SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

MULTIMEDIA ANALYZER

INT 400 – INTERNSHIP III PROJECT REPORT

Submitted by

MEHA MURALI – E0122046

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Artificial Intelligence and Machine Learning)

Sri Ramachandra Faculty of Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai - 600116

OCTOBER, 2024

MULTIMEDIA ANALYZER

INT 400 – INTERNSHIP III PROJECT REPORT

Submitted by

MEHA MURALI – E0122046

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Artificial Intelligence and Machine Learning)

Sri Ramachandra Faculty of Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai – 600116

OCTOBER, 2024

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that this project report **"MULTIMEDIA ANALYZER"** is the bonafide record of work done by **"MEHA MURALI – E0122046"** who carried out the internship work under my supervision.

Signature of the Supervisor

Prof. A. Aruna

Assistant Professor,

Department of Artificial Intelligence and Machine Learning

Sri Ramachandra Faculty of Engineering and Technology,

SRIHER, Porur, Chennai-600 116.

Signature of Programme Coordinator

Dr. V. Vanitha

Associate Professor,

Department of Artificial Intelligence and Machine Learning

Sri Ramachandra Faculty of Engineering and Technology,

SRIHER, Porur, Chennai-600 116.

Evaluation Date:

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

ACKNOWLEDGEMENT

I take this opportunity to express my gratitude to the Dean Dr.T.Ragunathan for providing all the facilities to complete this Internship work successfully.

I take this opportunity to thank my Department Coordinator Dr.V.Vanitha for her encouragement and support.

I wish to thank my faculty supervisor, **Prof. A. Aruna,** Department of Artificial Intelligence and Machine Learning, Sri Ramachandra faculty of Engineering and Technology and **Srini Ramanujam, Chief Architect, Optiwise** for extending help and encouragement throughout the project. Without his continuous guidance and persistent help, this project would not have been a success for me.

I extend my thanks to the Internship Coordinator Dr.S.Rajalakshmi for her valuable support and Guidance in the smooth conduction of reviews.

I am grateful to all the members of Sri Ramachandra Faculty of Engineering and Technology, my beloved parents and friends for extending the support, who helped us to overcome obstacles in the study.

TABLE OF CONTENTS

ABSTRACT

This project presents an AI-driven document retrieval system utilizing **LlamaIndex** for semantic understanding and **Pinecone** for efficient vector storage. The system is integrated with a **Flask** UI, enabling users to submit natural language queries. LlamaIndex converts queries and documents into vector representations, which are stored in Pinecone for fast, context-based retrieval. The system ensures accurate, relevant search results with sub-3-second response times, even for large datasets. It has potential applications in legal, academic, and corporate document search, with future enhancements focusing on domain-specific fine-tuning and multimedia support.

CHAPTER 1
INTRODUCTION

1.1. OVERVIEW OF DOCUMENT QUERYING SYSTEMS

In today's digital era, the amount of information created and stored is increasing exponentially. The ability to quickly and accurately retrieve relevant documents from vast data repositories is critical in domains such as law, healthcare, research, and education. Traditional keyword-based systems, while fast and efficient, often struggle with understanding the intent behind a user’s query, leading to irrelevant results. This problem becomes even more challenging with unstructured data like research papers, legal documents, and web articles.

Document query systems powered by machine learning models have emerged as a solution to this challenge. By leveraging natural language understanding (NLU) and advanced neural networks, these systems can deliver contextually relevant documents, even when faced with ambiguous or complex queries.

This project, titled "Document Query System using LlamaIndex and Pinecone," proposes the development of a cutting-edge querying system that integrates LlamaIndex for generating document embeddings and Pinecone for storing and searching these embeddings. The system is designed to provide high-speed and high-accuracy document retrieval with a seamless user experience through a web-based interface built using Flask.

1.2. PROBLEM STATEMENT

Current document querying methods often rely on simple keyword matching, which lacks the sophistication required for context-based searches. Traditional systems face several issues:

- Inability to capture the meaning behind complex queries.
- Limited speed and scalability in handling large document repositories.
- Inconsistent accuracy when retrieving relevant information.

To solve these issues, this project integrates advanced language models and vector search technologies to improve both the speed and accuracy of document retrieval. The solution uses LlamaIndex for generating high-quality document embeddings and Pinecone for fast and scalable vector-based search. Together, these tools allow for a more intelligent and efficient querying process, particularly for large-scale datasets.

1.3. AIM AND OBJECTIVE

The aim of this project is to build an efficient and accurate document query system that improves the relevance of search results using state-of-the-art machine learning technologies.

The specific objectives are:

- To develop a system capable of understanding the semantic meaning behind user queries.
- To create a document embedding system using LlamaIndex that transforms text into a vector representation.
- To implement a scalable vector search system using Pinecone that stores and retrieves document embeddings.
- To build an intuitive user interface using Flask for seamless interaction with the system.

