

랭체인으로 RAG 구현하기

간단한 챗봇 만들기

간단한 챗봇 만들기

- 챗봇 생성은 스트림릿(streamlit)을 사용
- 스트림릿은 데이터 시각화 및 웹 애플리케이션을 파이썬으로 쉽게 만들 수 있도록 도와주는 오픈 소스 라이브러리

```
!pip install langchain  
!pip install streamlit  
!pip install openai
```

```
#!pip install langchain  
#!pip install streamlit  
#!pip install openai
```

- !pip install langchain-classic

간단한 챗봇 만들기

간단한 챗봇 만들기

- 챗봇 생성은 스트림릿(streamlit)을 사용
- 스트림릿은 데이터 시각화 및 웹 애플리케이션을 파이썬으로 쉽게 만들 수 있도록 도와주는 오픈 소스 라이브러리
- 단순히 gpt-4 모델에서 제공하는 정보만 활용하겠다는 의미
- 챗봇 서비스는 스트림릿을 이용해서 웹페이지로 보여줄 예정

간단한 챗봇 만들기

간단한 챗봇 만들기

- temperature는 모델이 얼마나 정확한 답변을 제공할지 결정하는 파라

```
import os
from dotenv import load_dotenv
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

import streamlit as st
from langchain_classic.chat_models import ChatOpenAI
st.set_page_config(page_title="🐼👁️ 뭐든지 질문하세요~ ")
st.title('🐼👁️ 뭐든지 질문하세요~ ')

def generate_response(input_text): #llm이 답변 생성
    llm = ChatOpenAI(temperature=0, # 창의성 0으로 설정
                     model_name='gpt-4.1-mini', # 모델명
                     )
    st.info(llm.invoke(input_text).content)

with st.form('Question'):
    text = st.text_area('질문 입력:', 'What types of text models does OpenAI provide?') #첫 페이지가 실행될 때 보여줄 질문
    submitted = st.form_submit_button('보내기')
    generate_response(text)
```

간단한 챗봇 만들기

간단한 챗봇 만들기

- temperature=0으로 지정하면 모델이 가장 정확한 답변만 제공
- LLM이 생성한 결과는 예측 가능하고 일관성 있지만 창의성은 떨어질 수 있음
- temperature가 높은 값을 가지면 모델이 제공하는 답변의 정확성은 좀 떨어질 수 있지만 창의적
- 이때 창의적이라는 것은 답변이 좀 더 유연하다는 의미

간단한 챗봇 만들기

간단한 챗봇 만들기

◦ 스트림릿으로 결과 확인

- 코드를 실행했던 주피터 노트북의 Save and Export Notebook As... > Executable Script를 클릭
- 파일이 다운로드 되었다면 다음과 같이 'chat.py'로 이름을 바꿔 줌
(참고로 뒤의 .py는 파이썬 파일임을 나타내는 확장자)
- '!pip install streamlit' 명령어를 실행했다면 Scripts 폴더에 streamlit.exe 파일이 생성되었을 것
- 'C:\Users\<사용자 계정>\anaconda3\envs\Wllm\Scripts' 위치에 streamlit.exe와 chat.py 파일이 있는지 확인

간단한 챗봇 만들기

간단한 챗봇 만들기

◦ 스트림릿으로 결과 확인

- cmd 창에서 C:\WMyLangchain>
- **streamlit run chat.py**
- 추가적인 정보를 보여주면서 웹페이지가 하나 뜸
- 실행된 웹페이지는 파이썬 코드에서 첫 페이지가 실행될 때 자기소개 해 줄 수 있니?

질문을 했으므로 그에 대한 답변

  뭐든지 질문하세요~

질문 입력:

자기소개 해 줄 수 있니?

보내기

안녕하세요, 저는 OpenAI의 인공지능 비서, 챗봇입니다. 다양한 주제에 대해 대화를 나눌 수 있으며, 정보 검색, 일정 관리, 간단한 질문에 대한 답변 등 다양한 업무를 도와드릴 수 있습니다. 언제든지 필요한 도움을 말씀해주세요.

RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- 랭체인과 챗GPT로 RAG 기반의 챗봇 서비스를 구현



RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- 랭체인과 챗GPT로 RAG 기반의 챗봇 서비스를 구현
- RAG를 구현할 것이므로 나만의 데이터가 필요
- 위키피디아에서 찾은 'AI' 검색 결과를 텍스트(text)로 저장해둔 파일을 사용
- 사용할 라이브러리를 설치

```
!pip install unstructured
!pip install sentence-transformers
!pip install chromadb
!pip install openai
!pip install langchain-openai
```

RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- unstructured : 텍스트 파일 같은 구조화되지 않은 데이터를 다루는 데 사용
- chromadb : 벡터를 저장하고 유사도 검색을 지원
- langchain-openai : 오픈시의 대규모 언어 모델(예, GPT 등)과 통합하여 다양한 자연어 처리 및 생성 작업을 쉽게 수행할 수 있도록 도와주는 파이썬 라이브러리

RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- 파일을 불러오는 것은 TextLoader를 이용
- TextLoader는 말 그대로 텍스트 파일을 가져올 때 사용

```
from langchain_classic.document_loaders import TextLoader
documents = TextLoader("AI.txt").load()
```

RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- 문장을 청크로 분할
- 이것은 큰 덩어리의 문서를 작은 덩어리로 분할하는 과정으로 랭체인에서 제공하는 RecursiveCharacterTextSplitter를 이용

```
from langchain_classic.text_splitter import RecursiveCharacterTextSplitter

# 문서를 청크로 분할
def split_docs(documents, chunk_size=1000, chunk_overlap=20):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    docs = text_splitter.split_documents(documents)
    return docs

# docs 변수에 분할 문서를 저장
docs = split_docs(documents)
```

RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- 벡터 데이터베이스인 크로마에 임베딩 처리된 벡터를 저장
- 임베딩 처리는 오픈AI의 text-embedding-ada-002 모델을 사용

#OpenAI의 임베딩 모델 사용

```
from langchain_openai import OpenAIEmbeddings  
embeddings = OpenAIEmbeddings(model="text-embedding-ada-002", api_key=OPENAI_API_KEY)
```

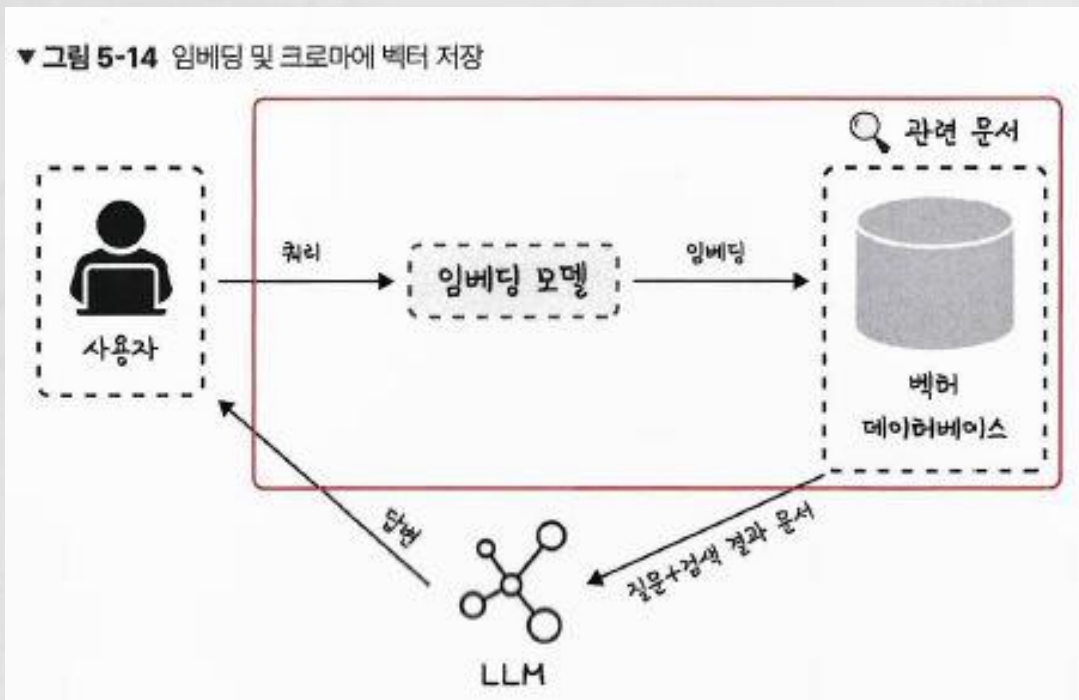
Chromdb에 벡터 저장

```
from langchain_classic.vectorstores import Chroma  
db = Chroma.from_documents(docs, embeddings, persist_directory="data")
```

RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- 청크로 나누고 임베딩 처리해서 크로마에 데이터를 저장하는 것은 다음 단계에 해당



RAG 기반의 챗봇 만들기

RAG 기반의 챗봇 만들기

- 텍스트 파일에서 관련 내용을 찾아서 LLM에 제공하면, LLM이 답변을 생성

```
from langchain_classic.chat_models import ChatOpenAI
model_name = "gpt-4.1-mini"
llm = ChatOpenAI(model_name=model_name, api_key=OPENAI_API_KEY)

# Q&A 체인을 사용하여 쿼리에 대한 답변 얻기
from langchain_classic.chains.question_answering import load_qa_chain
chain = load_qa_chain(llm, chain_type="stuff", verbose=True)

# 쿼리를 작성하고 유사성 검색을 수행하여 답변을 생성, 따라서 txt에 있는 내용을 질의해야 합니다
query = "AI란?"
matching_docs = db.similarity_search(query)
answer = chain.run(input_documents=matching_docs, question=query)
answer
```

