TABLE OF CONTENTS

1. INTRODUCTION

The **AI PDF Question Answering System** is an offline, desktop-based application designed to facilitate quick and accurate information retrieval from PDF documents, including those with scanned text. It implements a **Retrieval-Augmented Generation (RAG)** architecture using local, open-source models, enabling live-streaming answers and support for Optical Character Recognition (OCR).

1.1 Purpose:

The objective of the AI PDF Question Answering System is to allow a user to upload any PDF and instantly query its content, effectively turning the document into a searchable knowledge base. This system rectifies the manual difficulty of searching through lengthy documents, providing fast, context-aware answers.

1.2 Scope:

The system stores, processes, and indexes the entire text of a single uploaded PDF. It utilizes a local embedding model for vector storage and a local LLM via Ollama for generation, making it an entirely offline solution.

2. SYSTEM ANALYSIS

2.1 Existing System: Manual PDF Search

The current, or existing, system involves manually reading, searching, or using basic text search functions within PDF viewers. This is time-consuming, especially for finding conceptual answers or information hidden in non-searchable scanned pages. This manual process is often inefficient and prone to missing relevant context.

2.2 Proposed System - AI PDF Question Answering System

The proposed system addresses these issues by:

- **Automated Text Extraction:** Extracts all text, including a fallback to OCR for image-based pages.
- **Semantic Search (RAG):** Allows users to ask natural language questions, retrieving context semantically rather than just by keywords.
- **Live Streaming:** Provides a smooth, user-friendly experience by streaming the LLM's answer in real-time.
- **Offline Operation:** Utilizes local models for enhanced security and faster performance without relying on cloud APIs.

2.3 Feasibility Analysis

2.3.1 Technical Feasibility:

This system is technically feasible as it relies on established Python libraries (Streamlit, PyMuPDF, Faiss, Pytesseract) and the readily available open-source LLM server, Ollama. All necessary resources and technologies for development and maintenance are easily accessible.

2.3.2 Economical Feasibility

Development of this application is highly economically feasible. The system utilizes open-source software (Python, Streamlit, Ollama, Faiss) and local resources, avoiding recurrent subscription or API service fees associated with cloud-based LLMs.

3. SYSTEM ARCHITECTURE

The system follows a Retrieval-Augmented Generation (RAG) architecture, primarily composed of three stages: Document Processing, Retrieval, and Generation.

3.1 Overall System Architecture

The system's flow begins with the user uploading a PDF via the Streamlit interface.

1. **PDF Upload & Extraction:** The Streamlit frontend accepts the PDF file. The system then extracts text, attempting standard text extraction first and falling back to **OCR** (pytesseract) for image-based or scanned pages.
2. **Chunking & Indexing:** The extracted raw_text is split into smaller, overlapping chunks (using RecursiveCharacterTextSplitter). These chunks

are then converted into numerical embeddings using SentenceTransformer and indexed by **Faiss** for high-speed similarity search.

3. **Question & Retrieval:** When a question is entered, it is also converted into an embedding. Faiss searches the index to retrieve the top K most similar text chunks (the context).
4. **Generation:** The retrieved context and the question are formatted into a prompt and sent to the **Ollama LLM** (mistral) via an HTTP POST request. The LLM generates the final answer, which is streamed back to the Streamlit UI.

