

# IntelliDocs: A Comprehensive AI-Powered Document Exploration & Summarization Platform

**Author:** Vaibhav Vesmaker

## 1. Introduction

IntelliDocs is a next-generation web application designed to help users **upload, explore, and summarize** large bodies of text—such as academic papers, policy documents, legal briefs, or long-form articles—using cutting-edge AI. This tool leverages a *Retrieval-Augmented Generation (RAG)* approach, coupled with Large Language Models (LLMs), to deliver responsive, contextually grounded insights in real-time.

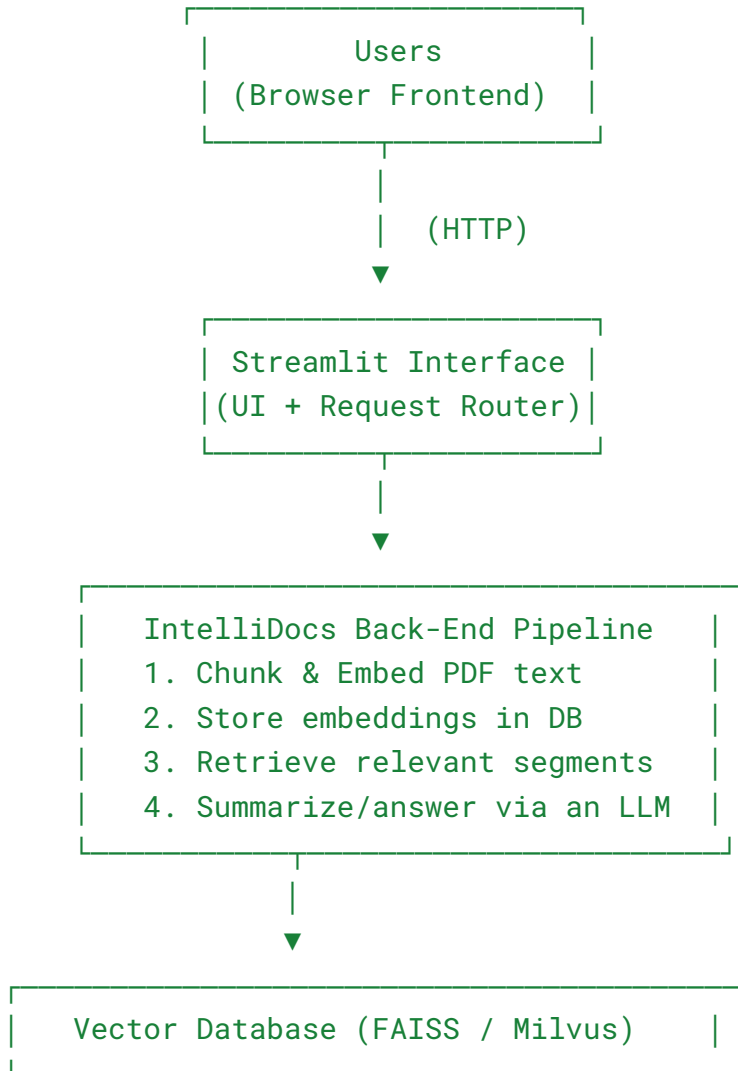
## Key Highlights

- 1. User-Centric Interface**
  - Built with [Streamlit](#) for an intuitive, interactive web experience.
  - Simple file upload flow, with immediate feedback on document processing and summarization.
- 2. Robust Document Processing**
  - Automated chunking of PDFs into manageable text segments.
  - Embedding-based retrieval using a vector database, enabling rapid and relevant content lookups.
- 3. Concise, Context-Rich Answers**
  - Customizable answer lengths (short summaries or in-depth explanations).
  - Retrieval of the most relevant segments to reduce off-topic or “hallucinated” information from the model.
- 4. Modular, Developer-Friendly Framework**
  - Clear separation between front-end UI, back-end retrieval logic, and LLM inference pipelines.
  - Adaptable to various embedding techniques (local or cloud-based) and multiple vector databases.
- 5. Scalable Deployment**
  - Streamlit-based solution that can run locally or on major cloud platforms (e.g., AWS, Azure, GCP).
  - Optional Docker support for containerized environments and consistent deployments.