PDF 요약 웹사이트 만들기

PDF 요약 웹 사이트 만들기

- PDF 파일을 요약해주는 서비스
- 웹페이지에 PDF 파일을 업로드하면 요약해서 보여주는 것
- PyPDF2는 PDF 파일을 읽고, 분할하고, 병합하고, 순서를 바꾸고, 추가하거나, 암호화하는 등의 작업을 할 수 있게 해주는 라이브러리
- 파이썬에서 PDF 파일을 처리하기 위한 라이브러리로 PyPDF와 유사한 기능을 제공

```
!pip install langchain
!pip install streamlit
!pip install PyPDF2
!pip install langchain-openai
```


PDF 요약 웹사이트 만들기

PDF 요약 웹 사이트 만들기

- 사용자가 PDF 파일을 업로드하면 그것을 청크로 분할하고 임베딩 처리
- LLM한테 해당 파일을 요약해 달라고 쿼리를 던지는 흐름

```
import os
from dotenv import load_dotenv
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

from PyPDF2 import PdfReader
import streamlit as st
from langchain_classic.text_splitter import CharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_classic import FAISS
from langchain_classic.chains.question_answering import load_qa_chain
from langchain_classic.chat_models import ChatOpenAI
from langchain_classic.callbacks import get_openai_callback
```

PDF 요약 웹사이트 만들기

PDF 요약 웹 사이트 만들기

- 사용자가 PDF 파일을 업로드하면 그것을 청크로 분할하고 임베딩 처리
- LLM한테 해당 파일을 요약해 달라고 쿼리를 던지는 흐름

```
def process_text(text):  
    #CharacterTextSplitter를 사용하여 텍스트를 청크로 분할  
    text_splitter = CharacterTextSplitter(  
        separator="\n",  
        chunk_size=1000,  
        chunk_overlap=200,  
        length_function=len  
    )  
    chunks = text_splitter.split_text(text)  
  
    #임베딩 처리(벡터 변환), 임베딩은 OpenAI 모델을 사용합니다.  
    embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")  
    documents = FAISS.from_texts(chunks, embeddings)  
    return documents
```