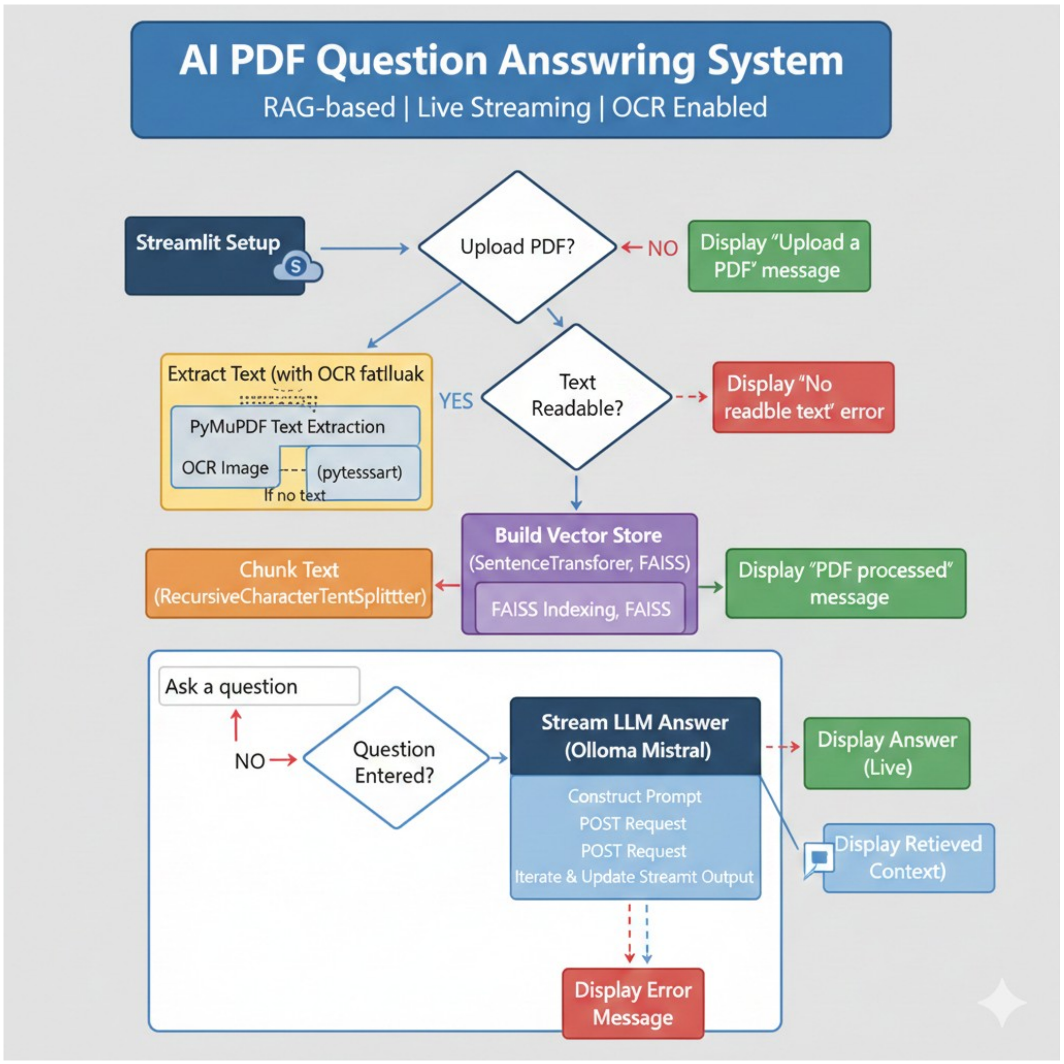
3.2 Modules

The system's logic is modularized into the following functional components:

Module	**Description**	**Key Components/Function**
UI Module	Handles user input (file upload, question) and displays all output.	streamlit, @st.cache_resource, st.file_uploader, st.text_input
Extraction Module	Extracts text from the PDF, including OCR fallback for scanned pages.	extract_pdf_text_with_ocr(), fitz (PyMuPDF), pytesseract (OCR)
Vector Store Module	Chunks text, creates embeddings, and builds the Faiss index.	chunk_text(), build_vector_store(), SentenceTransformer, Faiss
RAG/Retrieval Module	Finds the most relevant text chunks (context) for a given question using vector search.	retrieve_context(), Faiss index.search()
LLM Streaming Module	Generates the final, structured answer from the LLM and streams it to the UI.	stream_llm_answer(), requests (to Ollama), LLM_MODEL (mistral)

3.3 Data Flow Diagram

Figure: Data Flow Diagram (AI PDF QA System)


AI PDF Question Ansswring System
RAG-based | Live Streaming | OCR Enabled
Streamlit Setup
Upload PDF?
NO
Display "Upload a PDF" message
Extract Text (with OCR fatlluak
PyMuPDF Text Extraction
OCR Image
(pytesssart)
If no text
YES
Text Readable?
Display "No readable text" error
Build Vector Store (SentenceTransforer, FAISS)
FAISS Indexing, FAISS
Chunk Text (RecursiveCharacterTentSplittter)
Display "PDF processed" message
Ask a question
NO
Question Entered?
Stream LLM Answer (Olloma Mistral)
Construct Prompt
POST Request
POST Request
Iterate & Update Streamt Output
Display Answer (Live)
Display Retieved Context)
Display Error Message

The Data Flow Diagram (DFD) illustrates the movement of information through the system.

