PROGRESS

1. GPT-Neo 125M

Pros

- Open-source and freely available for research and development.
- Lightweight model, fast for inference and experimentation.
- Suitable for smaller tasks or prototyping.
- Fine-tuning possible for specific use cases.

Cons

- Limited performance due to the small number of parameters (125M).
- Lacks the ability to handle complex queries or tasks compared to larger models.
- Lower accuracy and coherence for long-form or detailed text generation.

---

2. GPT-2

Pros

- Freely available and widely used in research.
- Proven effectiveness for various NLP tasks like summarization, text generation, and more.
- Scalable with multiple parameter sizes (small, medium, large).
- Good documentation and community support.

Cons

- Outdated compared to more modern models like GPT-3 or GPT-4.
- Lacks fine-tuned versions for instruction-based tasks or chat-specific purposes.
- Requires significant computational resources for larger parameter sizes.

---

3. DialoGPT

Pros

- Optimized for conversational tasks; designed specifically for dialogue generation.
- Open-source and available for custom fine-tuning.
- Pre-trained on conversational datasets, making it more effective for chatbots and dialogue applications.

Cons

- Limited general-purpose capabilities outside conversational tasks.
- Performance varies significantly depending on the dialogue complexity.
- Outperformed by newer models like ChatGPT and other conversational LLMs.

---

4. Milvus

Pros

- An open-source vector database for similarity search and embedding storage.
- Optimized for managing and querying large-scale embeddings, such as those from language models.
- Can be integrated with many LLMs like GPT-Neo, GPT-J, or GPT-2 for efficient retrieval-based applications.
- Excellent scalability and performance for retrieval-based tasks.

Cons

- Not an LLM; requires external models for generating or managing embeddings.
- Complexity in setup and integration for non-expert users.
- Requires significant computational resources for large datasets.

---

5. GPT-J

Pros

- Open-source and comparable to GPT-3 in terms of architecture.
- Larger model (6B parameters) than GPT-Neo 125M, providing better accuracy and text coherence.
- Suitable for many NLP tasks, including summarization, question-answering, and more.
- Supports fine-tuning for custom applications.

Cons

- High computational requirements for training and inference.
- Limited support and pre-trained datasets compared to proprietary models like GPT-3 or GPT-4.
- Lacks conversational fine-tuning by default.

---

6. GPT-Neo (Varied Sizes)

Pros

- Open-source with multiple sizes (e.g., 125M, 1.3B, 2.7B).
- Good balance of performance and accessibility for medium and large parameter models.
- Widely used for text generation, summarization, and other NLP tasks.
- Community-driven development and documentation.

Cons

- Performance depends on the model size (smaller models struggle with complex tasks).
- Requires more computational resources than smaller models like GPT-2 or GPT-Neo 125M.
- No native instruction-following fine-tuning like proprietary GPT-3 models.

Summary

Model/Tool	**Key Strengths**	**Key Weaknesses**
GPT-Neo 125M	Lightweight, fast, open-source	Limited capabilities, low accuracy
GPT-2	Versatile, widely supported	Outdated compared to newer models
DialoGPT	Optimized for dialogue	Limited for non-conversational tasks
Milvus	Scalable vector storage	Requires integration with LLMs
GPT-J	Powerful, open-source alternative	High computational requirements
GPT-Neo	Variety of sizes, open-source	Dependent on model size for tasks

Code used for summarization

```
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
from haystack.document_stores import InMemoryDocumentStore
from haystack.nodes import BM25Retriever
from haystack.schema import Document
import fitz  # PyMuPDF
import os
import re
from transformers import pipeline  # Hugging Face for summarization
```

```
# Initialize Flask App
app = Flask(__name__)
CORS(app)

# Directory to save uploaded files
UPLOAD_FOLDER = "data/uploads/"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Initialize the document store
document_store = InMemoryDocumentStore(use_bm25=True)
retriever = BM25Retriever(document_store=document_store)

# Load the summarization pipeline (using a free model)
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")  # BART model for summarization

# Status flag for document processing
document_initialized = False


def extract_text_from_pdf(pdf_path):
    """Extracts text from each page of the PDF and returns a list of pages."""
    doc = fitz.open(pdf_path)
    pages = []
    for page_num, page in enumerate(doc, start=1):
        pages.append({"page_number": page_num, "content": page.get_text()})
    return pages


def index_document(pages):
    """Indexes the document into the document store."""
    documents = [
        Document(content=page["content"], meta={"page_number": page["page_number"]})
        for page in pages
    ]
    document_store.write_documents(documents)


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


@app.route('/initialize', methods=['POST'])
def initialize():
    global document_initialized
    try:
        if 'file' not in request.files:
            return jsonify({"error": "No file uploaded."}), 400

        file = request.files['file']
        if file.filename == '':
            return jsonify({"error": "No selected file."}), 400

        if not file.filename.endswith('.pdf'):
            return jsonify({"error": "Only PDF files are allowed."}), 400

        file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(file_path)

        # Process the PDF and index it
        pages = extract_text_from_pdf(file_path)
        index_document(pages)
        document_initialized = True
        return jsonify({"message": "Document uploaded and processed successfully!"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500



@app.route('/ask', methods=['GET'])
def ask():
    if not document_initialized:
        return jsonify({"error": "No document has been uploaded or processed yet."}), 400

    query = request.args.get('query')
    if not query:
        return jsonify({"error": "No query provided."}), 400

    try:
        results = retriever.retrieve(query, top_k=3)
        print("results:",results)
        if results:
            # Combine content from retrieved results
            combined_content = "\n".join([result.content for result in results])



            # Use summarizer to process the retrieved content into a user-friendly answer
            augmented_answer = summarizer(
                combined_content,
                max_length=200,
                min_length=50,
                do_sample=False
            )[0]["summary_text"]

            return jsonify({"answer": augmented_answer})
        else:
            return jsonify({"answer": "No relevant information found."})
    except Exception as e:
        return jsonify({"error": str(e)}), 500


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

GOOD CODE WITH VECTOR EMBEDDINGS

```
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
from haystack.document_stores import FAISSDocumentStore
from haystack.nodes import EmbeddingRetriever
from haystack.pipelines import ExtractiveQAPipeline
from transformers import pipeline as hf_pipeline
import fitz  # PyMuPDF
import os
import logging
import time

# Initialize Flask App
app = Flask(__name__)
CORS(app)

# Directory to save uploaded files
UPLOAD_FOLDER = "data/uploads/"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
```

```
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def safe_remove(file_path, retries=3, delay=1):
    """Attempt to remove a file with retries."""
    for _ in range(retries):
        try:
            if os.path.exists(file_path):
                os.remove(file_path)
                print(f"{file_path} removed successfully.")
                return
        except PermissionError as e:
            print(f"PermissionError: {e}. Retrying in {delay} seconds...")
            time.sleep(delay)
    print(f"Failed to remove {file_path} after {retries} retries.")