PDF 요약 웹사이트 만들기

PDF 요약 웹 사이트 만들기

```
def main(): #streamlit을 이용한 웹사이트 생성
    st.title("📄 PDF 요약하기")
    st.divider()

    pdf = st.file_uploader('PDF파일을 업로드해주세요', type='pdf')

    if pdf is not None:
        pdf_reader = PdfReader(pdf)
        text = "" # 텍스트 변수에 PDF 내용을 저장
        for page in pdf_reader.pages:
            text += page.extract_text()

        documents = process_text(text)
        query = "업로드된 PDF 파일의 내용을 약 3~5문장으로 요약해주세요." # LLM에 PDF파일 요약 요청

        if query:
            docs = documents.similarity_search(query)
            llm = ChatOpenAI(model="gpt-4.1-mini", temperature=0.1)
            chain = load_qa_chain(llm, chain_type='stuff')

            with get_openai_callback() as cost:
                response = chain.run(input_documents=docs, question=query)
                print(cost)

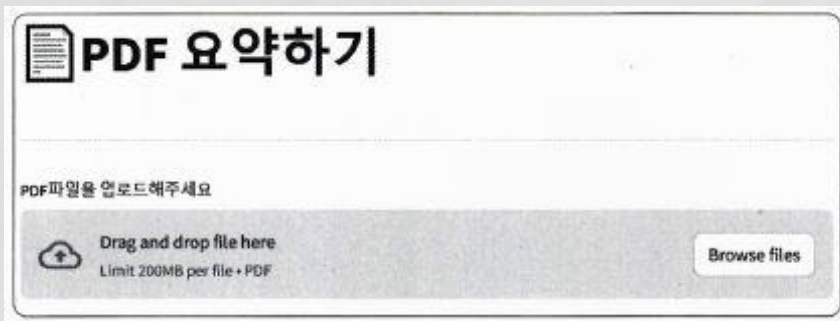
            st.subheader('--요약 결과--:')
            st.write(response)

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

PDF 요약 웹사이트 만들기

PDF 요약 웹 사이트 만들기

- .py 확장자로 파일을 내려받은 후 pdf_summary.py로 이름을 바꿈
- 해당 파일을 'C:\WMyLangChin'로 이동시키고 cmd(아나콘다 프롬프트) 창에서 다음 명령어를 실행
- **streamlit run pdf_summary.py**



PDF 요약하기

PDF파일을 업로드해주세요



Drag and drop file here

Limit 200MB per file • PDF

Browse files



The_Adventures_of_Tom_Sawyer.pdf 2.6MB



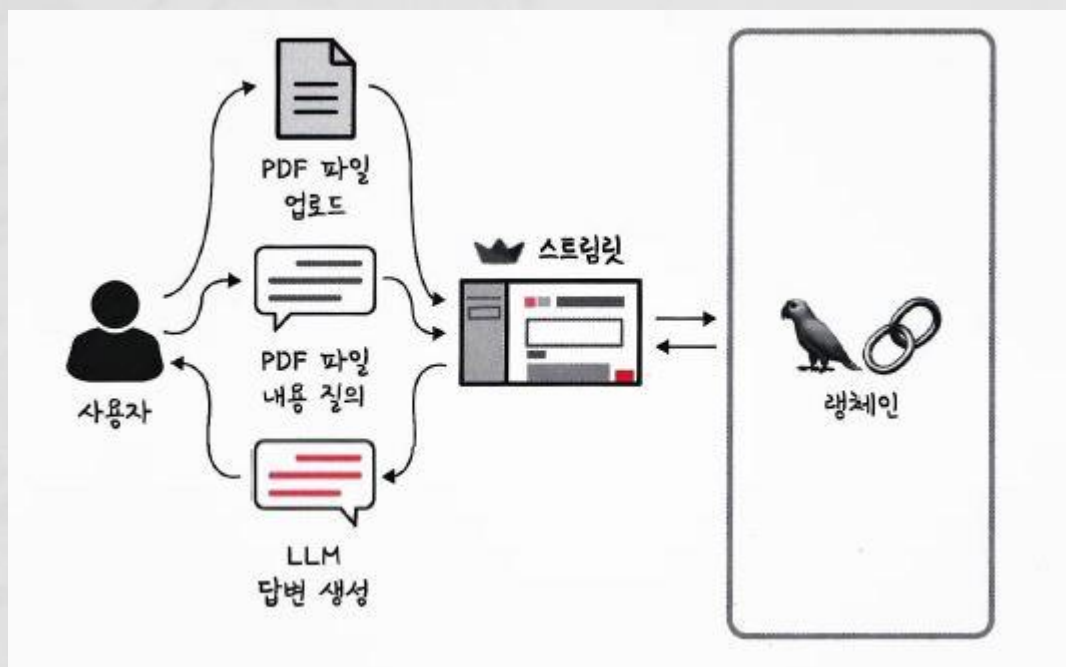
--요약 결과--:

이 PDF 파일은 펄크 리더스 시리즈에 속하는 책인 "톰 소여의 모험"에 대한 정보를 제공합니다. 이 책은 쉬운 단계부터 고급 단계까지 다양한 수준의 독자들을 위해 설계된 간소화된 텍스트입니다. 이 책은 톰 소여가 집, 학교, 친구들과 함께 모험을 즐기는 이야기를 다루고 있습니다. 그는 무서운 묘지와 동굴에서 각각 하나의 모험을 경험하게 되는데, 거기에서 누구를 만나고 왜 두려워하는지 알 수 있습니다. 이

독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

- 가지고 있는 PDF 파일을 이용하여 RAG를 구성
- RAG는 질문을 받으면 먼저 가지고 있는 PDF 파일에서 내용을 찾아보고 질문과 찾은 결과를 LLM에 제공하여 답변을 하는 방식



독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

◦ 필요한 라이브러리를 설치

```
!pip install langchain
!pip install streamlit
!pip install PyPDF2
!pip install langchain-openai
```

◦ 설치한 라이브러리를 가져오기

```
import streamlit as st
from PyPDF2 import PdfReader
from langchain_openai import OpenAIEmbeddings
from langchain_classic.chat_models import ChatOpenAI
from langchain_classic.chains import ConversationalRetrievalChain, RetrievalQA
from langchain_classic.memory import ConversationBufferWindowMemory
from langchain_classic.vectorstores import FAISS
from langchain_classic.document_loaders import PyPDFLoader
from langchain_classic.text_splitter import RecursiveCharacterTextSplitter

import os
from dotenv import load_dotenv
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

- PDF 문서를 가져와서 텍스트를 추출하고 청크로 분할

```
#PDF 문서에서 텍스트를 추출
def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PdfReader(pdf)
        for page in pdf_reader.pages:
            text += page.extract_text()
    return text

#지정된 조건에 따라 주어진 텍스트를 더 작은 덩어리로 분할
def get_text_chunks(text):
    text_splitter = RecursiveCharacterTextSplitter(
        separators="\n",
        chunk_size=1000,
        chunk_overlap=200,
        length_function=len
    )
    chunks = text_splitter.split_text(text)
    return chunks
```