## 2. Architecture Overview

At a high level, IntelliDocs consists of a simple flow:

vbnet  
CopyEdit



### Workflow in Detail

1. **Document Upload**: Users upload a PDF, which the system reads and parses.
2. **Segment & Index**: The text is chunked into segments; each chunk is turned into an embedding and stored in a vector database.
3. **Query Processing**: When a question is asked, IntelliDocs searches the vector database for chunks most relevant to the query.

4. **Answer Generation:** A Large Language Model (LLM) uses those chunks to compose a concise or extended answer.
5. **Result Display:** The final answer is returned to the user in the Streamlit app.

## 3. Technology Choices

- **Python (>=3.8)** – Primary programming language.
- **Streamlit** – Front-end web framework.
- **PDF Parsing** – [pdfplumber](#) or [PyPDF2](#).
- **Vector Database** – [FAISS](#), [Milvus](#), or [Chroma](#) for storing and querying embeddings.
- **Embedding Model** – Could be local (e.g., a self-hosted LLM) or cloud-based (e.g., OpenAI embeddings).
- **LLM for Answer Generation** – For example, a locally hosted model like [DeepSeek-R1:1.5b](#), [Ollama](#), or a cloud API such as GPT-3.5/GPT-4.

## 4. Detailed Implementation

### 4.1 Project Structure

A sample folder layout:

```
graphql
CopyEdit
IntelliDocs/
├─ README.md
├─ requirements.txt
├─ Dockerfile (optional)
├─ .gitignore
├─ app.py (Streamlit entry point)
├─
├─ src/
│   ├─ chunking.py          # PDF parsing and text segmentation
│   ├─ embeddings.py        # Generating and storing vector embeddings
│   ├─ retrieval.py         # Searching the vector database
│   ├─ rag_pipeline.py      # Orchestration layer for RAG
│   └─ llm_inference.py     # Interfacing with chosen LLM
├─
├─ data/
│   └─ sample.pdf           # Example PDF for testing
├─
```

```
└─ docs/
  └─ architecture.png # Any diagrams or references
```

## 4.2 Requirements Installation

Create a `requirements.txt`:

```
text
CopyEdit
streamlit==1.21.0
pdfplumber==0.9.0
faiss-cpu==1.7.2
numpy==1.23.5
requests==2.28.1
# Additional dependencies for LLM, e.g., OpenAI or local models
openai==0.27.0
pypdf2==3.0.1
```

Install via:

```
bash
CopyEdit
pip install -r requirements.txt
```

## 4.3 Parsing & Chunking (src/chunking.py)

```
python
CopyEdit
import pdfplumber

def extract_text_from_pdf(pdf_path):
    all_text = []
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            page_text = page.extract_text()
            if page_text:
                all_text.append(page_text)
    return "\n".join(all_text)
```

```
def chunk_text(text, chunk_size=500):
    words = text.split()
    chunks = []
    current_chunk = []

    for word in words:
        current_chunk.append(word)
        if len(current_chunk) >= chunk_size:
            chunks.append(" ".join(current_chunk))
            current_chunk = []

    if current_chunk:
        chunks.append(" ".join(current_chunk))

    return chunks
```

#### 4.4 Generating Embeddings (src/embeddings.py)

python

CopyEdit

```
import openai
```

```
def get_embedding(text, model="text-embedding-ada-002"):
    response = openai.Embedding.create(
        input=[text],
        model=model
    )
    return response["data"][0]["embedding"]
```

*(Replace with local model calls if using self-hosted embeddings.)*

#### 4.5 Vector Database & Retrieval (src/retrieval.py)

python

CopyEdit

```
import faiss
```

```
import numpy as np
```

```
class VectorStore:
```

```

def __init__(self, dimension):
    self.dimension = dimension
    self.index = faiss.IndexFlatL2(self.dimension)
    self.stored_embeddings = []
    self.text_chunks = []

def add(self, embedding, chunk):
    self.stored_embeddings.append(embedding)
    self.text_chunks.append(chunk)

def build(self):
    vectors = np.array(self.stored_embeddings, dtype='float32')
    self.index.add(vectors)

def search(self, query_embedding, k=5):
    query_vector = np.array([query_embedding], dtype='float32')
    distances, indices = self.index.search(query_vector, k)
    results = [self.text_chunks[i] for i in indices[0]]
    return results