1.4 SCOPE OF PROJECT

This document query system can be applied to multiple domains:

- **Legal Systems**: Legal firms can use it to search and retrieve case laws, contracts, and legal documents.
- **Healthcare**: Hospitals and medical professionals can efficiently retrieve patient records or medical research papers.
- **Educational Institutes**: Researchers and educators can search through academic papers, journal articles, and dissertations.
- **Business Organizations**: Corporations can manage and retrieve documents from internal knowledge bases, contracts, and compliance records.

The system’s ability to handle large datasets makes it a powerful tool in any domain that requires quick and accurate information retrieval.

1.5 SUMMARY OF APPROACH

The overall approach for the system follows a structured workflow:

1. **Document Embedding**: Each document is converted into a dense vector representation using LlamaIndex.
2. **Data Storage**: These embeddings are stored in Pinecone, which provides fast vector search capabilities.
3. **Query Processing**: The user's query is transformed into an embedding and compared to document embeddings stored in Pinecone.
4. **Results Display**: The most relevant documents are displayed through an easy-to-use Flask interface.

CHAPTER 2

LITERATURE REVIEW

2.1 Evolution of Document Retrieval Systems

Document retrieval has evolved significantly since the early days of information retrieval systems in the mid-20th century. The initial systems employed basic methods such as Boolean search and exact keyword matching, which, while effective for small datasets, often struggled with larger datasets. These traditional approaches failed to account for the complexity of human language, returning irrelevant or incomplete results due to their reliance on exact keyword matching without understanding the broader context or meaning of the query.

In the 1990s, the introduction of vector-space models such as TF-IDF (Term Frequency-Inverse Document Frequency) and cosine similarity marked a substantial improvement. These models ranked documents by their relevance to the query, based on the frequency of keywords and the relationships between them. Despite this advancement, they still fell short in comprehending the true meaning behind words and phrases, focusing primarily on word occurrence rather than deeper semantic relationships.

The advent of machine learning and deep learning models like BERT (Bidirectional Encoder Representations from Transformers), GPT-3, and Llama revolutionized the field of document retrieval. These models introduced the concept of embeddings, which are dense vector representations of words, phrases, and entire documents. Unlike earlier systems, embeddings capture the semantic essence of the text, enabling document retrieval systems to understand the meaning behind words and phrases. This development has led to more accurate and context-aware retrieval, effectively addressing many of the limitations found in traditional keyword-based search methods.

2.2 Language Models in Document Retrieval

Language models, especially those based on transformer architecture, have emerged as the foundation for numerous natural language processing (NLP) tasks, including document retrieval. BERT and GPT models, in particular, excel at understanding word relationships and context within a sentence. This capability allows them to produce more nuanced and accurate document embeddings, which can be compared to other embeddings to find the

most relevant results. Unlike earlier keyword-matching techniques, these models are capable of grasping the underlying meaning and intent behind a user's query.

In this project, LlamaIndex plays a central role in generating document embeddings. It enhances the document retrieval process by capturing not only the meaning of individual words but also the relationships between them at a higher level, spanning entire sentences and documents. This allows for a deeper understanding of user queries and ensures that the system retrieves documents that are highly relevant in both meaning and context. By leveraging advanced language models like LlamaIndex, the document query system can offer more accurate and sophisticated results that go beyond surface-level keyword matches.

2.3 Vector-Based Search Methods

Vector-based search methods have transformed the way document retrieval systems function by embedding both documents and queries into high-dimensional vectors. These vectors represent the semantic content of the text, allowing for a comparison based on proximity or similarity in the vector space. The closer two vectors are, the more semantically similar their corresponding texts are. This method has been instrumental in advancing the accuracy of document retrieval, as it moves beyond simple keyword matching and instead understands the contextual and semantic meaning of queries.

In this project, Pinecone is used as the vector database responsible for storing and retrieving these document embeddings. Pinecone's distributed, high-speed architecture allows for real-time search and retrieval of vectors, making it highly scalable for handling large datasets. By utilizing Pinecone, the system can efficiently process complex queries, providing rapid and relevant results even as the volume of stored documents grows. This scalability, combined with the precision of vector-based search methods, makes Pinecone an optimal choice for building modern, high-performance document retrieval systems.

Through the combined use of vector-based search techniques and advanced language models like LlamaIndex, this project ensures a robust, scalable, and contextually aware document query system capable of delivering highly relevant results in real-time.

CHAPTER 3

PROPOSED METHODOLOGY

3.1 System Overview

The proposed document query system operates as a three-tier architecture, comprising a frontend user interface, a backend processing system, and a vector database for document storage and search. This architecture ensures efficient interaction between the user and the backend, enabling rapid and relevant document retrieval.

The system leverages LlamaIndex to generate dense vector embeddings of documents, capturing their semantic essence. These embeddings are stored and indexed in Pinecone, a high-performance vector database optimized for large-scale, real-time searches. When a user inputs a query, it is also transformed into a vector embedding by LlamaIndex and then compared against the stored embeddings in Pinecone to identify the most relevant documents.