# Ensure file cleanup
safe_remove("faiss_document_store.db")
safe_remove("faiss_index")

# Initialize FAISS Document Store with correct embedding dimensions
try:
    document_store = FAISSDocumentStore(
        sql_url="sqlite:///faiss_document_store.db",
        faiss_index_factory_str="Flat",
        embedding_dim=384  # Match the embedding model dimension
    )
    print("FAISS Document Store initialized successfully.")
except Exception as e:
    logging.error(f"Error initializing FAISS Document Store: {str(e)}")
    raise
```

```
# Initialize Embedding Retriever
retriever = EmbeddingRetriever(
    document_store=document_store,
    embedding_model="sentence-transformers/all-MiniLM-L6-v2",
    use_gpu=True
)

# Update embeddings only if documents are present
if document_store.get_document_count() > 0:
    document_store.update_embeddings(retriever)

# Initialize Hugging Face QA pipeline with accessible model
qa_pipeline = hf_pipeline("question-answering", model="deepset/roberta-base-squad2")

# Status flag for document processing
document_initialized = False

def extract_text_from_pdf(pdf_path):
    """Extract text from a PDF and return chunks for indexing."""
    doc = fitz.open(pdf_path)
    chunks = []
    for page_num, page in enumerate(doc, start=1):
        content = page.get_text()
        chunks.append({
            "content": content,
            "meta": {"page_number": page_num, "source": pdf_path}
        })
    return chunks

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


@app.route('/initialize', methods=['POST'])
def initialize():
    global document_initialized
    try:
        if 'file' not in request.files:
            return jsonify({"error": "No file uploaded."}), 400

        file = request.files['file']
        if file.filename == '':
            return jsonify({"error": "No selected file."}), 400

        if not file.filename.endswith('.pdf'):
            return jsonify({"error": "Only PDF files are allowed."}), 400

        file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(file_path)

        # Extract text and index it
        chunks = extract_text_from_pdf(file_path)
        documents = [
            {"content": chunk["content"], "meta": chunk["meta"]}
            for chunk in chunks
        ]
        document_store.write_documents(documents)

        # Update embeddings in FAISS
        document_store.update_embeddings(retriever)
        document_initialized = True
        return jsonify({"message": "Document uploaded and processed successfully!"}), 200
    except Exception as e:
        logging.error(f"Error during initialization: {str(e)}")
        return jsonify({"error": str(e)}), 500

@app.route('/ask', methods=['GET'])
def ask():
    if not document_initialized:
        return jsonify({"error": "No document has been uploaded or processed yet."}), 400

    query = request.args.get('query')
    if not query:
        return jsonify({"error": "No query provided."}), 400

    try:
        # Retrieve relevant documents
        retrieved_docs = retriever.retrieve(query=query, top_k=3)
        context = " ".join([doc.content for doc in retrieved_docs])

        # Use QA pipeline for answering
        result = qa_pipeline(question=query, context=context)
        answer = result["answer"] if result else "No relevant information found."

        return jsonify({"answer": answer}), 200
    except Exception as e:
        logging.error(f"Error during question answering: {str(e)}")
        return jsonify({"error": str(e)}), 500

