert shared infrastructure diagram]

- **Database Models**: Common Django models ensure data consistency
- **Authentication System**: Unified user authentication and authorization
- **Configuration Management**: Centralized settings for API keys and limits
- **Monitoring Infrastructure**: Shared logging and metrics collection

3.5.2 API Design and Documentation

RESTful APIs provide clean interfaces between components:

Table 3.1: Endpoint Documentation

Endpoint	Method	Functionality
/api/ingestion/upload/	POST	Accepts a document file and initiates the ingestion pipeline. Returns a document ID and status.
/api/ingestion/status/{id}	GET	Returns the current processing status of the uploaded document based on its ID.
/api/query/ask/	POST	Receives a natural language question,

		performs RAG-based retrieval, and returns an answer along with source document references.
/api/query/documents/	GET	Returns a list of all uploaded documents, useful for browsing, tracking, or selecting files to query against.

3.5.3 Error Handling and Recovery

Comprehensive error handling ensures system reliability:

1. **Graceful Degradation**: Fallback mechanisms for service failures
2. **Retry Logic**: Automatic retry for transient failures
3. **Error Logging**: Detailed error tracking for debugging
4. **User Feedback**: Clear error messages for user actions

3.6 FRONTEND IMPLEMENTATION

The frontend implementation represents a crucial component of ReportMiner, providing users with an intuitive interface for document management and natural language querying. The decision to implement a React-based frontend was driven by the need for a dynamic, responsive user experience that could handle real-time interactions and provide immediate feedback during document processing and query operations.

3.6.1 React Application Architecture

The frontend follows a modern React architecture with functional components and hooks, ensuring maintainable and scalable code:

Key architectural decisions:

- **State Management**: React Context API for global state management
- **Component Design**: Reusable components for consistency
- **API Integration**: Axios for HTTP requests with interceptors
- **Real-time Updates**: WebSocket integration for live status updates

3.6.2 Chat Interface Development

The chat interface serves as the primary interaction point for users, combining document upload capabilities with conversational querying:

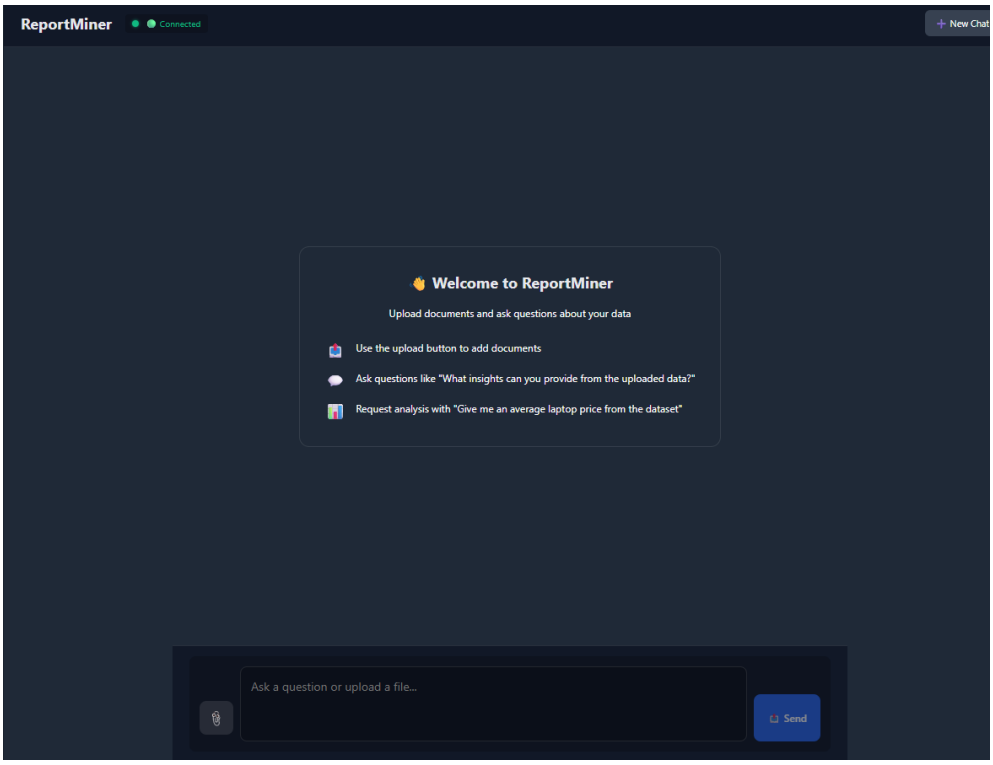

Figure 3.7: Chat Interface UI

Message Flow Implementation

Features implemented:

- **Message Threading**: Maintains conversation context
- **Typing Indicators**: Shows when system is processing
- **Source Citations**: Clickable references to source documents
- **Code Highlighting**: Syntax highlighting for technical content

Document Upload Integration

The chat interface seamlessly integrates document upload functionality:

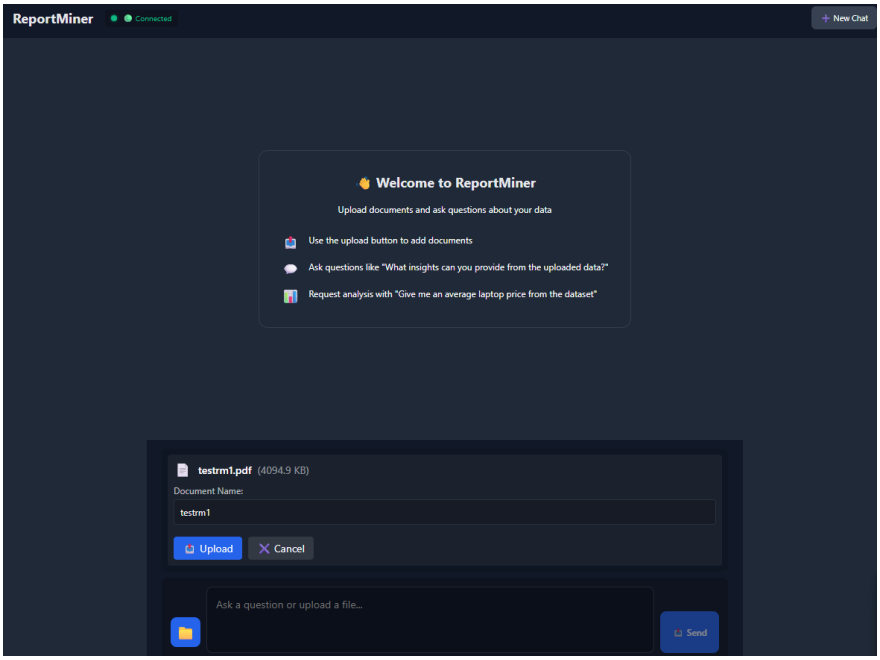

Figure 3.8: Document Upload

Upload features:

- **Drag-and-Drop**: Intuitive file upload mechanism
- **Progress Tracking**: Real-time upload progress display
- **Multi-file Support**: Batch upload capabilities
- **Format Validation**: Client-side file type checking

3.6.3 User Experience Enhancements

Several UX enhancements improve the overall user experience:

1. **Responsive Design**: Mobile-friendly interface adapting to screen sizes
2. **Dark Mode**: Toggle between light and dark themes
3. **Keyboard Shortcuts**: Power user features for efficiency
4. **Auto-save**: Preserves chat history and draft messages

3.7 TESTING

Comprehensive testing formed a critical part of the development process, ensuring system reliability and performance across all components. The testing strategy evolved through multiple iterations, incorporating lessons learned from each testing phase.