독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

- 분할된 청크는 임베딩 처리를 하고 파이스(FAISS)에 저장

```
#주어진 텍스트 청크에 대한 임베딩을 생성하고 FAISS를 사용하여 벡터 저장소를 생성  
def get_vectorstore(text_chunks):  
    embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")  
    vectorstore = FAISS.from_texts(texts=text_chunks, embedding=embeddings)  
    return vectorstore
```


독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

- 주고받은 대화는 ConversationBufferWindowMemory를 이용하여 저장
- ConversationalRetrievalChain을 통해 챗봇에 쿼리를 전달

#주어진 벡터 저장소로 대화 체인을 초기화

```
def get_conversation_chain(vectorstore):
```

#ConversationBufferWindowMemory에 이전 대화 저장

```
memory = ConversationBufferWindowMemory(memory_key='chat_history', return_message=True)
```

```
conversation_chain = ConversationalRetrievalChain.from_llm(
```

```
    llm=ChatOpenAI(temperature=0, model_name='gpt-4.1-mini'),
```

```
    retriever=vectorstore.as_retriever(),
```

```
    get_chat_history=lambda h: h,
```

```
    memory=memory
```

```
) #ConversationalRetrievalChain을 통해 Langchain 챗봇에 쿼리 전송
```

```
return conversation_chain
```

독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

- PDF 파일을 업로드할 수 있는 버튼을 만들고 업로드 된 PDF 파일은 앞에서 정의해둔 함수(get_text_chunks)를 호출하여 처리

```
user_uploads = st.file_uploader("파일을 업로드해주세요~", accept_multiple_files=True)
if user_uploads is not None:
    if st.button("Upload"):
        with st.spinner("처리중.."):
            # PDF 텍스트 가져오기
            raw_text = get_pdf_text(user_uploads)
            # 텍스트에서 청크 검색
            text_chunks = get_text_chunks(raw_text)
            # PDF 텍스트 저장을 위해 FAISS 벡터 저장소 만들기
            vectorstore = get_vectorstore(text_chunks)
            # 대화 체인 만들기
            st.session_state.conversation = get_conversation_chain(vectorstore)
```

독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

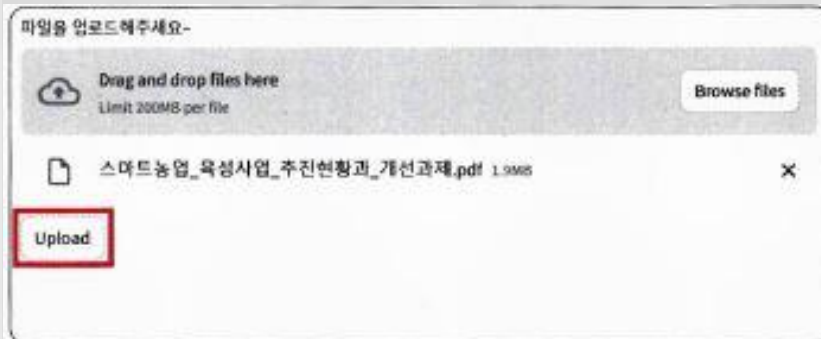
- 스트림릿을 이용해 웹페이지 화면을 구성

```
if user_query := st.chat_input("질문을 입력해주세요~"):
    # 대화 체인을 사용하여 사용자의 메시지를 처리
    if 'conversation' in st.session_state:
        result = st.session_state.conversation({
            "question": user_query,
            "chat_history": st.session_state.get('chat_history', [])
        })
        response = result["answer"]
    else:
        response = "먼저 문서를 업로드해주세요~."
    with st.chat_message("assistant"):
        st.write(response)
```

독립형 질문 챗봇 만들기

독립형 질문 챗봇 만들기

- 스트림릿을 이용해 웹페이지 화면을 구성
- pdf_query.py로 이름을 바꾸고 파일
- **streamlit run pdf_query.py**