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

Code using LALMA:

```
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
from haystack.document_stores import FAISSDocumentStore
from haystack.nodes import EmbeddingRetriever
import fitz  # PyMuPDF
import os
import logging
from transformers import AutoModelForCausalLM, AutoTokenizer

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = Flask(__name__)
CORS(app)

UPLOAD_FOLDER = "data/uploads/"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

# Initialize FAISS Document Store
document_store = FAISSDocumentStore(
    sql_url="sqlite:///faiss_document_store.db",
    faiss_index_factory_str="Flat",
    embedding_dim=384,
)

# Initialize Retriever
retriever = EmbeddingRetriever(
    document_store=document_store,
    embedding_model="sentence-transformers/all-MiniLM-L6-v2",
    use_gpu=False,
    max_seq_len=256,
)

# Load LLaMA 2
llama_model = AutoModelForCausalLM.from_pretrained(
    "meta-llama/Llama-2-7b-chat-hf", device_map="auto", torch_dtype="float16"
)
llama_tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-chat-hf")

def llama_answer(query, context):
    """Use LLaMA 2 to generate an answer based on the query and context."""
    prompt = f"Context: {context}\n\nQuestion: {query}\nAnswer:"
    inputs = llama_tokenizer(prompt, return_tensors="pt").to("cuda")
    outputs = llama_model.generate(inputs["input_ids"], max_new_tokens=200, temperature=0.7)
    return llama_tokenizer.decode(outputs[0], skip_special_tokens=True)

def extract_text_from_pdf(pdf_path):
    """Extract text from a PDF and return chunks for indexing."""
    doc = fitz.open(pdf_path)
    chunks = []
    for page_num, page in enumerate(doc, start=1):
        content = page.get_text("text")
        if content.strip():  # Skip empty pages
            chunks.append({
                "content": content,
                "meta": {"page_number": page_num, "source": os.path.basename(pdf_path)},
            })
    return chunks
```

```
@app.route("/")
def home():
    return render_template("index.html")


@app.route("/initialize", methods=["POST"])
def initialize():
    try:
        # Ensure a file is uploaded
        if "file" not in request.files:
            return jsonify({"message": "No file uploaded."}), 400

        file = request.files["file"]
        if not file.filename.endswith(".pdf"):
            return jsonify({"message": "Only PDF files are allowed."}), 400

        # Save the uploaded PDF
        file_path = os.path.join(app.config["UPLOAD_FOLDER"], file.filename)
        file.save(file_path)

        # Extract text from PDF and prepare documents
        chunks = extract_text_from_pdf(file_path)
        documents = [
            {"content": chunk["content"], "meta": chunk["meta"]}
            for chunk in chunks
        ]

        # Clear existing data to avoid conflicts
        document_store.delete_documents()
        document_store.faiss_index.reset()

        # Add documents to the store and update embeddings
        document_store.write_documents(documents)
        document_store.update_embeddings(retriever)

        return jsonify({"message": "Document processed and indexed successfully!"}), 200
    except Exception as e:
        logger.error(f"Error during initialization: {e}")
        return jsonify({"message": "An error occurred.", "error": str(e)}), 500

@app.route("/ask", methods=["GET"])
def ask():
    query = request.args.get("query", "").strip().lower()
    if not query:
        return jsonify({"message": "Query parameter is missing."}), 400

    try:
        # Retrieve top 5 relevant documents
        retrieved_docs = retriever.retrieve(query, top_k=5)
        context = " ".join([doc.content for doc in retrieved_docs])

        # Generate response using LLaMA 2
        answer = llama_answer(query, context)

        return jsonify({"query": query, "answer": answer}), 200
    except Exception as e:
        logger.error(f"Error during query: {e}")
        return jsonify({"message": "An error occurred.", "error": str(e)}), 500

@app.route("/validate", methods=["GET"])
def validate():
    """Validate that the FAISS index and SQL database are in sync."""
    try:
        num_documents = len(document_store.get_all_documents())
        num_embeddings = document_store.faiss_index.ntotal

        if num_documents != num_embeddings:
            return jsonify({
                "message": "The FAISS index and SQL database are not synchronized.",
                "num_documents": num_documents,
                "num_embeddings": num_embeddings,
            }), 400

        return jsonify({
            "message": "FAISS index and SQL database are synchronized.",
            "num_documents": num_documents,
            "num_embeddings": num_embeddings,
        }), 200
    except Exception as e:
        logger.error(f"Error during validation: {e}")
        return jsonify({"message": "An error occurred.", "error": str(e)}), 500

if __name__ == "__main__":
    # Validate FAISS and SQL synchronization on startup
    try:
        num_documents = len(document_store.get_all_documents())
        num_embeddings = document_store.faiss_index.ntotal

        if num_documents != num_embeddings:
            logger.warning(
                f"The number of documents ({num_documents}) does not match "
                f"the number of embeddings ({num_embeddings}). Rebuilding FAISS index..."
            )
            document_store.update_embeddings(retriever)
    except Exception as e:
        logger.error(f"Error during startup validation: {e}")

    app.run(debug=True)

# Steps to Fix the Issue
# 1. Check the FAISS Index Initialization
#    Ensure that the FAISS index is properly created and updated with embeddings. You can do this by running the following code snippet after writing the documents to the document_store:
#    ```python
#    # After writing documents to the document store
#    document_store.update_embeddings(retriever)
#    ```
#    This step ensures that embeddings are generated for the documents and added to the FAISS index.

# 2. Clear the SQL Database and Recreate the Index
#    If the issue persists, it might be because old documents or embeddings are causing conflicts. You can clear the database and start fresh:
#    ```python
#    # Delete all documents and recreate the index
#    document_store.delete_documents()
#    document_store.faiss_index.reset()  # Reset FAISS index
#    ```
#    After clearing, reprocess your PDF file to re-add the documents and update the embeddings.

# 3. Ensure FAISS and SQL Configurations Are Consistent
#    Double-check that the FAISSDocumentStore is configured correctly. The sql_url and FAISS index must point to the same database and files.
#    ```python
#    document_store = FAISSDocumentStore(
#      sql_url="sqlite:///faiss_document_store.db",  # Ensure the database matches the one used earlier
#        faiss_index_factory_str="Flat",
```

```
#       embedding_dim=384,  # Ensure the embedding dimension matches the retriever model
#    )
#    ```

# 4. Rebuild the FAISS Index
#    If the FAISS index file is corrupted or missing, you can rebuild it by re-indexing the documents:
#    ```python
#    document_store.update_embeddings(retriever)  # Rebuild FAISS index with fresh embeddings
#    ```

# 5. Validate Synchronization
#    After running the above steps, validate that the FAISS index is synchronized with the SQL database:
#    ```python
#    num_documents = len(document_store.get_all_documents())
#    num_embeddings = document_store.faiss_index.ntotal

#    if num_documents != num_embeddings:
#       raise ValueError(
#           f"The number of documents ({num_documents}) does not match the number of embeddings
({num_embeddings})."
#       )
#    ```
```