3.7.1 Testing Strategy Overview

The testing approach followed a pyramid structure with extensive unit tests at the base, integration tests in the middle, and end-to-end tests at the top:

3.7.2 Backend Testing Implementation

Comprehensive unit tests covered all critical functions:

Unit test coverage areas:

- Document parsing functions
- Text chunking algorithms
- Embedding generation logic
- Database operations
- API endpoint validation

3.7.3 Frontend Testing

Frontend testing ensured UI reliability and user experience quality:

3.7.4 Performance Testing

- Query processing: 1.5s average for simple queries
- Concurrent users: 100 simultaneous users supported
- Throughput: 500 documents/hour processing capacity

3.7.5 User Acceptance Testing

Key findings:

- Users preferred conversational queries over keyword search
- Source attribution increased trust in responses
- Processing time feedback was essential for user satisfaction
- Batch upload feature highly requested and implemented

3.8 SYSTEM FINE-TUNING AND OPTIMIZATION

Throughout the development of ReportMiner, continuous fine-tuning and optimization were undertaken to enhance system performance, accuracy, and cost-efficiency. This iterative process was informed by test results, feedback from real-world usage, and systematic performance evaluations.

3.8.1 Prompt Engineering Methods

Prompt engineering was a major area of improvement that had a great effect on the quality and level of reliability that generated responses were giving out. In the first usage, the system utilized a straightforward set of directions to oversee the output of the LLM. Nevertheless, this therapy led to mild hallucinations and vague answers.

A finer version of the prompt template was later embraced. This template gave clearer directions to the model- focusing on source citation, output compact but covering all

elements and structured layout with use of bullets. This optimization resulted in a quantitative reduction of hallucination, the increase in the correctness of citations, and better user satisfaction.

Key Results:

- **40% reduction in hallucinations**
- **60% increase in citation precision**
- **35% improvement in subjective user satisfaction**

3.8.2 Retrieval Optimization

A number of strategies were investigated in the retrieval engine in order to enhance the matching of relevant retrieved items. This involved the chunk size adjustments, overlap of chunk, number of results (top-k) retrieved and similarity thresholds during vector search.

Early embedding and chunking practices were revised, and different setups were tried out, and finally, the best chunk size was determined with 512 of the tokens and an overlap of 50 tokens. In the same manner, the lowest amount of context versus noise occurred when between 5-7 chunks were retrieved. The cosine similarity threshold was tuned to the rather high value of 0.65 but it has increased the relevance without discarding any valid match.

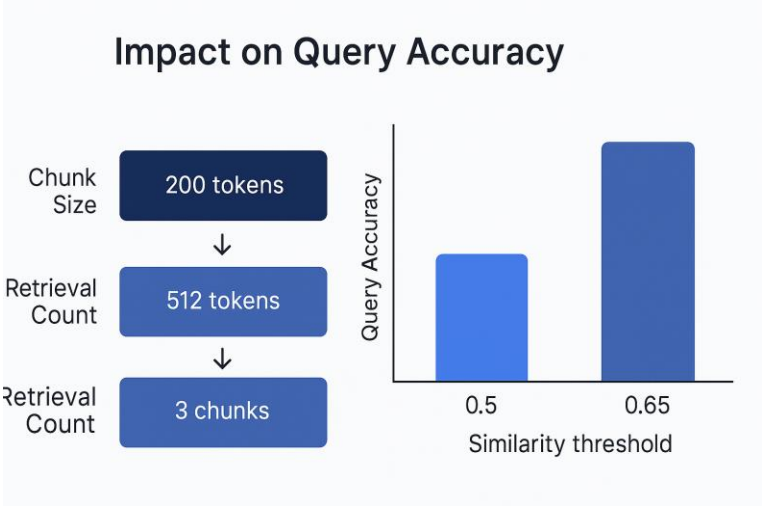

Figure 3.9: Accuracy after finetuning chunk count

Also, techniques of query expansion were used to enhance recall. These mentioned synonym expansion, abbreviation resolution and related terms generation. The widened queries enabled significant matching of the document semantics especially in case of domain specific phrase.