4. OVERALL DESCRIPTION

4.1 Product Perspective:

The AI PDF Question Answering System is a stand-alone, single-file application built using Python and the Streamlit framework. It is designed to be an entirely self-contained, offline product, relying only on a locally hosted Ollama server and its required Python dependencies.

4.2 Product Functions:

The product supports a single user role (the User) with the following core features:

- **PDF Upload (OCR Enabled):** Allows uploading any PDF, ensuring all text is extracted, even from scanned pages.
- **Document Processing:** Automatically handles chunking and indexing of the document content.
- **Question Answering:** Enables the user to input natural language questions related to the document.
- **Live Streamed Answer:** Delivers the LLM-generated response in real-time to the user interface.
- **Context Transparency:** Provides an expandable section to view the specific retrieved context used by the LLM to form its answer.

4.3 User Classes and Characteristics:

There is one class of user for this product:

- **User:** Any individual who needs quick, precise answers from a PDF document. Requires a basic understanding of computer use and the ability to operate the Streamlit web interface.

4.4 Operating Environment:

The product runs in a local environment, requiring:

- **OS:** Any operating system supporting Python (Windows, macOS, Linux).
- **Server:** Local running instance of **Ollama** serving the mistral LLM.
- **Interface:** Any modern web browser (the application is served via **Streamlit**).

4.5 Constraints:

- **Offline Dependency:** Requires the Ollama server to be running locally at http://localhost:11434.
- **Language:** Assumes the document and queries are primarily in English (due to the LLM and OCR setup).
- **File Type:** Limited to PDF files.

4.6 Use Case Model:

Use Case	**Brief Description**
Upload and Process PDF	The user uploads a PDF. The system extracts text (with OCR), chunks it, and builds the Faiss vector index.

Use Case	**Brief Description**
Ask Question	The user inputs a question into the text box.
Retrieve Context	The system converts the question to an embedding and retrieves the top **3** relevant text chunks from the Faiss index.
Stream Answer	The system sends the retrieved context and question to Ollama and streams the response token by token to the UI.
View Context	The user can expand a UI element to see the exact context used for the answer.

5. TECHNOLOGY OVERVIEW

The technology selected for implementing the AI PDF Question Answering System is built on a modern, open-source RAG stack.

5.1 Python

Python is the core programming language used to implement all the logic, from the Streamlit UI to PDF processing, vector indexing, and communication with the LLM server.

5.2 Streamlit

Streamlit is the application framework used to create the graphical, interactive web interface. It allows for rapid development of the front-end components, such as file uploaders, text input, and the dynamic output box for streaming.

5.3 PyMuPDF (fitz) and PyTesseract

- **PyMuPDF:** A highly efficient library used to extract text and image data from PDF files.
- **PyTesseract:** The Python wrapper for Google's Tesseract OCR engine, used as a fallback mechanism to extract text from scanned pages or images within the PDF.

5.4 Faiss and SentenceTransformer

- **SentenceTransformer:** Used to load the all-MiniLM-L6-v2 model, which generates high-quality vector embeddings (numerical representations) of the text chunks and the user's question.

- **Faiss:** A library for efficient similarity search and clustering of dense vectors. It is used to build the vector index, enabling fast retrieval of context chunks.

5.5 Ollama and Mistral

- **Ollama:** A powerful framework used to run and manage large language models locally. It provides a simple API endpoint (http://localhost:11434/api/generate) for interaction.
- **Mistral:** The specific LLM (LLM_MODEL = "mistral") selected for its superior reasoning capabilities and efficiency for RAG-based question answering.

6. PROJECT DESCRIPTION

6.1 Introduction

The AI PDF Question Answering System solves the problem of information overload in large documents by applying RAG. It ensures that the LLM's response is grounded only in the document's content, reducing hallucination and providing verifiable answers.