CODE using deepset

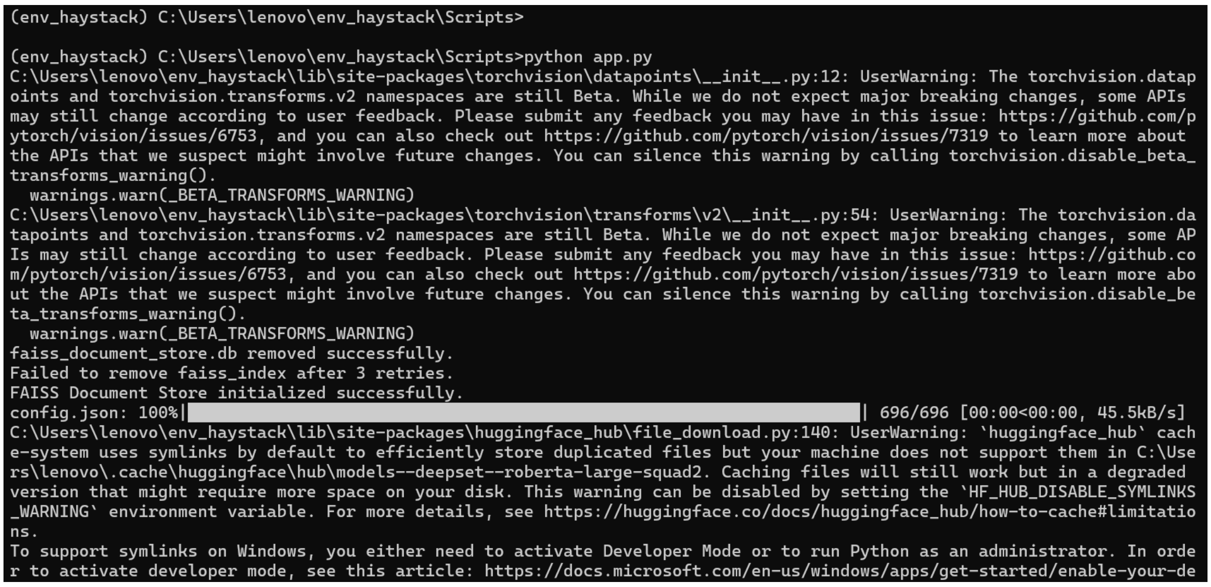
```
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
from haystack.document_stores import FAISSDocumentStore
from haystack.nodes import EmbeddingRetriever
from haystack.pipelines import ExtractiveQAPipeline
from transformers import pipeline as hf_pipeline
import fitz  # PyMuPDF
```

```
import os

import logging

import time


# Initialize Flask App

app = Flask(__name__)

CORS(app)


# Directory to save uploaded files

UPLOAD_FOLDER = "data/uploads/"

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


def safe_remove(file_path, retries=3, delay=1):

    """Attempt to remove a file with retries."""

    for _ in range(retries):

        try:

            if os.path.exists(file_path):

                os.remove(file_path)

                print(f"{file_path} removed successfully.")

                return

        except PermissionError as e:

            print(f"PermissionError: {e}. Retrying in {delay} seconds...")

            time.sleep(delay)

    print(f"Failed to remove {file_path} after {retries} retries.")


# Ensure file cleanup

safe_remove("faiss_document_store.db")

safe_remove("faiss_index")


# Initialize FAISS Document Store with correct embedding dimensions
```