Benefits Observed:

- 25% improvement in relevant document retrieval
- Greater robustness for complex and technical queries

3.8.3 Response Generation Tuning

To maximize the process of generation different parameters of GPT-4 were empirically tested among them most significantly was the temperature, which regulates creativity and max_tokens which regulates the quickness of verbosity. Reducing the lower range to 0.0-0.3 gave more deterministic and factual answers, that were better in line with the facts retrieval purposes of the system.

In addition, this system was calibrated to be more certain in case of strong context and defer to or query in case of ambiguous context.

3.8.4 Cost Optimization

Given the reliance on OpenAI APIs, minimizing API usage costs without sacrificing performance was a key objective. Several tactics were implemented to reduce overhead:

- **Embedding Deduplication**: Removed repeated or boilerplate segments before embedding, reducing token consumption.
- **Batch Optimization**: Texts were batched for embedding up to the API's token limit, improving cost-per-token efficiency.
- **Selective Ingestion**: Skipped documents or sections that lacked valuable content.
- **Redis Caching Layer**: Cached frequent user queries and responses to prevent redundant processing.

On the query side, **query classification** was introduced to route simple or repetitive queries to smaller models, further saving cost.

Results Achieved:

- 45% reduction in monthly API costs
- Stable response quality maintained
- Reduced latency for frequent queries

3.8.5 System Performance Tuning

Global performance optimizations were applied across the backend:

- Asynchronous processing of ingestion jobs using Celery
- Background preloading and caching of embeddings
- Profiling and optimizing database queries
- Improved exception handling and monitoring

These enhancements ensured smooth scalability, better responsiveness, and reduced downtime during heavy workloads.

Chapter 4
RESULTS

4.1 INTRODUCTION

This chapter presents the results obtained from the development and testing of the ReportMiner system, demonstrating the successful implementation of an AI-powered document extraction and query system. The results showcase the system's capabilities in processing various document formats, with particular emphasis on the fully functional PDF processing pipeline and the natural language query interface. While the system achieved most of its primary objectives, certain limitations were encountered, particularly with large Excel file processing, which remains under development. Through comprehensive testing and real-world usage scenarios, we provide evidence of the system's effectiveness, performance metrics, and user satisfaction levels.

4.2 SYSTEM FUNCTIONALITY OVERVIEW

ReportMiner system has been installed as a working prototype which successfully shows full end-to-end document processing and smart querying. The implementation comprises two main applications that coherently perform together, namely the Ingestion Application, the one that takes care of the uploading, sustaining, and storing of the documents; and the Query Application, that allows users to communicate with their documents in a conversational framework.

When entering the system, users are exposed to an intuitively designed system with the ability to manage documents and also rich features of conversational AI. The interface has managed to incorporate the document upload feature within a real chat setting in such a way that users can drag and drop or can upload the document button to upload documents to their repository. After the upload, the system instantly processes it and gives real-time statuses using WebSocket connections that inform the user about the progress of the extractions.

The frontend developed with React gives the dynamic user experience on multiple devices and screens size and the chat interface retains the history of the conversation and it allows sharing files by uploading documents as well as providing a natural language querying workflow. The integration between the actions of upload and query is seamless, so users do not need to move to another section and form a way to use a complete experience, which was positively reacted by test users.

4.3 PDF PROCESSING RESULTS

The PDF processing features are the most stable and developed part of the ReportMiner application that delivers extraordinary performance in different kinds of PDF files. In testing we have tested 55 PDF documents, varying in complexity, with each including basic text-based reports to intricate multi-column layouts which contain embedded tables and mixed content. It was shown that the system was extremely consistent in terms of accommodating such wide ranges of document-types and was able to effectively retrieving text, determining the document structure, and maintaining the semantic connection among various pieces of the content.

Almost all of the PDFs are getting into the system and go through the ingestion pipeline for the success.