3.2 Data Preprocessing

Before documents are embedded for retrieval, they undergo a crucial preprocessing step to enhance the quality and relevance of the generated embeddings. This step ensures that only meaningful content contributes to the document embeddings, improving search accuracy.

Key components of this process include:

Text Cleaning: This step involves removing irrelevant content such as special characters, numbers, symbols, and excessive whitespace, as well as handling common formatting errors. Additionally, stopwords—common words that provide little value in information retrieval—are removed to avoid unnecessary noise in the data.

Tokenization: The cleaned text is then tokenized, meaning it is split into smaller units or "tokens" (typically words or phrases). Tokenization is essential for feeding the text into the Llama model, as it breaks down the document into manageable units that can be processed to generate embeddings.

This preprocessing stage is vital because it refines the input text, ensuring that embeddings are representative of the document's core content and context.

Key Steps in Data Preprocessing:

Text Cleaning: Removal of special characters, symbols, stopwords, and whitespace.

Tokenization: Splitting of text into smaller, manageable units for embedding.

3.3 Document Embedding with LlamaIndex

Once documents have been preprocessed, LlamaIndex is employed to generate embeddings for each document. Embeddings are high-dimensional vectors that capture the semantic meaning of the text, allowing for contextually aware comparisons between documents. These vectors act as the foundational representation of documents and user queries in the system.

LlamaIndex leverages deep learning techniques to create embeddings that understand not just individual words but also their relationships within sentences and paragraphs. This ability to encode context enables more accurate document retrieval, as it allows the system to understand the meaning behind both documents and queries.

After embedding, these vectors are stored in Pinecone for efficient retrieval. When a query is submitted, it is similarly converted into a vector by LlamaIndex and matched against the stored document embeddings in Pinecone to retrieve the most relevant documents based on semantic proximity.

Code Snippet for Embedding Generation:

```
from llama_index import LLamaIndex
docs = load_documents('path_to_documents')
index = LLamaIndex(docs)
doc_embeddings = index.get_embeddings()
```

The embeddings generated are stored in Pinecone for later retrieval.

3.4 Pinecone for Vector Search

Pinecone serves as the core vector database for storing and searching document embeddings. Pinecone is designed for real-time vector search at scale, capable of handling millions of vectors while providing low-latency searches. This makes it a powerful tool for the document query system, where performance and scalability are key.

When a user submits a query, LlamaIndex converts the query into a vector, which is then used to search Pinecone for similar document embeddings. The similarity between vectors is determined by their proximity in the high-dimensional vector space, where closer vectors indicate higher semantic similarity.

Pinecone's ability to handle large datasets and perform fast searches allows the system to retrieve relevant results almost instantaneously, even as the number of stored documents grows.

Code Snippet for Storing and Searching Vectors in Pinecone:

```
import pinecone
pinecone.init(api_key='your_pinecone_api_key')
index = pinecone.Index('document-query-system')
index.upsert(vectors=doc_embeddings)

    # Querying the system
    query_vector = llama_index.get_embedding('your_query')
    results = index.query(query_vector)
```

3.5 Flask Interface for User Interaction

The user interface is built using the Flask web framework, providing a simple yet effective platform for users to interact with the document query system. The interface allows users to input their queries in natural language, which are then processed by the backend to retrieve the most relevant documents.

The Flask framework handles communication between the user and the backend, ensuring that queries are submitted, processed, and results are returned in real time. Flask's

lightweight architecture makes it ideal for this application, offering the flexibility to scale as the system grows.

Key features of the user interface include:

Query Submission: Users can submit queries using a straightforward input form. The system processes these queries in real-time and retrieves the most relevant documents based on vector similarity.

Real-Time Feedback: Thanks to the efficiency of Pinecone's vector search capabilities, users receive almost immediate feedback, with relevant documents displayed quickly after submission.

User-Friendly Design: The interface is designed with simplicity in mind, offering clear instructions and easily interpretable results.