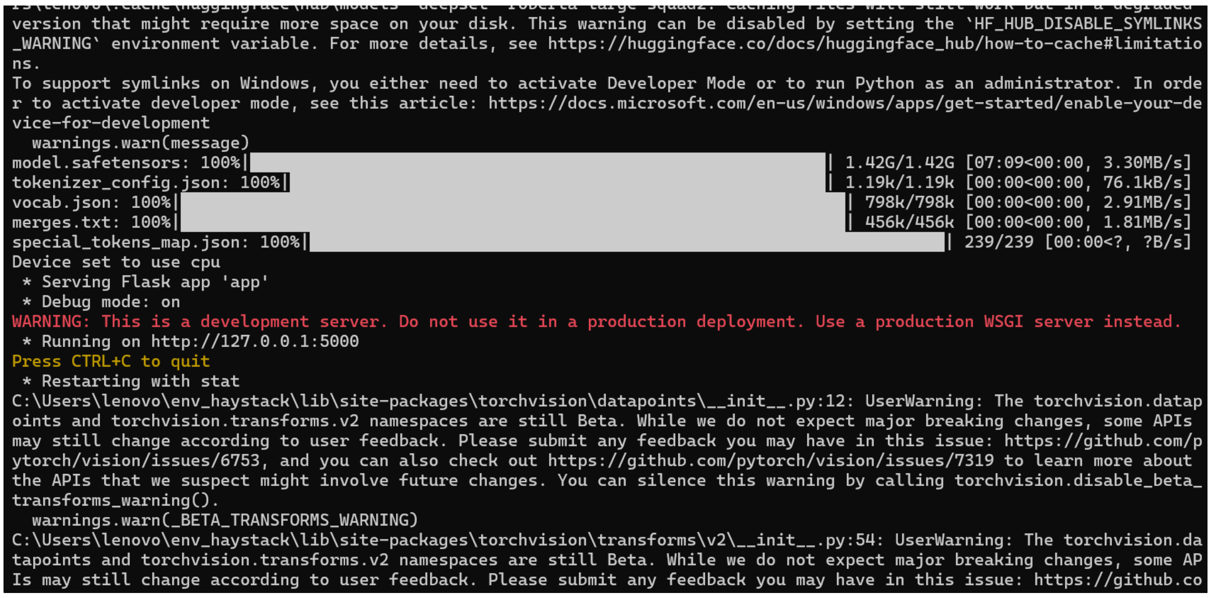
```
try:
    document_store = FAISSDocumentStore(
        sql_url="sqlite:///faiss_document_store.db",
        faiss_index_factory_str="Flat",
        embedding_dim=384  # Match the embedding model dimension
    )
    print("FAISS Document Store initialized successfully.")
except Exception as e:
    logging.error(f"Error initializing FAISS Document Store: {str(e)}")
    raise

# Initialize Embedding Retriever
retriever = EmbeddingRetriever(
    document_store=document_store,
    embedding_model="sentence-transformers/all-MiniLM-L6-v2",
    use_gpu=True
)

# Update embeddings only if documents are present
if document_store.get_document_count() > 0:
    document_store.update_embeddings(retriever)

# Initialize Hugging Face QA pipeline with accessible model
qa_pipeline = hf_pipeline(
    "question-answering",
    model="deepset/roberta-large-squad2",
    tokenizer="deepset/roberta-large-squad2",
    framework="pt"
)

# Status flag for document processing
```

```
document_initialized = False


def extract_text_from_pdf(pdf_path):
    """Extract text from a PDF and return chunks for indexing."""
    doc = fitz.open(pdf_path)
    chunks = []
    for page_num, page in enumerate(doc, start=1):
        content = page.get_text()
        chunks.append({
            "content": content,
            "meta": {"page_number": page_num, "source": pdf_path}
        })
    return chunks


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


@app.route('/initialize', methods=['POST'])
def initialize():
    global document_initialized
    try:
        if 'file' not in request.files:
            return jsonify({"error": "No file uploaded."}), 400

        file = request.files['file']
        if file.filename == '':
            return jsonify({"error": "No selected file."}), 400

        if not file.filename.endswith('.pdf'):
            return jsonify({"error": "Only PDF files are allowed."}), 400
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(file_path)

        # Extract text and index it
        chunks = extract_text_from_pdf(file_path)
        documents = [
            {"content": chunk["content"], "meta": chunk["meta"]}
            for chunk in chunks
        ]
        document_store.write_documents(documents)

        # Update embeddings in FAISS
        document_store.update_embeddings(retriever)
        document_initialized = True

        return jsonify({"message": "Document uploaded and processed successfully!"}), 200
    except Exception as e:
        logging.error(f"Error during initialization: {str(e)}")
        return jsonify({"error": str(e)}), 500

@app.route('/ask', methods=['GET'])
def ask():
    if not document_initialized:
        return jsonify({"error": "No document has been uploaded or processed yet."}), 400

    query = request.args.get('query')
    if not query:
        return jsonify({"error": "No query provided."}), 400

    try:
        # Retrieve relevant documents
        retrieved_docs = retriever.retrieve(query=query, top_k=5)  # Adjust top_k as needed
        contexts = [doc.content for doc in retrieved_docs]

        # Debugging: Log the context being passed to the QA pipeline
        logging.info(f"Contexts for query '{query}': {contexts}")

        # Process each chunk with the QA pipeline and aggregate answers
        answers = []
        for context in contexts:
            result = qa_pipeline(question=query, context=context)
            answers.append(result["answer"] if result else "")

        # Combine all answers into one response
        combined_answer = " ".join(answers)

        return jsonify({"query": query, "answer": combined_answer}), 200
    except Exception as e:
        logging.error(f"Error during question answering: {str(e)}")
        return jsonify({"error": str(e)}), 500

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

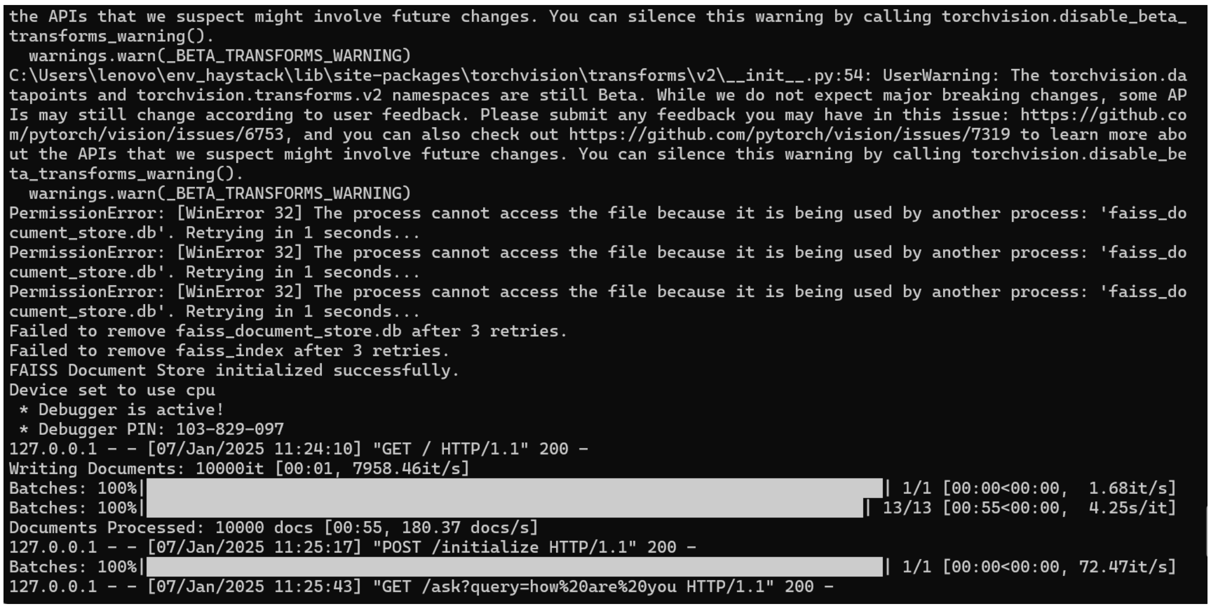
OUTPUT:



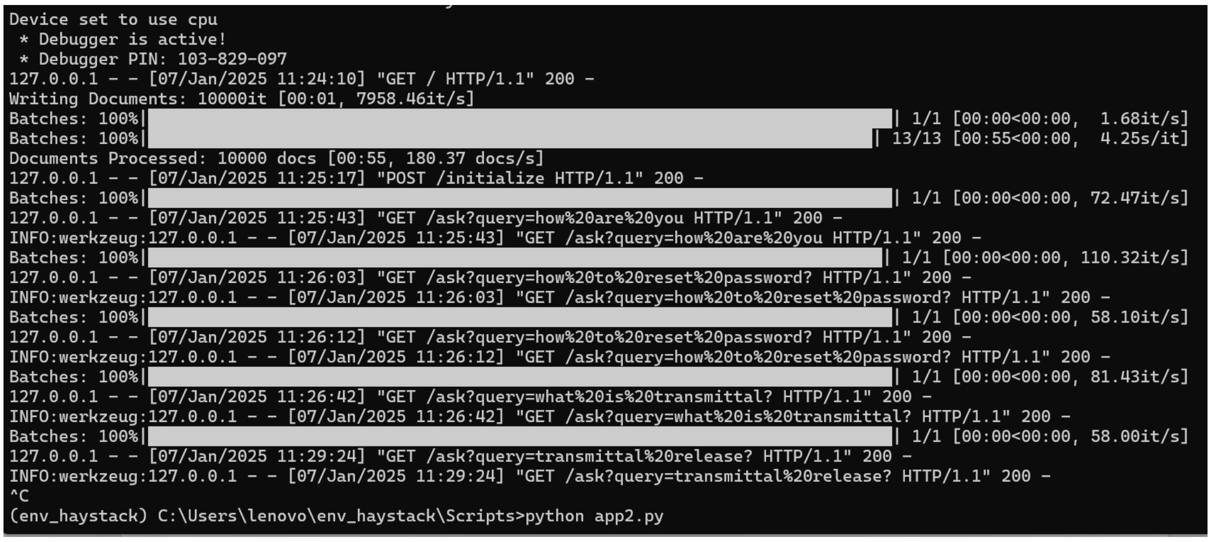
```
(env_haystack) C:\Users\lenovo\env_haystack\Scripts>