```

## 4.6 RAG Pipeline Orchestration (src/rag\_pipeline.py)

python

CopyEdit

```

from src.chunking import extract_text_from_pdf, chunk_text
from src.embeddings import get_embedding
from src.retrieval import VectorStore
from src.llm_inference import generate_answer

def process_pdf(pdf_path, vector_store):
    text = extract_text_from_pdf(pdf_path)
    chunks = chunk_text(text, chunk_size=500)
    for chunk in chunks:
        emb = get_embedding(chunk)
        vector_store.add(emb, chunk)
    vector_store.build()

def rag_query(query, vector_store):
    query_emb = get_embedding(query)

```

```
relevant_chunks = vector_store.search(query_emb, k=5)
return generate_answer(query, relevant_chunks)
```

#### 4.7 LLM Inference (src/llm\_inference.py)

python

CopyEdit

```
import openai
```

```
def generate_answer(question, context_chunks, model="gpt-3.5-turbo"):
    prompt = f"Use the following context to answer the question accurately.\n\nContext:\n{chr(10).join(context_chunks)}\n\nQuestion: {question}\nAnswer:"
    response = openai.ChatCompletion.create(
        model=model,
        messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message["content"]
```

*(Adjust to your local or alternative cloud-based LLM as needed.)*

#### 4.8 Streamlit Front-End (app.py)

python

CopyEdit

```
import streamlit as st
```

```
from src.rag_pipeline import process_pdf, rag_query
```

```
from src.retrieval import VectorStore
```

```
DIMENSION = 1536 # For text-embedding-ada-002, adjust if using
another model
```

```
vector_store = VectorStore(DIMENSION)
```

```
def main():
```

```
    st.title("IntelliDocs")
```

```
    st.subheader("Empower your documents with AI-driven insights")
```

```
    uploaded_pdf = st.file_uploader("Upload a PDF", type=["pdf"])
```

```
    if uploaded_pdf:
```

```

# Save temp file
pdf_path = "temp.pdf"
with open(pdf_path, "wb") as f:
    f.write(uploaded_pdf.read())

st.write("Processing your document...")
process_pdf(pdf_path, vector_store)
st.success("Document indexed! Ask any question below.")

user_query = st.text_input("Ask something about the document:")
if user_query:
    with st.spinner("Generating answer..."):
        answer = rag_query(user_query, vector_store)
        st.markdown(f"**Answer:** {answer}")

if __name__ == "__main__":
    main()

```

## 5. Testing & Usage

### 1. Local Execution

- `streamlit run app.py`
- Open the browser at <http://localhost:8501>.
- Upload a PDF, wait for indexing, then enter a question.

### 2. Scenarios & Edge Cases

- **Small PDFs:** Quick to index, minimal chunking.
- **Large PDFs:** Might require chunk overlap or a more robust chunking strategy.
- **Scanned PDFs:** Requires additional OCR if text is not inherently extractable.

## 6. Deployment

### 6.1 Docker Container (Optional)

dockerfile

CopyEdit

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt



```
COPY . .  
EXPOSE 8501  
CMD ["streamlit", "run", "app.py", "--server.port=8501",  
"--server.address=0.0.0.0"]
```

#### Build & Run:

bash

CopyEdit

```
docker build -t intellidocs:latest .  
docker run -p 8501:8501 intellidocs:latest
```

•

## 6.2 Cloud Platforms

- **Streamlit Community Cloud:** Easiest if using a public GitHub repo.
- **AWS / GCP / Azure / Heroku:** Connect your repo, configure environment variables (e.g., API keys), and deploy with the recommended specs.

## 7. Potential Enhancements

1. **Multi-Document Indexing**
  - Store and manage multiple PDFs at once, enabling cross-document queries.
2. **OCR Integration**
  - Use Tesseract or another OCR library for scanned PDFs.
3. **Advanced Prompt Engineering**
  - Provide multiple summary styles or question-answer formats.
4. **Fine-Tuning or Custom LLMs**
  - Domain-specific training for legal, medical, or financial documents.
5. **User Authentication & Permissions**
  - Protect sensitive documents and limit access to authorized users.
6. **Analytics & Monitoring**
  - Track usage patterns, model performance, and user feedback.

## 8. Conclusion

With IntelliDocs, you have a **complete blueprint** for an AI-driven application that ingests PDF documents, transforms them into searchable chunks, and provides **accurate, context-based answers** in response to user queries. By combining the power of modern LLMs with an efficient RAG pipeline, IntelliDocs can handle diverse documents—from academic research to policy memos—at scale.