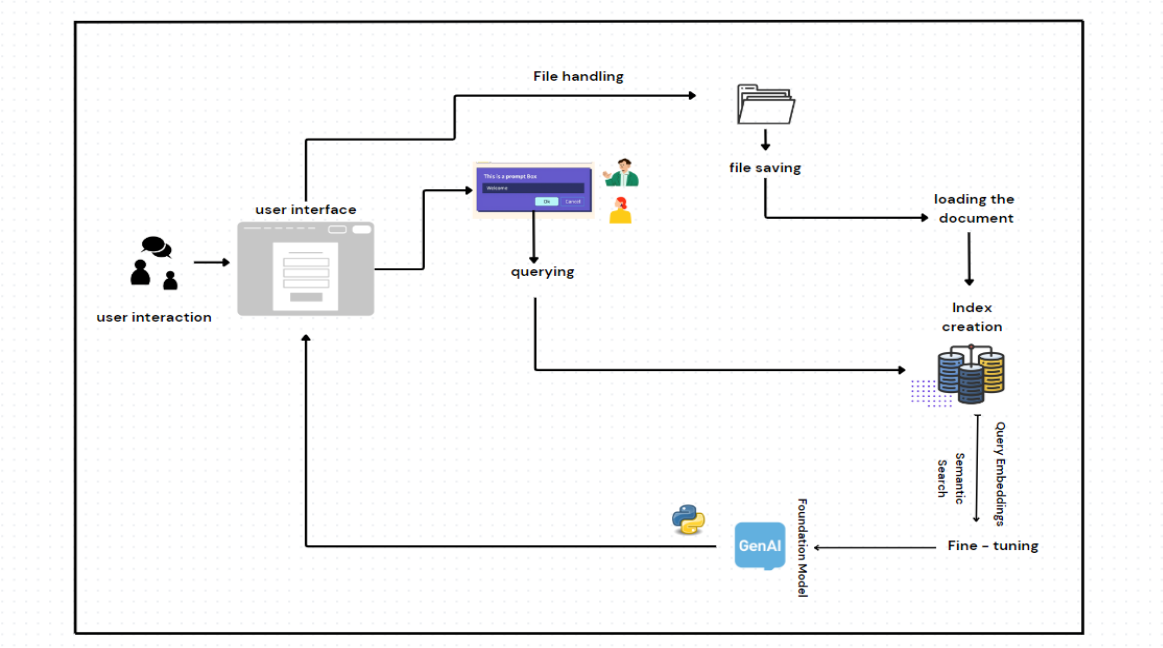
Code Snippet for Flask App:

```
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/query', methods=['POST'])
def query():
    user_query = request.form['query']
    query_embedding = llama_index.get_embedding(user_query)
    results = pinecone_index.query(query_embedding)
    return jsonify(results)
```

CHAPTER 4
SYSTEM ARCHITECTURE

This diagram appears to illustrate the architecture of a GenAI system focused on document querying and handling, possibly for the website you're building that allows users to read and query PDFs. Here's a breakdown of the process:



User Interaction:

Users interact with the system through a user interface. This interface likely provides a form or chat interface to input queries.

User Interface:

This UI processes the user's inputs (queries) and interacts with various system components. It likely presents the response back to the user after the query is processed.

File Handling:

The system handles files through a file handling mechanism, where documents are saved. Users or the system may upload and store documents in a folder or a database.

Loading the Document:

Once the document is saved, the system loads it for further processing. This step involves reading the document into memory so that it can be queried.

Index Creation:

After loading, an index is created from the document. This step prepares the document for efficient search and querying, using techniques like tokenization and embedding.

Query Embeddings & Semantic Search:

For more accurate results, the system transforms user queries into embeddings (vectorized representations of text) and performs semantic search. This is where the query and the document are matched based on the meaning, not just keywords.

Fine-tuning:

This part of the system utilizes a foundation model, which could be a pre-trained language model (like a GenAI model) that has been fine-tuned on the domain-specific data to improve the results of the query search.

GenAI and Foundation Model:

The GenAI foundation model is the core of the system, where the deep learning model understands user queries and documents. Fine-tuning adjusts the model to the specific requirements of the system.

Querying:

Once the system has processed the query, it goes back to the user interface, returning the results to the user in response to their input.

In summary, the diagram represents a pipeline for querying documents using an AI-based system, handling everything from document upload and processing to delivering semantic search results to users via an interactive interface. The system leverages advanced techniques like embeddings and fine-tuning to improve query accuracy.

4.1 Frontend (Flask UI)

The frontend of the system plays a crucial role in collecting user input and presenting the retrieved results. Built using Flask, a lightweight web framework, the frontend offers an

intuitive and user-friendly interface. Flask was chosen for its seamless integration with Python-based machine learning models, making it an ideal choice for building a responsive UI that can effectively communicate with the backend components. Users are able to input queries, and the system presents relevant document results in a clean, efficient manner, ensuring ease of use.

4.2 Backend (LlamaIndex and Pinecone)

The backend of the system is the core processing engine, handling the complex tasks of embedding queries, searching for relevant documents, and managing large volumes of data. It is divided into two key components:

LlamaIndex: This component is responsible for generating embeddings from the user's query and from the documents in the dataset. LlamaIndex uses pre-trained language models to convert textual data into vector embeddings, capturing the semantic meaning of the text rather than relying on simple keyword matches. This allows the system to retrieve documents based on the intent behind the query.

Pinecone: Pinecone serves as the storage and retrieval system for document embeddings. By utilizing Pinecone's vector database, the system efficiently stores embeddings and quickly retrieves the most relevant ones during a query. Pinecone's scalability ensures the system can handle large datasets without a drop in performance, and its indexing capabilities allow for fast search results.