파일을 업로드해주세요~



Drag and drop files here
Limit 200MB per file

Browse files



The_Adventures_of_Tom_Sawyer.pdf 2.6MB



Upload

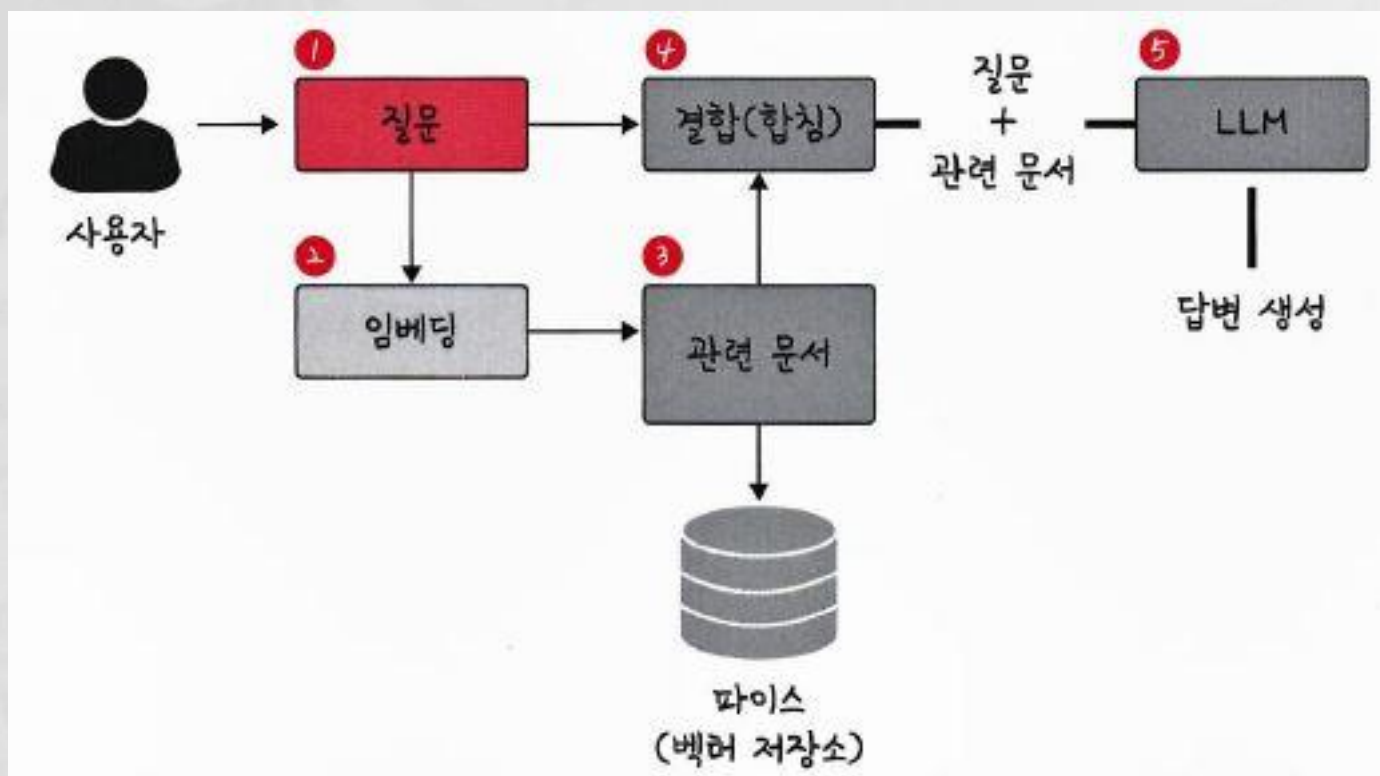


이 PDF는 "The Adventures of Tom Sawyer"라는 책의 Penguin Readers 시리즈 버전에 대한 소개입니다. 이 책은 Mark Twain의 원작을 Jacqueline Kehl이 Level 1 수준으로 재구성한 것입니다. Penguin Readers는 다양한 수준의 독자들을 위해 고안된 graded Readers 시리즈로, 이 책은 특히 영어 학습자들을 위한 것입니다. PDF에는 책의 출판 정보, 저작권 정보, 그리고 Penguin Readers 시리즈에 대한 일반적인 정보가 포함되어 있습니다. 또한, 책의 일부 내용이 간략하게 소개되어 있으며, Tom Sawyer와 그의 친구들이 겪는 모험에 대한 이야기가 담겨 있습니다. PDF는 Pearson Education Limited와 Penguin Books Ltd에 의해 출판되었습니다.