Feel free to clone and enhance this repository to fit your own requirements. Whether you are building an internal research assistant or a public-facing knowledge search engine, IntelliDocs demonstrates how straightforward it can be to harness AI for real-time document analysis.

## Happy Innovating!

### You said:

can you create me read me file as well

Reasoned about a complete README file for a second

Below is a **sample README.md** file you can include in your GitHub repository. It provides an overview, installation instructions, usage steps, and suggestions for customization.

---

# IntelliDocs

**IntelliDocs** is an end-to-end AI-powered document exploration and summarization platform. It uses a **Retrieval-Augmented Generation (RAG)** approach with Large Language Models (LLMs) to help you ask questions about PDF documents and receive contextually grounded, concise answers.

## Table of Contents

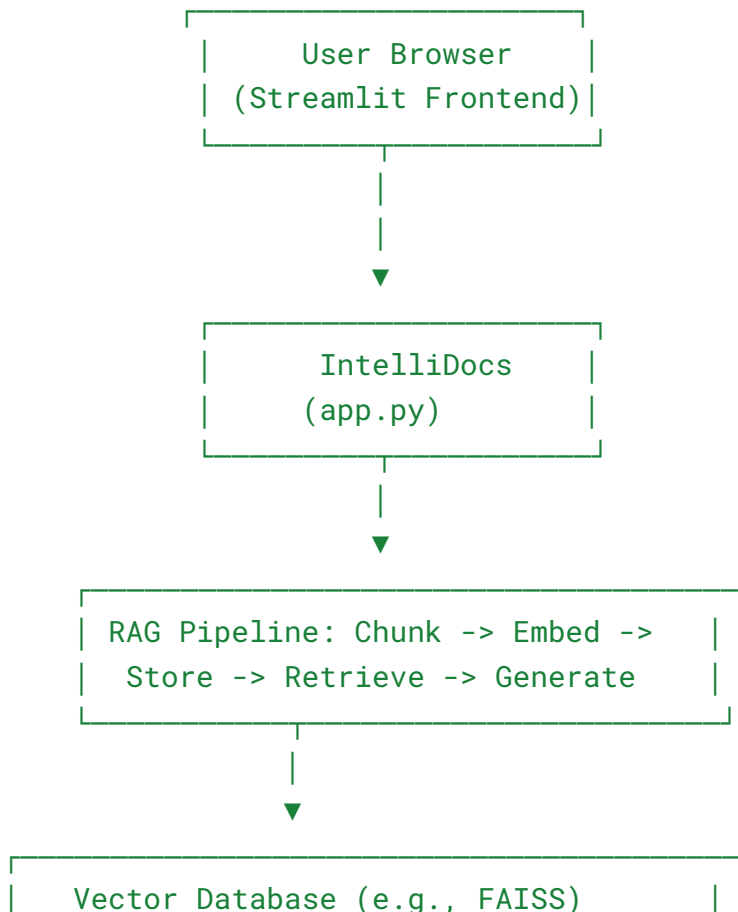
- Features
  - Architecture
  - Installation
    - Prerequisites
    - Setup Steps
  - Usage
    - Local Execution
    - Docker Execution
  - Project Structure
  - How It Works
    - Document Processing and Chunking
    - Embeddings and Vector Storage
    - Query & Answer Generation
  - Customization
  - Roadmap
  - License
-

## Features

- **Simple Upload & Query:** Drag-and-drop PDFs into a Streamlit UI and start asking questions.
  - **Efficient Text Chunking:** Splits long PDFs into smaller, manageable sections.
  - **Semantic Search:** Stores chunk embeddings in a vector database for quick, relevant retrieval.
  - **LLM-Powered Answers:** Answers are generated by state-of-the-art language models, grounded in retrieved chunks to reduce hallucinations.
  - **Modular & Scalable:** Easily swap out components like the vector database (FAISS, Milvus, Chroma) or the LLM (DeepSeek-R1:1.5b, Ollama, OpenAI, etc.).
  - **Developer-Friendly:** Clear folder structure and code organization to encourage experimentation and extension.
- 

## Architecture

rust  
CopyEdit



---

# Installation

## Prerequisites

- Python 3.8 or higher
- [pip](#)
- (Optional) [Docker](#) for containerized deployments

## Setup Steps

**Clone this repository:**

bash

CopyEdit

```
git clone https://github.com/yourusername/IntelliDocs.git
cd IntelliDocs
```

1.