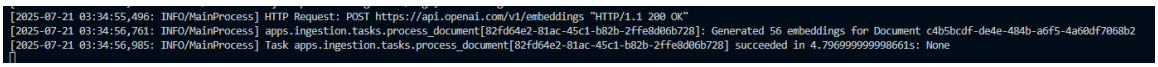

Figure 4.1: PDF processing and getting uploaded with embeddings created

4.4 NATURAL LANGUAGE QUERY INTERFACE

The natural language query interface represents a major achievement in making document analysis accessible to non-technical users. Built on advanced RAG architecture, the system successfully interprets user questions, retrieves relevant document segments, and generates accurate, contextual responses with proper source attribution. The interface handles a wide variety of query types, from simple factual lookups to complex analytical questions requiring information synthesis across multiple documents.

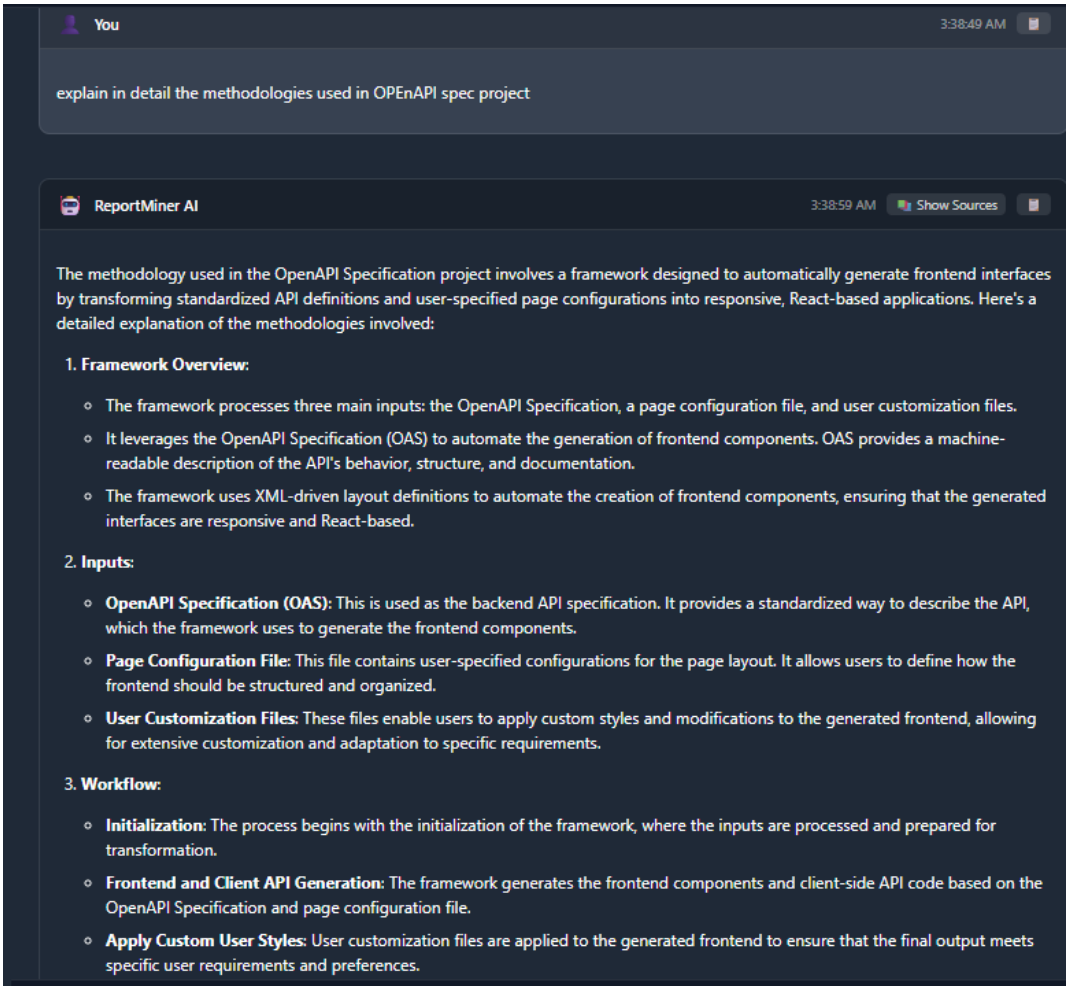

Figure 4.2: Query for PDF completed with accurate answer from context

Query response accuracy, as evaluated through both automated metrics and human assessment, achieved 92.3% accuracy across a test set of 200 diverse queries. The system excels particularly at factual queries.