대화형 챗봇 만들기

대화형 챗봇 만들기

- 챗GPT 사이트와 유사한 대화형 챗봇



대화형 챗봇 만들기

대화형 챗봇 만들기

- 필요한 라이브러리를 설치
- faiss-cpu는 벡터 검색을 위한 인덱싱 및 검색 알고리즘
- streamlit_chat은 챗봇 사용자 인터페이스를 생성하는 데 사용

```
!pip install streamlit-chat  
!pip install streamlit  
!pip install langchain  
!pip install faiss-cpu
```

- !pip install langchain-classic

대화형 챗봇 만들기

대화형 챗봇 만들기

- 설치한 라이브러리 가져옴

```
import os
from dotenv import load_dotenv
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

import streamlit as st
from streamlit_chat import message
from langchain_classic.embeddings.openai import OpenAIEmbeddings
from langchain_classic.chat_models import ChatOpenAI
from langchain_classic.chains import ConversationalRetrievalChain
from langchain_classic.vectorstores import FAISS
import tempfile
from langchain_classic.document_loaders import PyPDFLoader
```

대화형 챗봇 만들기

대화형 챗봇 만들기

- PDF 파일이 업로드되면 파일을 가져와서(loader.load()) 임베딩 처리하고 파이스(FAISS)에 저장
- 사용자가 PDF 파일에 대해 질문하면 그것이 새로운 질문인지, 혹은 과거 질문 이력이 있는지 확인하여 답변을 처리

```
uploaded_file = st.sidebar.file_uploader("upload", type="pdf")

if uploaded_file :
    with tempfile.NamedTemporaryFile(delete=False) as tmp_file:
        tmp_file.write(uploaded_file.getvalue())
        tmp_file_path = tmp_file.name

    loader = PyPDFLoader(tmp_file_path)
    data = loader.load()

    embeddings = OpenAIEmbeddings()
    vectors = FAISS.from_documents(data, embeddings)
```


대화형 챗봇 만들기

대화형 챗봇 만들기

```
chain = ConversationalRetrievalChain.from_llm(
    llm = ChatOpenAI(temperature=0.0,model_name='gpt-4.1-mini'),
    retriever=vectors.as_retriever())

def conversational_chat(query): #문맥 유지를 위해 과거 대화 저장 이력에 대한 처리
    result = chain({"question": query, "chat_history": st.session_state['history']})
    st.session_state['history'].append((query, result["answer"]))
    return result["answer"]

if 'history' not in st.session_state:
    st.session_state['history'] = []

if 'generated' not in st.session_state:
    st.session_state['generated'] = ["안녕하세요! " + uploaded_file.name + "에 관해 질문주세요."]

if 'past' not in st.session_state:
    st.session_state['past'] = ["안녕하세요!"]
```

대화형 챗봇 만들기

대화형 챗봇 만들기

```
#챗봇 이력에 대한 컨테이너
response_container = st.container()
#사용자가 입력한 문장에 대한 컨테이너
container = st.container()

with container: #대화 내용 저장(기억)
    with st.form(key='Conv_Question', clear_on_submit=True):
        user_input = st.text_input("Query:", placeholder="PDF파일에 대해 얘기해볼까요? (:", key='input')
        submit_button = st.form_submit_button(label='Send')

    if submit_button and user_input:
        output = conversational_chat(user_input)

        st.session_state['past'].append(user_input)
        st.session_state['generated'].append(output)
```

대화형 챗봇 만들기

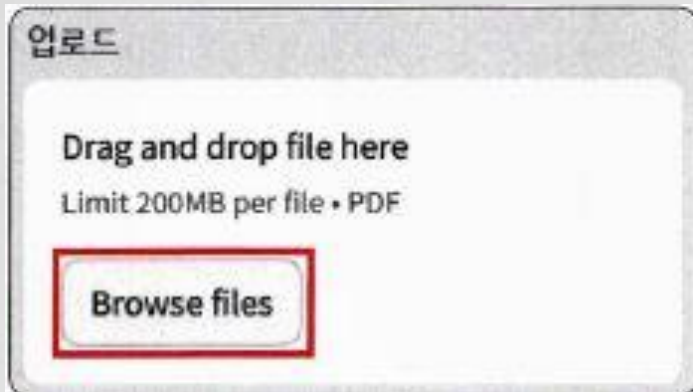
대화형 챗봇 만들기

```
if st.session_state['generated']:  
    with response_container:  
        for i in range(len(st.session_state['generated'])):  
            message(st.session_state["past"][i],  
                    is_user=True,  
                    key=str(i) + '_user',  
                    avatar_style = "fun-emoji",  
                    seed = "Nala")  
            message(st.session_state["generated"][i],  
                    key=str(i),  
                    avatar_style = "bottts",  
                    seed = "Fluffy")
```

대화형 챗봇 만들기

대화형 챗봇 만들기

- .py확장자로 내려받은 파일을 conversation.py로 이름을 변경
- 해당 파일 이동 후 쉘 창에서 다음의 명령어 실행
- `streamlit run conversation.py`



대화형 챗봇 만들기

대화형 챗봇 만들기

◦ 추가 답변



안녕하세요! The_Adventures_of_Tom_Sawyer.pdf에 관해 질문주세요.