(env_haystack) C:\Users\lenovo\env_haystack\Scripts>python app.py
C:\Users\lenovo\env_haystack\lib\site-packages\torchvision\datapoints\__init__.py:12: UserWarning: The torchvision.datap
oints and torchvision.transforms.v2 namespaces are still Beta. While we do not expect major breaking changes, some APIs
may still change according to user feedback. Please submit any feedback you may have in this issue: https://github.com/p
ytorch/vision/issues/6753, and you can also check out https://github.com/pytorch/vision/issues/7319 to learn more about
the APIs that we suspect might involve future changes. You can silence this warning by calling torchvision.disable_beta_
transforms_warning().
  warnings.warn(_BETA_TRANSFORMS_WARNING)
C:\Users\lenovo\env_haystack\lib\site-packages\torchvision\transforms\v2\__init__.py:54: UserWarning: The torchvision.da
tapoints and torchvision.transforms.v2 namespaces are still Beta. While we do not expect major breaking changes, some AP
Is may still change according to user feedback. Please submit any feedback you may have in this issue: https://github.co
m/pytorch/vision/issues/6753, and you can also check out https://github.com/pytorch/vision/issues/7319 to learn more abo
ut the APIs that we suspect might involve future changes. You can silence this warning by calling torchvision.disable_be
ta_transforms_warning().
  warnings.warn(_BETA_TRANSFORMS_WARNING)
faiss_document_store.db removed successfully.
Failed to remove faiss_index after 3 retries.
FAISS Document Store initialized successfully.
config.json: 100%|██████████████████████████████████████████████████████████| 696/696 [00:00<00:00, 45.5kB/s]
C:\Users\lenovo\env_haystack\lib\site-packages\huggingface_hub\file_download.py:140: UserWarning: `huggingface_hub` cach
e-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Use
rs\lenovo\.cache\huggingface\hub\models--deepset--roberta-large-squad2. Caching files will still work but in a degraded
version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS
_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitatio
ns.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In orde
r to activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-de
```



```
rs\lenovo\.cache\huggingface\hub\models--deepset--roberta-large-squad2. Caching files will still work but in a degraded
version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS
_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitatio
ns.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In orde
r to activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-de
vice-for-development
  warnings.warn(message)
model.safetensors: 100%|███████████████████████████████████████████████████| 1.42G/1.42G [07:09<00:00, 3.30MB/s]
tokenizer_config.json: 100%|███████████████████████████████████████████████| 1.19k/1.19k [00:00<00:00, 76.1kB/s]
vocab.json: 100%|██████████████████████████████████████████████████████████| 798k/798k [00:00<00:00, 2.91MB/s]
merges.txt: 100%|██████████████████████████████████████████████████████████| 456k/456k [00:00<00:00, 1.81MB/s]
special_tokens_map.json: 100%|█████████████████████████████████████████████████████| 239/239 [00:00<?, ?B/s]
Device set to use cpu
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
C:\Users\lenovo\env_haystack\lib\site-packages\torchvision\datapoints\__init__.py:12: UserWarning: The torchvision.datap
oints and torchvision.transforms.v2 namespaces are still Beta. While we do not expect major breaking changes, some APIs
may still change according to user feedback. Please submit any feedback you may have in this issue: https://github.com/p
ytorch/vision/issues/6753, and you can also check out https://github.com/pytorch/vision/issues/7319 to learn more about
the APIs that we suspect might involve future changes. You can silence this warning by calling torchvision.disable_beta_
transforms_warning().
  warnings.warn(_BETA_TRANSFORMS_WARNING)
C:\Users\lenovo\env_haystack\lib\site-packages\torchvision\transforms\v2\__init__.py:54: UserWarning: The torchvision.da
tapoints and torchvision.transforms.v2 namespaces are still Beta. While we do not expect major breaking changes, some AP
Is may still change according to user feedback. Please submit any feedback you may have in this issue: https://github.co
```

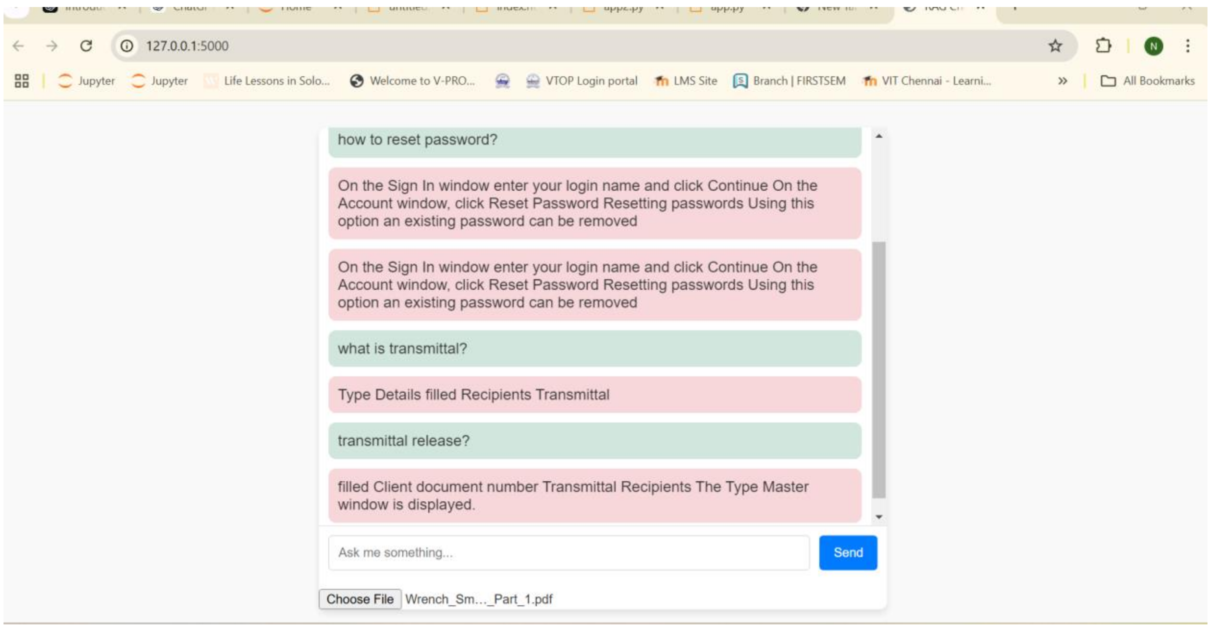