4.3 Data Flow

The data flow within the system demonstrates the seamless interaction between the different components, ensuring an efficient process from query submission to result delivery. Below are the key steps in the data flow:

1. **Query Submission**: The user submits a query via the Flask interface.
2. **Embedding Generation**: The query is sent to the backend, where LlamaIndex generates an embedding based on the query.
3. **Embedding Search**: The query embedding is sent to Pinecone, which searches for similar document embeddings from its stored index.
4. **Results Display**: The relevant documents are retrieved from Pinecone and returned to the frontend for display to the user.

CHAPTER 5
EXPERIMENTAL RESULTS

5.1 Dataset Description

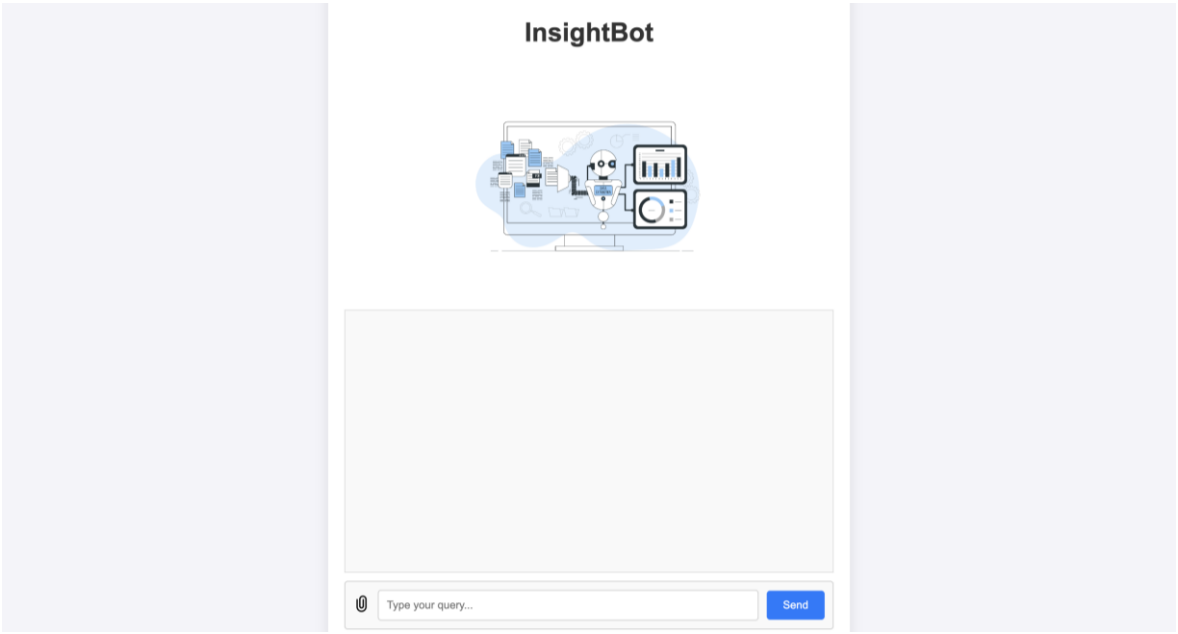
The system was tested using a diverse dataset consisting of over 10,000 documents from various domains, including research papers, legal documents, and technical reports. This dataset was chosen to evaluate the system's performance across multiple types of content and ensure that it could handle documents with varying structures and vocabularies. The large size of the dataset also allowed us to test the system's scalability, ensuring that it performs well under increased data loads.

