RAG Pipeline Project Report (Machine Learning PDFs)

SAMRA NAZ

---

1. Project Overview

This project implements a **Retrieval-Augmented Generation (RAG)** pipeline that allows users to query machine learning (ML) concepts from PDF documents. The pipeline extracts ML-related text from PDFs, generates embeddings, stores them in FAISS for fast retrieval, and uses a Question-Answering (QA) model to produce concise answers.

---

2. Tools & Libraries Used

Library / Tool	Purpose
`pypdf`	Extract text from PDF files
`sentence-transformers`	Generate text embeddings
`faiss`	Fast similarity search (vector database)
`transformers`	QA model (DistilBERT)
`flask`	Web app and API development
`numpy`	Handle arrays and embeddings
`os`	File system operations

---

3. Step-by-Step Code Explanation

Imports & Flask Setup

```
from pypdf import PdfReader
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np
from transformers import pipeline
from flask import Flask, request, jsonify, render_template, flash
import os

app = Flask(__name__)
app.secret_key = "rag_secret_key"
```

- Imports required libraries.

- Initializes Flask app and secret key for sessions and flash messages.

PDF Loading & Filtering

```
pdf_path = "machine_learning.pdf"
reader = PdfReader(pdf_path)
keywords = ["machine learning", "ML", "supervised", "unsupervised", "deep
learning", "neural network"]
documents = []
for page in reader.pages:
    text = page.extract_text()
    if text:
        text = text.strip()
        if any(kw.lower() in text.lower() for kw in keywords):
            documents.append(text)
```

- Reads PDF and extracts only ML-related paragraphs using keywords.

Validation

```
if not documents:
    raise ValueError("PDF is empty or no ML-related text extracted!")
```

- Ensures relevant text exists.

Embeddings Generation

```
model_path = "models/all-MiniLM-L6-v2"
if os.path.exists(model_path):
    embedding_model = SentenceTransformer(model_path)
else:
    embedding_model = SentenceTransformer("sentence-transformers/all-MiniLM-
L6-v2")

doc_embeddings = embedding_model.encode(documents, convert_to_numpy=True)
```

- Loads local embedding model or downloads from HuggingFace.
- Generates embeddings for each document.

FAISS Index

```
dimension = doc_embeddings.shape[1]
faiss_index = faiss.IndexFlatL2(dimension)
faiss_index.add(doc_embeddings)
```

- Creates FAISS index for similarity search and adds embeddings.

QA Model Setup

```
qa_model_path = "models/distilbert-base-cased-distilled-squad"
if os.path.exists(qa_model_path):
    qa_pipeline = pipeline("question-answering", model=qa_model_path)
else:
```

```
    qa_pipeline = pipeline("question-answering", model="distilbert-base-
cased-distilled-squad")
```

- Loads local QA model or downloads from HuggingFace.

RAG Pipeline Function

```
def rag_pipeline(query, max_sentences=3):
    if not query.strip():
        return "Please enter a valid question."

    D, I = faiss_index.search(embedding_model.encode([query]), k=3)
    retrieved_contexts = [documents[i] for i in I[0]]
    retrieved_context = " ".join(retrieved_contexts)

    if not retrieved_context.strip():
        return "No relevant ML information found in the documents."

    result = qa_pipeline(question=query, context=retrieved_context)
    answer = result.get('answer', 'Unable to generate answer.')

    sentences = answer.split('. ')
    short_answer = '. '.join(sentences[:max_sentences]).strip()
    if not short_answer.endswith('.'):
        short_answer += '.'

    return short_answer
```

- Takes user query, retrieves top 3 relevant documents using FAISS, and generates a concise answer with QA model.

API Route

```
@app.route("/ask", methods=["POST"])
def ask():
    data = request.json
    query = data.get("query", "")
    answer = rag_pipeline(query)
    return jsonify({"query": query, "answer": answer})
```

- JSON API endpoint for programmatic access.

HTML Form Route

```
@app.route("/", methods=["GET", "POST"])
def home():
    answer = None
    if request.method == "POST":
        query = request.form.get("query", "")
        if not query:
            flash("Please enter a question before submitting!")
        else:
```

```
        answer = rag_pipeline(query)
    return render_template("index.html", answer=answer)
```

- Web interface for user input and displaying answers.

Running Flask

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

- Starts Flask server locally.

---

4. Workflow Summary

1. Load PDF → Extract ML-related text → Filter paragraphs
2. Generate embeddings → Store in FAISS index
3. User submits query → Convert to embedding → Retrieve top-k documents
4. Pass retrieved context to QA model → Generate answer
5. Display answer in web interface or API