```
the APIs that we suspect might involve future changes. You can silence this warning by calling torchvision.disable_beta_
transforms_warning().
  warnings.warn(_BETA_TRANSFORMS_WARNING)
C:\Users\lenovo\env_haystack\lib\site-packages\torchvision\transforms\v2\__init__.py:54: UserWarning: The torchvision.da
tapoints and torchvision.transforms.v2 namespaces are still Beta. While we do not expect major breaking changes, some AP
Is may still change according to user feedback. Please submit any feedback you may have in this issue: https://github.co
m/pytorch/vision/issues/6753, and you can also check out https://github.com/pytorch/vision/issues/7319 to learn more abo
ut the APIs that we suspect might involve future changes. You can silence this warning by calling torchvision.disable_be
ta_transforms_warning().
  warnings.warn(_BETA_TRANSFORMS_WARNING)
PermissionError: [WinError 32] The process cannot access the file because it is being used by another process: 'faiss_do
cument_store.db'. Retrying in 1 seconds...
PermissionError: [WinError 32] The process cannot access the file because it is being used by another process: 'faiss_do
cument_store.db'. Retrying in 1 seconds...
PermissionError: [WinError 32] The process cannot access the file because it is being used by another process: 'faiss_do
cument_store.db'. Retrying in 1 seconds...
Failed to remove faiss_document_store.db after 3 retries.
Failed to remove faiss_index after 3 retries.
FAISS Document Store initialized successfully.
Device set to use cpu
 * Debugger is active!
 * Debugger PIN: 103-829-097
127.0.0.1 - - [07/Jan/2025 11:24:10] "GET / HTTP/1.1" 200 -
Writing Documents: 10000it [00:01, 7958.46it/s]
Batches: 100%|███████████████████████████████████████████████████████████| 1/1 [00:00<00:00,  1.68it/s]
Batches: 100%|█████████████████████████████████████████████████████████| 13/13 [00:55<00:00,  4.25s/it]
Documents Processed: 10000 docs [00:55, 180.37 docs/s]
127.0.0.1 - - [07/Jan/2025 11:25:17] "POST /initialize HTTP/1.1" 200 -
Batches: 100%|███████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 72.47it/s]
127.0.0.1 - - [07/Jan/2025 11:25:43] "GET /ask?query=how%20are%20you HTTP/1.1" 200 -
```



```
Device set to use cpu
 * Debugger is active!
 * Debugger PIN: 103-829-097
127.0.0.1 - - [07/Jan/2025 11:24:10] "GET / HTTP/1.1" 200 -
Writing Documents: 10000it [00:01, 7958.46it/s]
Batches: 100%|██████████████████████████████████████████████████████████████| 1/1 [00:00<00:00,  1.68it/s]
Batches: 100%|████████████████████████████████████████████████████████████| 13/13 [00:55<00:00,  4.25s/it]
Documents Processed: 10000 docs [00:55, 180.37 docs/s]
127.0.0.1 - - [07/Jan/2025 11:25:17] "POST /initialize HTTP/1.1" 200 -
Batches: 100%|██████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 72.47it/s]
127.0.0.1 - - [07/Jan/2025 11:25:43] "GET /ask?query=how%20are%20you HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [07/Jan/2025 11:25:43] "GET /ask?query=how%20are%20you HTTP/1.1" 200 -
Batches: 100%|█████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 110.32it/s]
127.0.0.1 - - [07/Jan/2025 11:26:03] "GET /ask?query=how%20to%20reset%20password? HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [07/Jan/2025 11:26:03] "GET /ask?query=how%20to%20reset%20password? HTTP/1.1" 200 -
Batches: 100%|██████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 58.10it/s]
127.0.0.1 - - [07/Jan/2025 11:26:12] "GET /ask?query=how%20to%20reset%20password? HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [07/Jan/2025 11:26:12] "GET /ask?query=how%20to%20reset%20password? HTTP/1.1" 200 -
Batches: 100%|██████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 81.43it/s]
127.0.0.1 - - [07/Jan/2025 11:26:42] "GET /ask?query=what%20is%20transmittal? HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [07/Jan/2025 11:26:42] "GET /ask?query=what%20is%20transmittal? HTTP/1.1" 200 -
Batches: 100%|██████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 58.00it/s]
127.0.0.1 - - [07/Jan/2025 11:29:24] "GET /ask?query=transmittal%20release? HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [07/Jan/2025 11:29:24] "GET /ask?query=transmittal%20release? HTTP/1.1" 200 -
^C
(env_haystack) C:\Users\lenovo\env_haystack\Scripts>python app2.py
```

HTML Terminal



how to reset password?

On the Sign In window enter your login name and click Continue On the Account window, click Reset Password Resetting passwords Using this option an existing password can be removed

On the Sign In window enter your login name and click Continue On the Account window, click Reset Password Resetting passwords Using this option an existing password can be removed

what is transmittal?

Type Details filled Recipients Transmittal

transmittal release?

filled Client document number Transmittal Recipients The Type Master window is displayed.

Ask me something...

Wrench_Sm..._Part_1.pdf

```
from flask import Flask, request, jsonify, render_template

from flask_cors import CORS

from haystack.document_stores import FAISSDocumentStore

from haystack.nodes import EmbeddingRetriever

from transformers import pipeline as hf_pipeline

import fitz  # PyMuPDF

import os

import logging

import time
```

```
# Initialize Flask App
app = Flask(__name__)
CORS(app)

# Directory to save uploaded files
UPLOAD_FOLDER = "data/uploads/"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def safe_remove(file_path, retries=3, delay=1):
    """Attempt to remove a file with retries."""
    for _ in range(retries):
        try:
            if os.path.exists(file_path):
                os.remove(file_path)
                print(f"{file_path} removed successfully.")
                return
        except PermissionError as e:
            print(f"PermissionError: {e}. Retrying in {delay} seconds...")
            time.sleep(delay)
    print(f"Failed to remove {file_path} after {retries} retries.")