**Install Python dependencies:**

bash

CopyEdit

```
pip install -r requirements.txt
```

2. Note: If you plan to use cloud-based LLM or embedding services (like OpenAI), ensure you add and configure those dependencies (e.g., [openai](#)).

**Configure environment variables** (if using API-based embeddings or LLMs). For example:

bash

CopyEdit

```
export OPENAI_API_KEY="your_openai_key_here"
```

3. Or create a [.env](#) file and load it in your code as needed.

---

# Usage

## Local Execution

### Start the Streamlit app:

bash

CopyEdit

```
streamlit run app.py
```

- 1.
2. **Open your browser** at the URL provided in the console (usually <http://localhost:8501>).
3. **Upload a PDF** file. Once the document finishes processing, type your questions into the text input box.

## Docker Execution

### Build the Docker image:

bash

CopyEdit

```
docker build -t intellidocs:latest .
```

- 1.

### Run the Docker container:

bash

CopyEdit

```
docker run -p 8501:8501 intellidocs:latest
```

- 2.
3. **Access** at <http://localhost:8501> on your host machine.

---

## Project Structure

graphql

CopyEdit

IntelliDocs/

```
|— README.md           # You're here!
|— requirements.txt
|— Dockerfile          # For optional containerization
|— app.py              # Streamlit entry point
|
|— src/
|   |— chunking.py      # PDF parsing & text chunking
|   |— embeddings.py    # Generating & managing vector embeddings
|   |— retrieval.py     # Searching the vector store
|   |— rag_pipeline.py  # High-level RAG orchestration
```

```
|   └─ llm_inference.py    # Functions for local/cloud LLM inference
|
| └─ data/
|   └─ sample.pdf          # Example PDF for testing
|
| └─ docs/
|   └─ architecture.png    # Architecture diagrams or references
```

---

## How It Works

### Document Processing and Chunking

1. **Extract Text:** The PDF is parsed using [pdfplumber](#) or [PyPDF2](#).
2. **Chunking:** Large text is split into smaller segments for efficient embedding and retrieval (usually around 500 tokens or words per chunk).

### Embeddings and Vector Storage

- Each chunk is converted into a *vector embedding* using a model of your choice (e.g., OpenAI's `text-embedding-ada-002` or a local model).
- Embeddings are stored in a **vector database** (e.g., FAISS). At query time, the system performs a nearest-neighbor search to quickly find the most relevant chunks.

### Query & Answer Generation

1. **User Query:** Converts the user's question into an embedding.
  2. **Nearest Neighbors:** Retrieves the top-k most relevant chunks.
  3. **Answer Composition:** Passes those chunks as context to an LLM (e.g., GPT-3.5, a local model, etc.). The LLM then generates a concise, context-aware response.
- 

## Customization

1. **Local vs. Cloud Models:** Swap out OpenAI embeddings for a local embedding model, or switch the LLM from GPT-3.5 to another provider.
2. **Vector Database:** Choose from [FAISS](#), [Milvus](#), or [Chroma](#).
3. **UI Adjustments:** Tailor the Streamlit interface to your workflow (e.g., add fields for specifying the number of chunks to retrieve, or an option to produce longer or shorter summaries).

4. **Advanced NLP:** Implement additional features like Named Entity Recognition (NER), sentiment analysis, or PDF metadata extraction.
- 

## Roadmap

- **Multi-Document Management:** Index multiple PDFs simultaneously for cross-document queries.
  - **OCR Support:** Integrate an OCR pipeline (e.g., Tesseract) for scanned or image-based PDFs.
  - **Fine-Tuning:** Explore domain-specific fine-tuning for specialized use cases (e.g., legal or medical texts).
  - **Advanced Caching:** Store partial query results to boost performance for repeated queries.
  - **User Authentication:** Add secure user management for private or sensitive document handling.
- 

## License

MIT License – Feel free to modify, distribute, and build upon this project for both commercial and non-commercial purposes. See LICENSE for details.

---

## Enjoy IntelliDocs!

For any questions or contributions, please open an Issue or Pull Request. We welcome feedback and collaboration to make IntelliDocs even more powerful and developer-friendly.