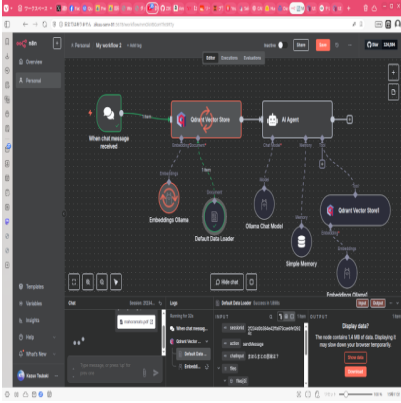




n8nã•Şç°jã•?ã•ªAlã? ¯ã?¼ã? ¯ã??ã?ã?¼ã??ã½?ã•ã•|ã•¿ã??

Description

Ollamaã?•Qdrantã?•n8nã•®çµ?ã•¿ã?ã?ã•?ã•Şç°jã•?ã•ªRAGã?£ã??ã?ã?±ã?¼ã?•ã?Şã?³ã??ã½?ã•£ã•
|ã•¿ã•¼ã•?ã•?ã??

å?•ã?•ã•|ã•®ã½¿ç?ã•ã•£ã•?ã•®ã•Şèªã•¹ã•ªã?ã??ã½?æ¥ã•?ã•|¼?æ??é??ã•ã??ã•?ã•?ã•£ã•?ã•
¨æ?•ã•?ã•¼ã•?ã??ã??ã?¼ã?³ã?¼ã??ã??ã?¼ã?«ã•Şã•?ã•®ã•Şã?•é•?ã??ã?ã?°ã?©ã??ã?¼ã•®æ?¹ã•
Şã??ã•å??ã•«ã½¿ã•?ã??ã•æ?•ã•?ã•¼ã•?ã??

Ollamaã?•Qdrantã?•n8nã?•Open WebUIã??Dockerã•Şã??ã??ã?ã?ã?ã?ã?ã?ã?ã?ã?ã?ã?ã?ã?ã?
ã?¼i¼?GPUç?jã•?i¼?ã•Şã®?èj?ã•?ã•¼ã•?ã•?ã??



ã?¢ã??ã?ªã?±ã?¼ã?-ã?§ã?³ã•®æ©?è?½ã-ã?PDFã??ã?jã?¤ã?«ã??æ.»ã»?ã•?ã•jã?ã•ñ•
®ã??ã?jã?¤ã?«ã•®ã? å®¹ã•«é?¢ã•?ã???è³ªå?•ã??ã??ã?ã?ã?ã?ã?ã?ã?«è`å®?ã•
?ã??ã??Qdrantã??ã?-ã?¿ã?¼ã?¹ã??ã?¢ã«Ollamaã•®å??ã?•è¾¼¼ã•¿ã?¢ã??ã?«ã??ä½¿ã•£ã•
jã??ã?-ã??ã?«ã??ã?¼ã?¿ã??ã?ã•jã¿•å?ã?ã•jã•?ã??ã?ã??ã?ã?ã?ã?ã?ã?ã?«å¾¼ã?ã??å??ç-
?ã??ç??æ?•ã•?ã•!è¿?ã•?ã•ã•?ã•?ã??ã•®ã•§ã•?ã??

ã•jãªã•¿ã•«ã»¥å?•ã•»ã•¼å?ã•?æ©?è?½ã??Pythonã?¹ã?¬ã?ªã?ã??ã?æ?_ã•?ã•?å'å•ã•¬ç'?200è;ã
•ã??ã?ã•®ã?³ã?¼ã??é?•ã•sã•ã?ã?ã??