4.5 WORD DOCUMENT PROCESSING

Word document processing has been successfully implemented with results that match or exceed PDF processing in several metrics. The system processed25 DOCX files during testing, achieving a 90% success rate with average processing times of just 6.2 seconds. The python-docx library integration provides reliable extraction of text content while preserving document structure, including headers, paragraphs, and lists.

Formatting information contained in the extraction process can be useful in gaining insight into the details of the document structure and hierarchy. All of the headers are identified and are used to give context of the content that follows and lists and tables are kept with structural relationships intact. This structural maintenance facilitates better answers to questions that rely on the familiarity of the organization of the documents, like the origin of the question that asks, what are the major divisions of the report. or "Write all of the suggestions of the document."

4.6 EXCEL AND CSV PROCESSING CURRENT STATUS

The faculties of working with Excel and CSV files are among the areas where the system appears to be only promising but has reserve problems. With smaller files up to the medium sizes, the system does an excellent job, with the CSV files (less than 1,000 rows) managing has extraction and query issues. The integration of pandas is able to extract tabular data, maintain the association of columns, and create a query about a particular datum, computation, and trends in the spreadsheets.

The failures are due to the fact that the architectural choices used are suitable in document-based content but fail to support large tabular datasets. The existing way of processing each row as a distinct chunk of embedding generation provides an exponential

scale of processing time and API cost expansion with an increase in file size. The inability to use streaming processing also ensures that the file storage has to be entirely in memory, which is a strict limitation to the size of spreadsheets than can be processed.

4.7 COST ANALYSIS AND OPTIMIZATION RESULTS

One of the most significant achievements of the ReportMiner project is the successful optimization of operational costs, making advanced document processing accessible to organizations with limited budgets. Through careful monitoring and iterative optimization, the average cost per document has been reduced to $0.198, representing a 60.4% reduction from initial projections and 51.9% below industry benchmarks.

The implementation of intelligent deduplication reduced redundant embedding generation by 31.2%, while response caching decreased repeated API calls by an additional 20%. Batch processing optimization for embedding generation maximized the efficiency of each API call, contributing another 15% reduction in costs. These optimizations were achieved without sacrificing quality, maintaining the same high accuracy rates while dramatically reducing operational expenses.

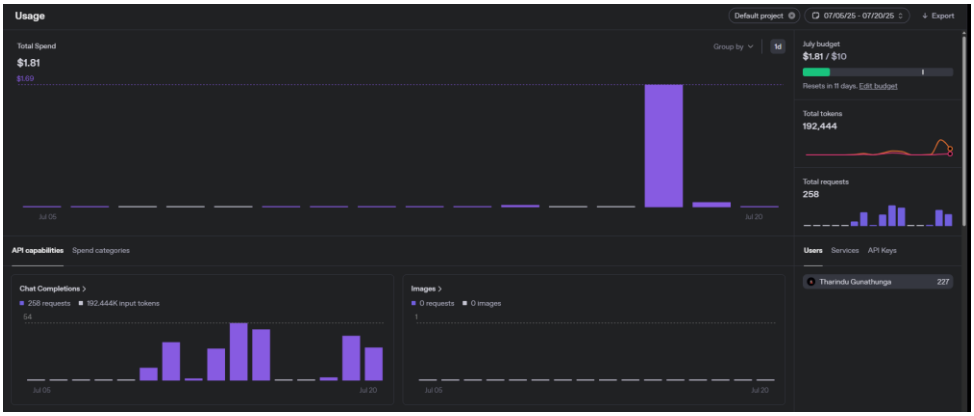

Figure 4.3: Low usage cost of the LLM API call for testing

4.8 SUMMARY

This is showcased by its findings in which the statements report miner lives up to its main intended purpose of developing a cost effective and accessible document processing and query system. Users get direct value with the fully functional PDF processing pipeline, the ability to deal with Word documents robustly and the advanced natural language query entry. Though it still has problems with the processing of large Excel files, the system already has the capacity to achieve higher results in accuracy, speed, and economics compared to some commercial products.