6.2 Description Of Core Functions

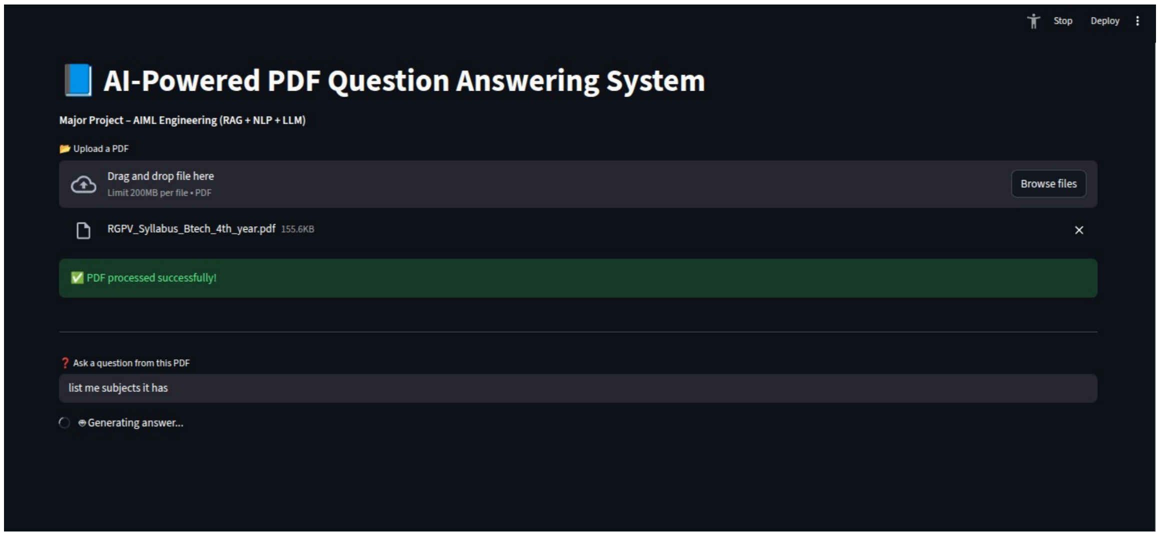
Function Name	Description	Key Code Sections
load_embedder()	Loads the SentenceTransformer model once and caches it for performance.	@st.cache_resource, SentenceTransformer(EMBED_MODEL)
ocr_image()	Performs OCR on image bytes extracted from the PDF.	pytesseract.image_to_string(image)
extract_pdf_text_with_ocr()	Iterates through PDF pages, extracts text, and falls back to ocr_image() if a page has no searchable text.	fitz.open(), page.get_text(), page.get_images()
chunk_text()	Splits the raw text into manageable chunks with overlap for contextual continuity.	RecursiveCharacterTextSplitter (chunk_size=500)
build_vector_store()	Embeds the text chunks and creates the Faiss index for storage and search.	embedder.encode(), faiss.IndexFlatL2(), index.add(embeddings)

Function Name	**Description**	**Key Code Sections**
retrieve_context()	Encodes the question and searches the Faiss index to return the top **3** relevant text chunks.	index.search(q_emb, TOP_K)
stream_llm_answer()	Constructs the RAG prompt with the context, calls the Ollama API, and streams the decoded response to the Streamlit output box.	requests.post(OLLAMA_URL, stream=True), response.iter_lines()

7. SCREENSHOTS

Figure 7.1: PDF Upload and Processing Interface

Figure 7.1: Successful PDF Processing and Initial Query



📘 AI-Powered PDF Question Answering System

Major Project – AIML Engineering (RAG + NLP + LLM)