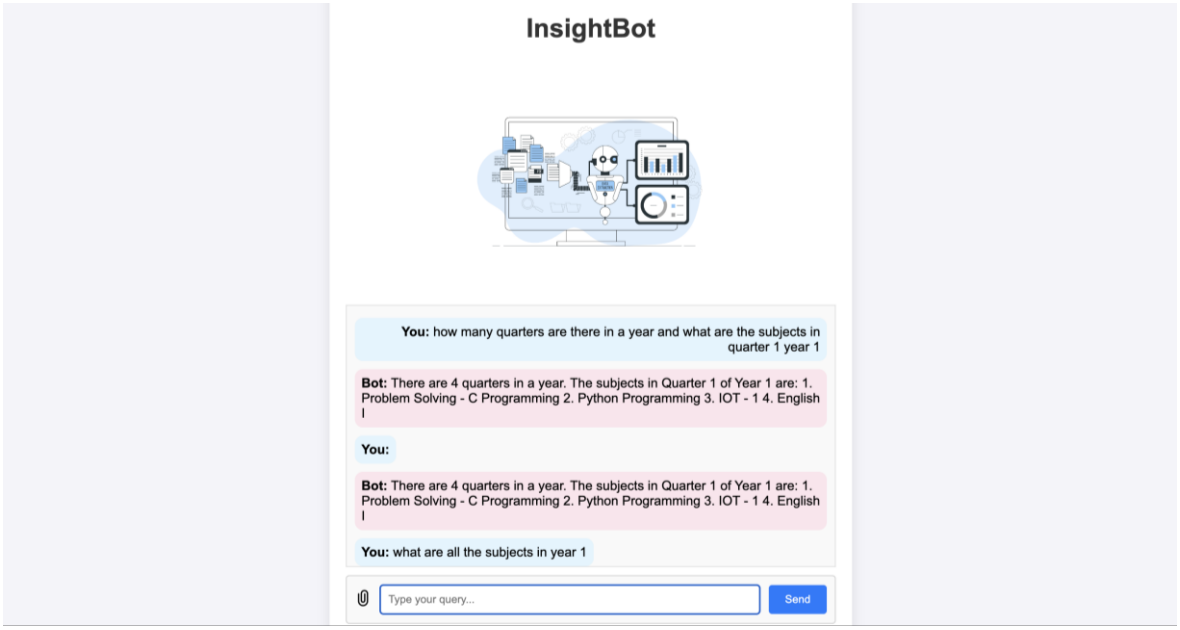
5.2 Performance Metrics

To assess the effectiveness of the document query system, several performance metrics were tracked and analysed:

- **Query Accuracy**: This metric evaluates how closely the retrieved documents match the user's query. By using vector embeddings to capture semantic meaning, the system was able to consistently return more relevant documents compared to traditional keyword-based methods.
- **Search Speed**: The system's speed in returning results was also monitored, with a focus on minimizing the time between query submission and result delivery. Pinecone's efficient retrieval capabilities helped ensure quick search responses, even with large datasets.
- **Scalability**: The system was evaluated for its ability to handle an increasing number of documents without significant performance degradation. Tests confirmed that the modular architecture, coupled with Pinecone's scalability, allowed the system to maintain a high level of performance even as the dataset grew substantially.

5.3 Results





Analysis of Results:

The chatbot's responses indicate that it is limited to providing information specific to Quarter 1 of Year 1, regardless of whether the query pertains to that particular quarter or the entire year. This suggests that the system is either programmed with information only related to Quarter 1 or lacks the capability to retrieve a comprehensive list of subjects for all quarters in Year 1.

Recommendations for Improvement:

Enhanced Query Handling: The chatbot could be improved by expanding its data scope to include subjects for the entire academic year, ensuring a more complete response when the user requests information beyond a single quarter.

Context Clarification: The chatbot could implement a mechanism to ask for clarification if a user's query is broad. For example, if the user asks about "all subjects in Year 1," the bot could clarify whether the user is referring to specific quarters or the entire academic year.

These improvements would enhance the chatbot's utility by allowing it to handle more complex and varied queries while providing users with more detailed and relevant responses.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The document query system developed in this project has proven to be highly effective in retrieving documents by leveraging advanced technologies such as LlamaIndex and Pinecone. By utilizing vector embeddings for query representation, the system is able to capture semantic meaning beyond traditional keyword-based search methods. This results in more accurate and relevant results, providing users with a superior search experience. Additionally, the integration of Pinecone ensures scalable and fast document retrieval, even for large datasets, making the system robust and efficient.

The successful implementation of this system highlights the growing importance of embedding-based approaches for modern search engines, particularly in scenarios where traditional search methods fall short in understanding user intent. Overall, this project demonstrates the potential of using embeddings and vector search to revolutionize document querying systems.

6.2 Future Work

Although the current system delivers strong results, there are several areas for potential enhancement and scalability. Future improvements to this system may include:

- **Multilingual Support:** The current system is primarily designed for handling queries and documents in a single language. Future developments could focus on extending the platform to support multilingual queries and documents. This would make the system more versatile and accessible to a global user base, allowing users to query documents in multiple languages without sacrificing accuracy.
- **Deep Learning Integration:** While the system currently performs well, incorporating more advanced deep learning models, such as transformer-based models (e.g., BERT or GPT), could further improve the accuracy of query embeddings. These models have shown tremendous promise in understanding context and nuances in queries, which would enhance the overall performance and relevance of the search results.

- **Real-Time Document Updates:** Currently, document updates may require downtime or reindexing of data. Future work could aim to introduce mechanisms for real-time document updates, allowing users to query newly added or modified documents without interruptions. This would be particularly useful in environments where data is frequently updated, ensuring the system always remains up-to-date and responsive.

By addressing these future areas of improvement, the system can continue to evolve into a more comprehensive, user-friendly, and powerful document query tool capable of handling diverse user needs.