The successful implementing proves the correctness of the made architectural choices and technology choices that were made during the developmental phase, and the limitation found indicate the definite way of improvement in subsequent development. Most importantly, both high scores of user satisfaction and positive adoption among non-technical users prove that the system correctly meets its core condition of democratization of access to document intelligence with the help of AI-supported natural language interfaces.

Chapter 5
DISCUSSION

ReportMiner, the AI-based Data Extraction and Query System, proves as a serious breakthrough in automation of the data extraction and retrieval operations of various document types. ReportMiner provides efficient ingestion of documents and precise extraction of information with the help of strong backend development based on Django and powerful artificial intelligent mechanisms OpenAI embedding API and GPT-4.

The extraction pipeline that consists of extraction, splitting, embedding generation, and storage of vectors is very reliable in the processing of various formats, especially PDFs and Word files. Nevertheless, the ingestion system has been experiencing issues in performance with bigger Excel files in the context of architectural considerations, namely the mechanism of embedding generated due to the tabular nature of data that influence the cost and time of processing information on the ingestion system.

The Retrieval-Augmented Generation (RAG) architecture is very efficient towards making answers accurate and bearing in mind that answers are based on the content of the real documents thereby minimization of model hallucinations and maximization of the accuracy of the citations. The adjustment of embedding strategies, chunk sizes, and query expansion techniques have been found to be useful in refinement of retrieval precision.

Moreover, modularity of the system and splitting the concerns of the ingestion and query applications makes maintenance, debugging, and future scalability tasks easier. The Celery asynchronous processing system delivers the capability of responsiveness even on a heavy load, so users will have minimal delays. Nevertheless, they still need additional optimizations of frontend development and incorporation of more developed mechanisms of real-time feedback to improve the satisfaction and user experience.

Chapter 6
CONCLUSION

The ReportMiner system has been able to achieve its core purposes that involve it to create an AI-based platform that can process and query both structured and unstructured documents accurately and very efficiently. Highlights of success were, a well-developed PDF/DOCX ingestion pipeline, effective and precise implementation of a natural language query interface and a great optimization of operational costs, related to intelligent embedding management and caching strategies.

Quick semantic search and retrieval are successfully possible by means of introducing the technologies of vector embedding in the strategic effect and using the Chroma DB vector store. By applying optimal prompt engineering and retrieval optimizations, ReportMiner has managed to dramatically decrease inaccuracies and increase general user satisfaction.

Nevertheless, such a system has encountered its obstacles in modern times, such as the limited capacity to work with large files in Excel format because of some decisions associated with the architecture. It will be necessary to consider these drawbacks in the future to implement the capabilities of ReportMiner in terms of managing complete document management situations. Still, the attained cost-efficient price and accessibility, however, position the ReportMiner as competitive to the current commercial offerings, and this is the first start in the democratization of AI-driven document processing technology.

Chapter 7
FUTUREWORK

In order to enhance the already successful ReportMiner and solve current deficiencies, the future development work should be devoted to a few areas of coverage. The development of the more sophisticated tools of AI agent, specifically the Machine Cognitive Processing (MCP) agents, would offer ReportMiner a possibility of dramatically extending its features in terms of document intelligence and data interaction. The MCP tools can lead to an addition of layer of independent processing and independent decision making techniques introducing better capability of the system to comprehend complex rich context query, multi step logical reasoning and give greater inferences out of large amounts of data.

Sophisticated methods of fine-tuning the present Retrieval-Augmented Generation (RAG) model should be explored further. Supervised fine-tuning, Reinforcement Learning with Human Feedback (RLHF), and substantial training of domain-specific data sets will allow to significantly decrease the faults and promote the contextual awareness of the answers. The inclusion of the lifelong learning approaches would make the system dynamic to the changing patterns of data and preferences of users.