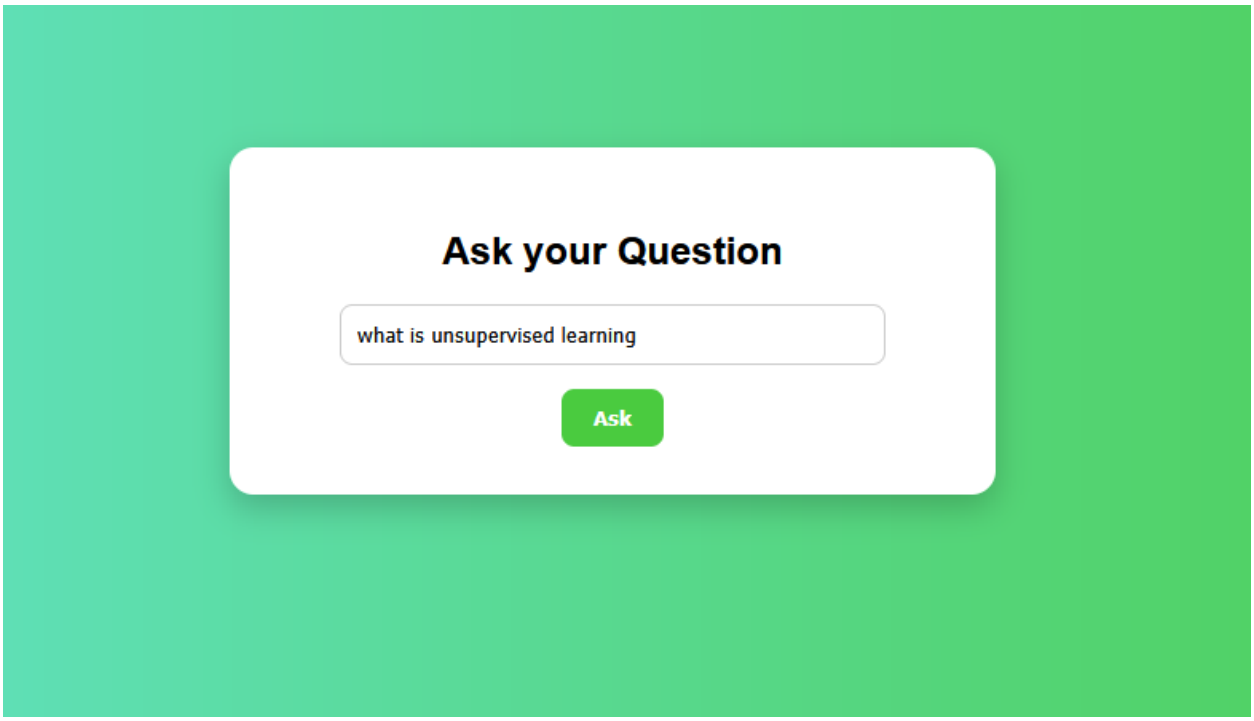
---

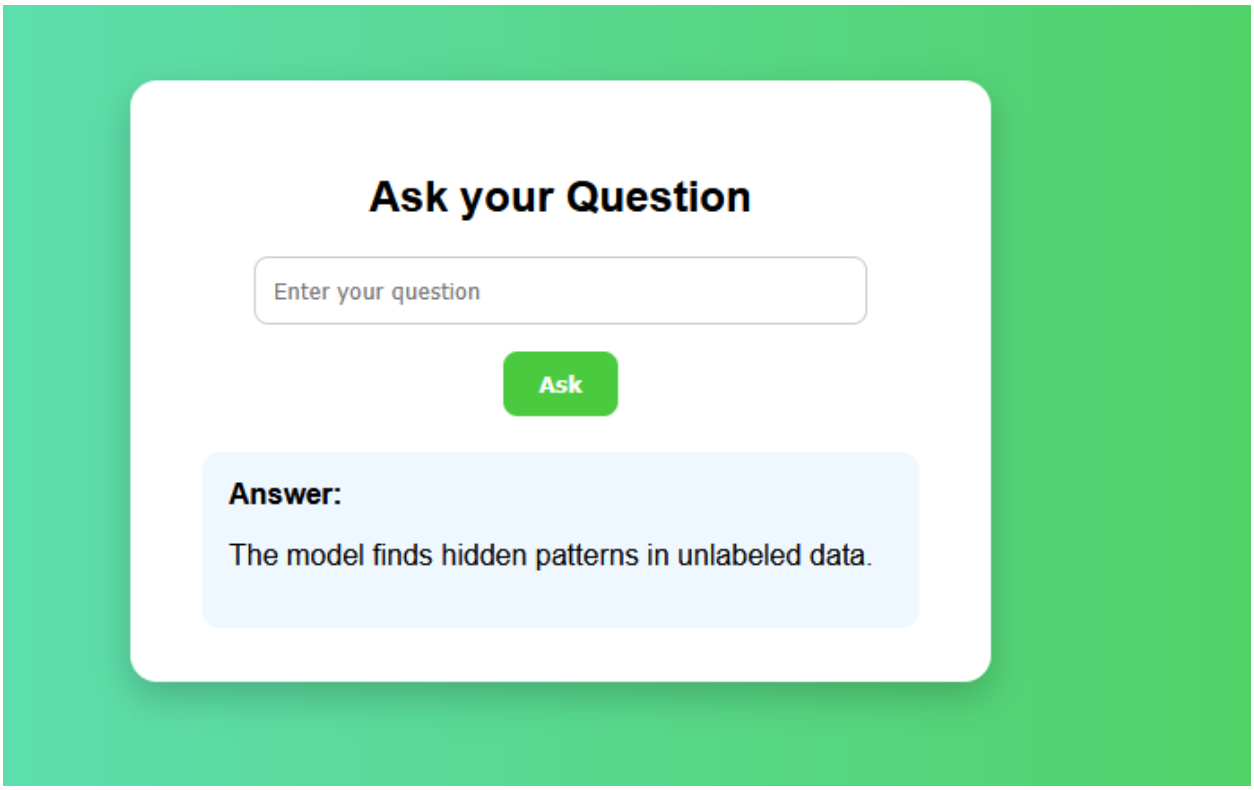
5. Screenshots Placeholders

- **FAISS index creation**: Print `faiss_index.ntotal`
- **Retrieved context**: Print `retrieved_contexts`
- **Final Answer**: Print output of `rag_pipeline("Your Question")`

---

6. out put

7. 
Ask your Question
what is unsupervised learning
Ask


Ask your Question
Enter your question
Ask
Answer:
The model finds hidden patterns in unlabeled data.