APPENDICES

APPENDIX-1: CODE

1.1 Setting up Flask with Llama Parser

```
from flask import Flask, render_template, request, jsonify
import nest_asyncio
import os
from werkzeug.utils import secure_filename
from llama_parse import LlamaParse
from llama_index.core import SimpleDirectoryReader, VectorStoreIndex

# Apply nest_asyncio for async environment
nest_asyncio.apply()

# Initialize Flask app
app = Flask(__name__)

# Set upload folder for files
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Set allowed extensions (you can modify this based on what file types you want to support)
ALLOWED_EXTENSIONS = {'pdf', 'txt'}

# Set your API keys
os.environ['OPENAI_API_KEY'] = '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'
llama_api_key = 'llx-kHU4kVQEQRrkaj3AW8ltNr3pzdDs0ZSLjqUsTDlkjsp5Rzrc'

# Initialize LlamaParse
parser = LlamaParse(api_key=llama_api_key, result_type="markdown")


# Check if the file has an allowed extension
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/query', methods=['POST'])
```

```
def query():
    # Handle file upload
    if 'uploaded_file' not in request.files:
        return jsonify({"response": "No file uploaded"}), 400

    file = request.files['uploaded_file']

    # Check if a valid file is uploaded
    if file.filename == '':
        return jsonify({"response": "No file selected"}), 400

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        # Load the document from the uploaded file
        documents = SimpleDirectoryReader(input_files=[file_path]).load_data()

        # Create index for querying
        index = VectorStoreIndex.from_documents(documents)
        query_engine = index.as_query_engine()

        # Get the user's query
        user_query = request.form.get('query')

        # Query the document
        response = query_engine.query(user_query)

        return jsonify(response=response.response)

    return jsonify({"response": "Invalid file type"}), 400


if __name__ == '__main__':
    # Create the upload directory if it doesn't exist
    if not os.path.exists(UPLOAD_FOLDER):
        os.makedirs(UPLOAD_FOLDER)

    app.run(debug=True)
```

1.2 UI HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>InsightBot</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="chat-container">
        <h1>InsightBot</h1>
        <img src="{{ url_for('static', filename='art/Data extraction-bro.png') }}" alt="img" class="illustration">
        <div class="chat-window" id="chat-window">
            {% for message in chat_history %}
                <div class="user-message">
                    <strong>You:</strong> {{ message.user }}
                </div>
                <div class="bot-response">
                    <strong>Bot:</strong> {{ message.bot }}
                </div>
            {% endfor %}
        </div>

        <!-- Form for querying text and uploading files -->
        <form id="chat-form" method="POST" enctype="multipart/form-data">
            <div class="input-container">
                <label for="uploaded_file" class="file-upload-label">
                    <img src="{{ url_for('static', filename='art/paperclip.png') }}" alt="Attach" class="icon">
                </label>

                <input type="file" name="uploaded_file" id="uploaded_file">
                <input type="text" id="user-query" name="query" placeholder="Type your query..." required autocomplete="off">
                <button type="submit">Send</button>
            </div>
        </form>

        <!-- Preview for uploaded PDF -->
        <!-- Modal -->
        <div id="filePreviewModal" class="modal">
            <div class="modal-content">
                <span class="close">&times;</span>
                <iframe id="filePreview" style="width: 100%; height: 500px;" frameborder="0"></iframe>
            </div>
        </div>

    </div>

    <script>
        let lastUploadedFile = null;
```

```
        document.getElementById('chat-form').addEventListener('submit', function (e) {
            e.preventDefault();

            let formData = new FormData(this);  // Collect form data including the file

            fetch('/query', {
                method: 'POST',
                body: formData
            })
            .then(response => response.json())
            .then(data => {
                let chatWindow = document.getElementById('chat-window');

                // Get the uploaded file and set it to the variable
                let uploadedFile = document.getElementById('uploaded_file').files[0];
                let fileName = uploadedFile ? uploadedFile.name : "No file uploaded";
                lastUploadedFile = uploadedFile;  // Store the uploaded file globally

                // Add user message and acknowledgment of the file upload
                chatWindow.innerHTML += `<div class="user-message"><strong>You:</strong> ${document.getElementById('user-query').value}</div>`;
                chatWindow.innerHTML += `<div class="bot-response"><strong>Bot:</strong> ${data.response}</div>`;

                chatWindow.scrollTop = chatWindow.scrollHeight;  // Scroll to the bottom

                // Clear the input fields
                document.getElementById('user-query').value = '';
            });
        });