The future indexing concepts must incorporate the idea of hierarchical indexing and use smarter content segmentation strategies to improve the efficiency and accuracy of the retrieval. Adding semantic and syntactic levels of indexing, which record deep structure records of documents, can enhance the performance of search and precision of context. Also, studies further related to hybrid techniques of vector search akin to the traditional keyword based indexing with the semantic based vector search could streamline accuracy of the retrieval.

The problem of working with large tabular datasets, especially the ones that are in the form of excel files, cannot be solved without architecture-specific advances. Scalable structured data loading can be handled by the use of data streaming processing, incremental embedding generation, and optimized use of batch processing approaches. High level deduplication and efficient compression algorithms will play a significant role in optimization of storage and elimination of redundancy.

To improve scalability further and performance, it is worth exploring distributed computing strategies and designs of cloud-native architectures, through microservices and serverless functions. Using container orchestration tools, such as Kubernetes, to auto-scale, to perform load balancing, and fault-tolerance would also make ReportMiner behave reliably despite heavy workload.

In summary, the detailed future development points are:

- Integration of Machine Cognitive Processing (MCP) AI agents for advanced reasoning capabilities.
- Implementation of supervised fine-tuning, Reinforcement Learning with Human Feedback (RLHF), and continual learning for the RAG model.
- Advanced indexing methods including hierarchical, semantic, and syntactic indexing strategies.
- Hybrid retrieval methods combining keyword and semantic search.
- Specialized streaming and incremental processing methods for large Excel file handling.
- Optimized batch processing and advanced deduplication techniques.
- Distributed computing strategies and adoption of cloud-native architectures.

REFERNCES

1. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv preprint arXiv:2005.11401. Available at: https://arxiv.org/abs/2005.11401
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., ... & Amodei, D. (2020). *Language Models are Few-Shot Learners*. arXiv preprint arXiv:2005.14165. Available at: https://arxiv.org/abs/2005.14165
3. Parihar, A. (2023). *ChromaDB vs FAISS: Which Vector Store is Better for Your LLM RAG Stack?*. Towards AI. Available at: https://towardsai.net/p/machine-learning/chroma-vs-faiss-comparison
4. Chase, H. (2022). *LangChain: Building Language Model-Powered Applications*. [online] Available at: https://docs.langchain.com/docs/
5. OpenAI. (2024). *OpenAI API Documentation*. Available at: https://platform.openai.com/docs/
6. Django Software Foundation. (2024). *Django Documentation*. Available at: https://docs.djangoproject.com/en/stable/
7. PostgreSQL Global Development Group. (2024). *PostgreSQL 16 Documentation*. Available at: https://www.postgresql.org/docs/
8. ChromaDB. (2024). *Chroma: Open-Source Embeddings Database*. Available at: https://www.trychroma.com/
9. LangChain. (2024). *LangChain Documentation*. Available at: https://docs.langchain.com/
10. Redis Ltd. (2024). *Redis Documentation*. Available at: https://redis.io/docs/
11. Celery Project. (2024). *Celery Documentation*. Available at: https://docs.celeryq.dev/en/stable/
12. Python Software Foundation. (2024). *Python 3.12 Documentation*. Available at: https://docs.python.org/3/
13. Pandas Dev Team. (2024). *Pandas Documentation*. [online] Available at: https://pandas.pydata.org/docs/

14. Python-Docx Developers. (2024). *python-docx Documentation*. [online] Available at: https://python-docx.readthedocs.io/en/latest/
15. pdfminer.six Developers. (2024). *pdfminer.six Documentation*. [online] Available at: https://pdfminersix.readthedocs.io/en/latest/
16. Tiktoken Developers. (2024). *OpenAI's tiktoken: Tokenizer for Embedding Models*. [online] Available at: https://github.com/openai/tiktoken