```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


class RAGPipeline:
    def __init__(self, retriever, generator):
        self.retriever = retriever
        self.generator = generator

    def process_query(self, user_query):
        # Step 1: Retrieve relevant documents
        documents = self.retriever.retrieve(user_query)

        # Step 2: Prepare context
        context = self.prepare_context(documents)

        # Step 3: Generate response
        response = self.generator.generate(context)

        return response

    def prepare_context(self, documents):
        # Combine documents into a single context string
        return "\n".join(documents)

# Usage
retriever = YourRetrievalModel()
generator = YourGenerativeModel()
rag_pipeline = RAGPipeline(retriever, generator)

user_query = "What are the benefits of using Crew AI?"
response = rag_pipeline.process_query(user_query)
print(response)


# In[2]:


pip install faiss-cpu transformers


# In[3]:


import numpy as np
import faiss
from transformers import pipeline

# Dummy dataset
documents = [
    "Crew AI helps automate workflows.",
    "Crew AI can enhance team collaboration.",
    "Using Crew AI improves project management.",
    "Crew AI is beneficial for resource allocation."
]

# Step 1: Create a simple retrieval model using FAISS
class SimpleRetriever:
    def __init__(self, documents):
        self.documents = documents
        self.embeddings = self.embed_documents(documents)
        self.index = faiss.IndexFlatL2(self.embeddings.shape[1])  # Using L2 distance
        self.index.add(self.embeddings)  # Add embeddings to the index

    def embed_documents(self, documents):
```

```
        # Here we simply use a dummy embedding: you should use a real model to embed your
documents
        return np.random.random((len(documents), 128)).astype('float32')

    def retrieve(self, query, k=2):
        # Embed the query (dummy embedding in this example)
        query_embedding = np.random.random((1, 128)).astype('float32')
        distances, indices = self.index.search(query_embedding, k)  # Retrieve top k documents
        return [self.documents[i] for i in indices[0]]

# Step 2: Create a simple generative model using Hugging Face
class SimpleGenerator:
    def __init__(self):
        self.generator = pipeline('text-generation', model='gpt2')  # You can choose any other
model

    def generate(self, context):
        return self.generator(context, max_length=50, num_return_sequences=1)[0]
['generated_text']

# Step 3: Create the RAG Pipeline
class RAGPipeline:
    def __init__(self, retriever, generator):
        self.retriever = retriever
        self.generator = generator

    def process_query(self, user_query):
        # Step 1: Retrieve relevant documents
        documents = self.retriever.retrieve(user_query)

        # Step 2: Prepare context
        context = self.prepare_context(documents)

        # Step 3: Generate response
        response = self.generator.generate(context)

        return response

    def prepare_context(self, documents):
        # Combine documents into a single context string
        return " ".join(documents)

# Usage
retriever = SimpleRetriever(documents)
generator = SimpleGenerator()
rag_pipeline = RAGPipeline(retriever, generator)

user_query = "What are the benefits of using Crew AI?"
response = rag_pipeline.process_query(user_query)
print(response)


# In[ ]:
```