ã?ã?¼ã??ã-ã?ã:ã??ã??GUIç?:ã?ã®æ??å°•é?•ã®å®?è£ã«ãªã£ã!ã?ã??ã¨æ?ã?ã¾ã?ã??

```
import sys
import os
import base64
from langchain_community.document_loaders import UnstructuredPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_ollama import ChatOllama
from qdrant_client import QdrantClient
from langchain_core.prompts import PromptTemplate
from langchain.chains import RetrievalQA
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_qdrant import QdrantVectorStore
```

```
history = []
MAX HISTORY SIZE = 50
```

```
class EmbeddingsManager:
    def __init__(
```

```
self,
model_name: str = "BAAI/bge-small-en",
device: str = "cpu",
encode_kwargs: dict = {"normalize_embeddings": True},
qdrant_url: str = "http://localhost:6333",
collection_name: str = "vector_db",
):
    self.model_name = model_name
    self.device = device
    self.encode_kwargs = encode_kwargs
    self.qdrant_url = qdrant_url
    self.collection_name = collection_name

    self.embeddings = HuggingFaceEmbeddings(
        model_name=self.model_name,
        model_kwargs={"device": self.device},
        encode_kwargs=self.encode_kwargs,
    )

def create_embeddings(self, pdf_path: str):
    if not os.path.exists(pdf_path):
        raise FileNotFoundError(f"The file {pdf_path} does not exist.")

    loader = UnstructuredPDFLoader(pdf_path)
    docs = loader.load()
    if not docs:
        raise ValueError("No documents were loaded from the PDF.")

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000, chunk_overlap=250
    )
    splits = text_splitter.split_documents(docs)
    if not splits:
        raise ValueError("No text chunks were created from the documents.")

    try:
        qdrant = QdrantVectorStore.from_documents(
            splits,
            self.embeddings,
            url=self.qdrant_url,
            prefer_grpc=False,
            collection_name=self.collection_name,
        )
    except Exception as e:
        raise ConnectionError(f"Failed to connect to Qdrant: {e}")
    return False

    return True

class ChatbotManager:
    def __init__(
        self,
        model_name: str = "BAAI/bge-small-en",
        device: str = "cpu",
        encode_kwargs: dict = {"normalize_embeddings": True},
```

```
llm_model: str = "gemma3:1b",
llm_temperature: float = 0.1,
qdrant_url: str = "http://localhost:6333",
collection_name: str = "vector_db",
):
    self.model_name = model_name
    self.device = device
    self.encode_kwargs = encode_kwargs
    self.llm_model = llm_model
    self.llm_temperature = llm_temperature
    self.qdrant_url = qdrant_url
    self.collection_name = collection_name

    self.embeddings = HuggingFaceEmbeddings(
        model_name=self.model_name,
        model_kwargs={"device": self.device},
        encode_kwargs=self.encode_kwargs,
    )

    self.llm = ChatOllama(
        model=self.llm_model,
        temperature=self.llm_temperature,
    )

    self.prompt_template = """Use the following pieces of information to a
If you don't know the answer, just say that you don't know, don't try to make

Context: {context}
Question: {question}

Only return the helpful answer. Answer must be detailed and well explained in
Helpful answer:
"""

    self.client = QdrantClient(
        url=self.qdrant_url, prefer_grpc=False
    )

    self.db = QdrantVectorStore(
        client=self.client,
        collection_name=self.collection_name,
        embedding=self.embeddings
    )

    self.prompt = PromptTemplate(
        template=self.prompt_template,
        input_variables=['context', 'question']
    )

    self.retriever = self.db.as_retriever(search_kwargs={"k": 1})

    self.chain_type_kwargs = {"prompt": self.prompt}

    self.qa = RetrievalQA.from_chain_type(
        llm=self.llm,
        chain_type="stuff",
```

```
        retriever=self.retriever,
        return_source_documents=False,
        chain_type_kwargs=self.chain_type_kwargs,
        verbose=False
    )

def get_response(self, query: str) -> str:
    try:
        response = self.qa.invoke(query)
        return response # 'response' is now a string containing only the
    except Exception as e:
        st.error(f"An error occurred while processing your request: {e}")
        return "Sorry, I couldn't process your request at the moment."

def load_pdf(filename):
    embedding_manager = EmbeddingsManager()

    print("Retrieving PDF and creating embeddings...")
    result = embedding_manager.create_embeddings(filename)

    if result:
        print("PDF document saved in vector store.")
    return result

def print_help():
    print("Help:")
    print("")
    print("--help          Show this help message")
    print("--pdf=filename  Retrieve PDF document for inference")
    print("")
    print("Example:")
    print("")
    print("    Retrieve PDF document and start chat")
    print("    python3 rag_chatbot.py --pdf=filename.pdf")
    print("")

    print("    Show help")
    print("    python3 rag_chatbot.py --help")

def main():
    args = sys.argv

    showHelp = False
    pdfFile = None

    if len(args) == 1:
        showHelp = True

    for m in args:
        if m == "--help":
            showHelp = True
        elif "--pdf=" in m:
            pdfFile = m[len("--pdf="):]

    if showHelp:
```

```
print_help()
exit(0)
else:
    if pdfFile:
        if load_pdf(pdfFile) == False:
            exit(1)

chatbot_manager = ChatbotManager()

while True:
    prompt = input("ã??ã?ã?³ã??ã??ã??å?¥å??\n")
    if prompt == "":
        break;

    history.append({"role": "user", "content": prompt})

    response = chatbot_manager.get_response(prompt)
    answer = response["result"]
    print("")
    print(answer)
    print("")
    history.append({"role": "assistant", "content": answer})
    while len(history) > MAX_HISTORY_SIZE:
        history.pop(1)

if __name__ == "__main__":
    main()
```

Category

- 1. æ?¥ã? ã•®æ´»å??

Tags

- 1. AI
- 2. Docker
- 3. LLM
- 4. n8n
- 5. Ollama
- 6. Qdrant
- 7. ä,?é??å??ä,?
- 8. å¡§è!•æ¨;è¨?èª?ã?¢ã??ã?«
- 9. å±±æ¢¨ç??

Date Created

2025å¹7æ??29æ?¥

Author

kazuo-tsubaki