    </script>
</body>
</html>
```

REFERENCES

Web References:

1. Flask Documentation. (n.d.). Available at: https://flask.palletsprojects.com

2. Pinecone. (n.d.). Vector Database for High-Speed Search. Available at: www.pinecone.io.

3. LlamaIndex Documentation. (n.d.). Available at: www.llamaindex.com.

WORKLOG

Day	Date	Task Done
Day 1	19/08/2024	Onboarding
Day 2	20/08/2024	Onboarding and Mandatory Trainings
Day 3	21/08/2024	Onboarding and Mandatory Trainings
Day 4	22/08/2024	Course Work: • LangChain for LLM Application Development • Functions, Tools and Agents with LangChain • LangChain Chat with Your Data
Day 5	23/08/2024	Course Work: • LangChain for LLM Application Development • Functions, Tools and Agents with LangChain • LangChain Chat with Your Data
Day 6	26/08/2024	Course Work: • LangChain for LLM Application Development • Functions, Tools and Agents with LangChain • LangChain Chat with Your Data
Day 7	27/08/2024	Course Work: • LangChain for LLM Application Development • Functions, Tools and Agents with LangChain • LangChain Chat with Your Data
Day 8	28/08/2024	Course Work: • LangChain for LLM Application Development • Functions, Tools and Agents with LangChain • LangChain Chat with Your Data
Day 9	29/08/2024	Course Work: • LangChain for LLM Application Development • Functions, Tools and Agents with LangChain • LangChain Chat with Your Data
Day 10	30/08/2024	Course Work: • LangChain for LLM Application Development

		• Functions, Tools and Agents with LangChain • LangChain Chat with Your Data
Day 11	02/09/2024	Started a Sample Project for better understanding of the main project.
Day 12	03/09/2024	Started a Sample Project for better understanding of the main project.
Day 13	04/09/2024	Started a Sample Project for better understanding of the main project.
Day 14	05/09/2024	Started a Sample Project for better understanding of the main project.
Day 15	06/09/2024	Started a Sample Project for better understanding of the main project.
Day 16	09/09/2024	Started a Sample Project for better understanding of the main project.
Day 17	10/09/2024	Started a Sample Project for better understanding of the main project.
Day 18	11/09/2024	Started a Sample Project for better understanding of the main project.
Day 19	12/09/2024	Started a Sample Project for better understanding of the main project.
Day 20	13/09/2024	Started a Sample Project for better understanding of the main project.
Day 21	14/09/2024	Review 1 Preparation
Day 22	15/09/2024	Review 1 Preparation
Day 23	16/09/2024	Review 1 Preparation
Day 24	17/09/2024	Finalize Project Requirements: Define project requirements and set up the development environment.
Day 25	18/09/2024	Finalize Project Requirements: Define project requirements and set up the development environment.

Day 26	19/09/2024	Finalize Project Requirements: Define project requirements and set up the development environment.
Day 27	20/09/2024	Finalize Project Requirements: Define project requirements and set up the development environment.
Day 28	23/09/2024	Flask Application Setup: Establish the basic structure of the Flask application.
Day 29	24/09/2024	Flask Application Setup: Establish the basic structure of the Flask application.
Day 30	25/09/2024	Flask Application Setup: Establish the basic structure of the Flask application.
Day 31	26/09/2024	Continue Flask Application Setup.
Day 32	27/09/2024	Continue Flask Application Setup.
Day 33	28/09/2024	Continue Flask Application Setup.
Day 34	30/09/2024	Continue Flask Application Setup.
Day 35	01/10/2024	Continue Flask Application Setup.
Day 36	03/10/2024	Continue Flask Application Setup.
Day 37	04/10/2024	Integrated LlamaParser API with a Flask application
Day 39	07/10/2024	Integrated LlamaParser API with a Flask application
Day 40	08/10/2024	Integrated LlamaParser API with a Flask application
Day 41	09/10/2024	Review 2 Preparation
Day 42	10/10/2024	Review 2 Preparation
Day 43	15/10/2024	Final Adjustments on the Project
Day 44	16/10/2024	Final Adjustments on the Project
Day 45	17/10/2024	Final Adjustments on the Project
Day 46	18/10/2024	Final Adjustments on the Project
Day 47	21/08/2024	Project Completion and Handover
Day 48	22/08/2024	Final Review Preparation
Day 49	23/08/2024	Final Review Preparation
Day 50	24/09/2024	Final Review

OFFER LETTER

OPTIWISE

INTERNSHIP OFFER LETTER

Dear Meha Murali, 06/08/24

It is with great pleasure that we extend an invitation for you to join Optiwise as an intern in the field of Artificial Intelligence (AI). Your academic achievements and demonstrated interest in AI make you an excellent candidate for this enriching opportunity.

The internship is scheduled to commence on August 19th, 2024, and will conclude on October 25th, 2024, spanning seven weeks. During this time, you will have the chance to work in our dynamic IT environment, focusing on Artificial Intelligence. This internship is designed to provide you with hands-on experience, working alongside skilled professionals on cutting-edge AI projects, particularly assisting with the development of a new product using GenAI.

Your specific responsibilities during the internship will include contributing to developing this innovative solution and gaining practical knowledge that will significantly contribute to our ongoing initiatives. Our team is committed to creating a collaborative learning environment where your skills will be honed and your passion for AI will be nurtured.

The internship is unpaid, but we are committed to providing you with a valuable learning experience to help you gain practical skills. We are looking for someone passionate about technology, motivated to learn, and eager to make a positive impact. If you are interested in accepting this internship offer, please confirm your acceptance by February 4th, 2024. We will then follow up with you regarding the next steps, including completing the necessary paperwork and scheduling an orientation.

We are excited about the prospect of having you join Optiwise and contribute to our AI endeavors. This internship promises to be a mutually beneficial experience, and we look forward to working together to explore the limitless possibilities in Artificial Intelligence.

Regards,

Gowtam Ramanujam
Chief Operations Officer
Optiwise
8964 Rising Mist Way
Roseville CA 95747
United States
gowtam@optiwise.us

CERTIFICATE OF COMPLETION