# Ensure file cleanup
safe_remove("faiss_document_store.db")
safe_remove("faiss_index")

# Initialize FAISS Document Store
try:
    document_store = FAISSDocumentStore(
        sql_url="sqlite:///faiss_document_store.db",
        faiss_index_factory_str="Flat",
        embedding_dim=384  # Match the embedding model dimension
    )
    print("FAISS Document Store initialized successfully.")
except Exception as e:
    logging.error(f"Error initializing FAISS Document Store: {str(e)}")
    raise

# Initialize Embedding Retriever
retriever = EmbeddingRetriever(
    document_store=document_store,
    embedding_model="sentence-transformers/all-MiniLM-L6-v2",
    use_gpu=True
)

# Update embeddings if documents are present
if document_store.get_document_count() > 0:
    document_store.update_embeddings(retriever)

# Initialize Hugging Face QA pipeline
qa_pipeline = hf_pipeline(
    "question-answering",
    model="deepset/roberta-large-squad2",
    tokenizer="deepset/roberta-large-squad2",
    framework="pt"
)

# Initialize BART paraphrasing pipeline
paraphrase_pipeline = hf_pipeline("text2text-generation", model="facebook/bart-large-cnn")

def paraphrase_answer(answer):
    """Paraphrase the generated answer using BART model."""
    paraphrased = paraphrase_pipeline(answer, max_length=512, num_return_sequences=1)
    return paraphrased[0]['generated_text']

def extract_text_from_pdf(pdf_path):
    """Extract text from a PDF and organize it into sections."""
    doc = fitz.open(pdf_path)
    sections = []
    for page_num, page in enumerate(doc, start=1):
        content = page.get_text("blocks")  # Extract content as blocks
        for block in content:
            text = block[4].strip()
            if len(text) > 20:  # Ignore very short lines
                sections.append({
                    "content": text,
                    "meta": {"page_number": page_num, "source": pdf_path}
                })
    return sections

def format_answer(query, raw_answer, context):
    """Format the response into structured output with additional details."""
    return (
        f"Here you go!\n\n"
        f"**{query.capitalize()}**\n\n"
        f"**Answer:** {raw_answer}\n\n"
        f"**Details:**\n{context}\n\n"
        f"Feel free to ask more questions!"
    )

def generate_structured_response(query, answer, context_list):
    """Generate a structured response dynamically."""
    if answer == "No relevant information found.":
        return (
            f"I'm sorry, I couldn't find any relevant information about '{query}'. "
            "Could you try rephrasing your question or provide more details?"
        )

    # Create a structured response
    context_summary = "\n".join(context_list)
    return (
        f"**{query.capitalize()}**\n\n"
        f"**Answer:** {answer}\n\n"
        f"**Additional Details:**\n{context_summary}\n\n"
        "Let me know if you'd like further clarification or have more questions!"
    )

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

@app.route('/initialize', methods=['POST'])
def initialize():
    global document_initialized
    try:
        if 'file' not in request.files:
            return jsonify({"error": "No file uploaded."}), 400

        file = request.files['file']
        if file.filename == '':
            return jsonify({"error": "No selected file."}), 400

        if not file.filename.endswith('.pdf'):
            return jsonify({"error": "Only PDF files are allowed."}), 400

        file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(file_path)

        # Extract text and index it
        sections = extract_text_from_pdf(file_path)
        documents = [
            {"content": section["content"], "meta": section["meta"]}
            for section in sections
        ]
        document_store.write_documents(documents)

        # Update embeddings in FAISS
        document_store.update_embeddings(retriever)
        document_initialized = True

        # Print embeddings for debugging
        embeddings = retriever.embed_queries([doc['content'] for doc in documents])
        for idx, embedding in enumerate(embeddings):
            print(f"Document {idx + 1} embedding: {embedding}")

        return jsonify({"message": "Document uploaded and processed successfully!"}), 200
    except Exception as e:
        logging.error(f"Error during initialization: {str(e)}")
        return jsonify({"error": str(e)}), 500

@app.route('/ask', methods=['GET'])
def ask():
    if not document_initialized:
        return jsonify({"error": "No document has been uploaded or processed yet."}), 400
```

```
query = request.args.get('query')
if not query:
    return jsonify({"error": "No query provided."}), 400

# Handle basic greetings or acknowledgments
lower_query = query.lower()
if lower_query in ["hi", "hello"]:
    return jsonify({"response": "Hello! How can I assist you today?"}), 200
elif lower_query in ["thank you", "thanks"]:
    return jsonify({"response": "You're welcome! Feel free to ask more questions."}), 200

try:
    # Retrieve relevant documents
    retrieved_docs = retriever.retrieve(query=query, top_k=10)  # Adjust top_k as needed
    contexts = [doc.content for doc in retrieved_docs]

    # Process combined context with QA pipeline
    combined_context = " ".join(contexts[:3])  # Combine top 3 contexts
    result = qa_pipeline(question=query, context=combined_context)

    # Format the answer
    raw_answer = result.get("answer", "No relevant information found.")
    context_snippet = "\n".join(contexts[:3])  # Include additional context for clarity
    formatted_answer = format_answer(query, raw_answer, context_snippet)

    # Paraphrase the answer before sending it back
    paraphrased_answer = paraphrase_answer(formatted_answer)

    return jsonify({"query": query, "answer": paraphrased_answer}), 200
except Exception as e:
        logging.error(f"Error during question answering: {str(e)}")
        return jsonify({"error": str(e)}), 500

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