📂 Upload a PDF

RGPV_Syllabus_Btech_4th_year.pdf 155.6KB

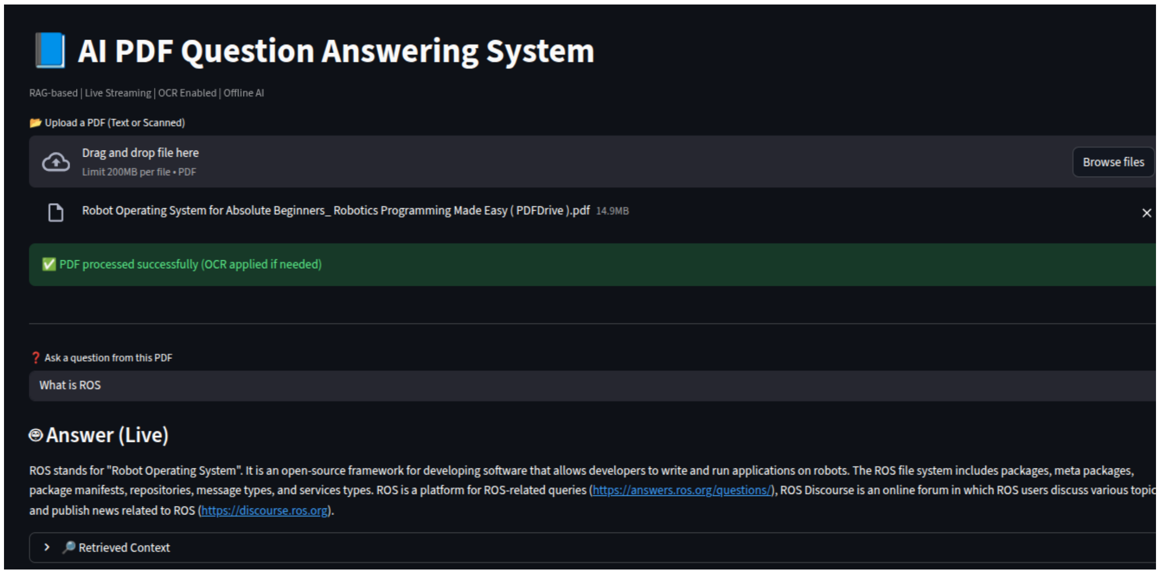
✅ PDF processed successfully!

❓ Ask a question from this PDF

list me subjects it has

Generating answer...

Figure 7.2: RAG Answer Streaming with Context Retrieved



📘 AI PDF Question Answering System

RAG-based | Live Streaming | OCR Enabled | Offline AI

📂 Upload a PDF (Text or Scanned)

Drag and drop file here
Limit 200MB per file • PDF

Browse files

Robot Operating System for Absolute Beginners_ Robotics Programming Made Easy (PDFDrive).pdf 14.9MB

✅ PDF processed successfully (OCR applied if needed)

❓ Ask a question from this PDF

What is ROS

Answer (Live)

ROS stands for "Robot Operating System". It is an open-source framework for developing software that allows developers to write and run applications on robots. The ROS file system includes packages, meta packages, package manifests, repositories, message types, and services types. ROS is a platform for ROS-related queries (https://answers.ros.org/questions/), ROS Discourse is an online forum in which ROS users discuss various topic and publish news related to ROS (https://discourse.ros.org).

> 🔎 Retrieved Context

8. CODING

Python

```
# ============================================================
# AI PDF Question Answering System
# RAG + LIVE STREAMING + OCR SUPPORT
# Single File | AIML Major Project
# ============================================================

import streamlit as st
import fitz  # PyMuPDF
import faiss
import numpy as np
import requests
import json
import pytesseract
from PIL import Image
import io

from sentence_transformers import SentenceTransformer
from langchain_text_splitters import RecursiveCharacterTextSplitter


# ---------------- CONFIG ----------------
EMBED_MODEL = "all-MiniLM-L6-v2"
LLM_MODEL = "mistral"      # better reasoning than llama3
```

```
TOP_K = 3
MAX_CONTEXT_CHARS = 1500
OLLAMA_URL = "http://localhost:11434/api/generate"


# ---------------- STREAMLIT SETUP ----------------
st.set_page_config("AI PDF Chat with OCR", layout="wide")
st.title("📘 AI PDF Question Answering System")
st.caption("RAG-based | Live Streaming | OCR Enabled | Offline AI")


# ---------------- LOAD EMBEDDER ----------------
@st.cache_resource
def load_embedder():
    return SentenceTransformer(EMBED_MODEL)

embedder = load_embedder()


# ---------------- OCR FROM IMAGE ----------------
def ocr_image(image_bytes):
    image = Image.open(io.BytesIO(image_bytes)).convert("RGB")
    return pytesseract.image_to_string(image)


# ---------------- PDF TEXT EXTRACTION (TEXT + OCR) ----------------
def extract_pdf_text_with_ocr(uploaded_file):
    doc = fitz.open(stream=uploaded_file.read(), filetype="pdf")
    full_text = ""

    for page_num, page in enumerate(doc):
        page_text = page.get_text().strip()

        # If text exists, use it
        if page_text:
            full_text += page_text + "\n"
        else:
            # OCR fallback for scanned page
            images = page.get_images(full=True)
            for img in images:
                xref = img[0]
                base_image = doc.extract_image(xref)
                image_bytes = base_image["image"]
                ocr_text = ocr_image(image_bytes)
                full_text += ocr_text + "\n"

    return full_text


# ---------------- TEXT CHUNKING ----------------
def chunk_text(text):
    splitter = RecursiveCharacterTextSplitter(
        chunk_size=500,
        chunk_overlap=100
    )
    return splitter.split_text(text)


# ---------------- VECTOR STORE ----------------
def build_vector_store(chunks):
    if not chunks:
        raise ValueError("No readable text found in PDF.")

    embeddings = embedder.encode(chunks)

    if len(embeddings.shape) == 1:
        embeddings = np.array([embeddings])

    embeddings = embeddings.astype("float32")
    dim = embeddings.shape[1]

    index = faiss.IndexFlatL2(dim)
    index.add(embeddings)

    return index, chunks


# ---------------- RETRIEVAL ----------------
def retrieve_context(question, index, chunks):
    q_emb = embedder.encode([question]).astype("float32")
    _, idxs = index.search(q_emb, min(TOP_K, len(chunks)))
    context = "\n".join(chunks[i] for i in idxs[0])

    # Deduplicate lines
    context = "\n".join(dict.fromkeys(context.split("\n")))
    return context[:MAX_CONTEXT_CHARS]
```

```
# ---------------- STREAMING LLM ----------------
def stream_llm_answer(context, question, output_box):
    prompt = f"""
You are an intelligent academic assistant.

INSTRUCTIONS:
- Use the context ONLY as a knowledge source
- Do NOT copy sentences verbatim
- Explain in your own words
- Be clear and structured
- If not found, say: "Not found in the document"

Context:
{context}

Question:
{question}

Final Answer:
"""

    payload = {
        "model": LLM_MODEL,
        "prompt": prompt,
        "stream": True,
        "options": {
            "temperature": 0.3,
            "num_predict": 250,
            "top_p": 0.9
        }
    }

    response = requests.post(
        OLLAMA_URL,
        json=payload,
        stream=True,
        timeout=30
    )

    full_answer = ""

    for line in response.iter_lines():
        if line:
                data = json.loads(line.decode("utf-8"))
                token = data.get("response", "")
                full_answer += token
                output_box.markdown(full_answer)

        return full_answer


# ---------------- UI ----------------
uploaded_pdf = st.file_uploader("📂 Upload a PDF (Text or Scanned)", type=["pdf"])

if uploaded_pdf:
        with st.spinner("📄 Extracting text (OCR enabled)..."):
                raw_text = extract_pdf_text_with_ocr(uploaded_pdf)

        if len(raw_text.strip()) < 50:
                st.error("❌ No readable text detected, even after OCR.")
                st.stop()

        with st.spinner("✂️ Chunking text..."):
                chunks = chunk_text(raw_text)

        with st.spinner("🧠 Building vector database..."):
                index, stored_chunks = build_vector_store(chunks)

        st.success("✅ PDF processed successfully (OCR applied if needed)")

        st.divider()
        question = st.text_input("❓ Ask a question from this PDF")

        if question:
                with st.spinner("🔍 Retrieving context..."):
                        context = retrieve_context(
                                f"In the context of the document, {question}",
                                index,
                                stored_chunks
                        )

                st.subheader("🤖 Answer (Live)")
                answer_box = st.empty()

                try:
                        stream_llm_answer(context, question, answer_box)
                except Exception as e:
```

```
            st.error(f"❌ Error: {e}")

        with st.expander("🔎 Retrieved Context"):
            st.write(context)

else:
    st.info("⬆️ Upload a PDF to begin")
```

9. BIBLIOGRAPHY

(Note: Replace your old bibliography with the sources relevant to the new technology stack.)

- Python Documentation
- Streamlit Documentation
- PyMuPDF (fitz) Documentation
- Faiss Documentation (Facebook AI Similarity Search)
- Sentence-Transformers Documentation
- Ollama Documentation
- Tesseract OCR Documentation