이 파일에 대해서 간략하게 설명해주세요



이 파일은 톰 소여의 모험에 대한 이야기를 담고 있습니다. 톰은 학교를 가는 것과 일하는 것을 싫어하며, 대신 놀고 모험을 즐깁니다. 그는 한 번 학교를 빼먹고 강으로 가서 이모인 폴리 아주머니가 화를 내고, 그를 벌로 페인트칠을 시킵니다. 톰은 친구들에게 페인트칠을 시키면서 음식을 받습니다. 또한, 톰과 베키라는 여자아이 사이에 갈등이 있습니다. 베키는 톰이 다른 여자아이 에이미와 이야기하는 것에 화를 내고, 톰은 베키가 다른 친구 알프레드와 이야기하는 것에 화를 냅니다. 이후 베키는 선생님의 책을 찢어버리는 사건이 발생하고, 선생님이 이를 누가 했는지 묻습니다. 이 파일은 펑귄 리더스 시리즈의 일부로, 펑귄 롱맨 출판사에서 출판되었습니다.

무덤에서 의사를 살해한 사람은 누구인가요?



이야기에서 의사를 살해한 사람은 Injun Joe입니다.

Query:

PDF파일에 대해 얘기해볼까요? (;

Send

번역 서비스 만들기

번역 서비스 만들기

- 번역 서비스
- 단순히 영어를 한글로 번역하는 것이 아니라 주어진 텍스트를 몇 개의 언어로 번역할 수 있는 서비스
- 필요한 라이브러리를 설치

```
!pip install langchain  
!pip install streamlit  
!pip install openai
```

- **!pip install langchain-classic**

번역 서비스 만들기

번역 서비스 만들기

- 필요한 라이브러리를 호출

```
import streamlit as st
from langchain_classic.chat_models import ChatOpenAI
from langchain_classic.prompts import PromptTemplate
from langchain_classic.chains import LLMChain
from langchain_classic.memory import ConversationBufferMemory
import os
from dotenv import load_dotenv
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

번역 서비스 만들기

번역 서비스 만들기

◦ 프롬프트는 PromptTemplate을 이용

```
# 웹페이지에 보여질 내용
langs = ["Korean", "Japanese", "chinese", "English"] #번역을 할 언어를 나열
left_co, cent_co, last_co = st.columns(3)

#웹페이지 왼쪽에 언어를 선택할 수 있는 라디오 버튼
with st.sidebar:
    language = st.radio('번역을 원하는 언어를 선택해주세요.:', langs)

st.markdown('### 언어 번역 서비스예요~')
prompt = st.text_input('번역을 원하는 텍스트를 입력하세요') #사용자의 텍스트 입력

trans_template = PromptTemplate(
    input_variables=['trans'],
    template='Your task is to translate this text to ' + language + 'TEXT: {trans}'
) #해당 서비스가 번역에 대한 것임을 지시
```


번역 서비스 만들기

번역 서비스 만들기

#memory는 텍스트 저장 용도

```
memory = ConversationBufferMemory(input_key='trans', memory_key='chat_history')
```

```
llm = ChatOpenAI(temperature=0.0,model_name='gpt-4.1-mini')
```

```
trans_chain = LLMChain(llm=llm,  
                        prompt=trans_template,  
                        verbose=True,  
                        output_key='translate',  
                        memory=memory)
```

프롬프트(trans_template)가 있으면 이를 처리하고 화면에 응답을 작성

```
if st.button("번역"):  
    if prompt:  
        response = trans_chain({'trans': prompt})  
        st.info(response['translate'])
```

번역 서비스 만들기

번역 서비스 만들기

- py확장자로 내려 받은 파일을 translate.py로 이름을 변경
- 해당 파일 이동 후 쉘 창에서 다음의 명령어 실행
- `streamlit run translate.py`

번역을 원하는 언어를 선택해주세요.:

- ☐ Korean
- ☐ Japanese
- ☐ chinese
- ☒ English

언어 번역 서비스예요~

번역을 원하는 텍스트를 입력하세요

이 실습은 원하는 언어로 번역을 해주는 서비스를 만드는 것에 대해 소개합니다.

번역

This practice introduces the service of creating a translation in the language you want.

번역 서비스 만들기

번역 서비스 만들기

번역을 원하는 언어를 선택해주세요.

- ☐ Korean
- ☐ Japanese
- ☐ chinese
- ☒ English

번역을 원하는 언어를 선택해주세요.

- ☐ Korean
- ☐ Japanese
- ☒ chinese
- ☐ English

언어 번역 서비스예요~

번역을 원하는 텍스트를 입력하세요

전자 공학에서 DDR3 SDRAM은 컴퓨터와 다른 디지털 회로 장치에서 데이터를 빠르게 처리하는 데 쓰

번역

In electronic engineering, DDR3 SDRAM is a RAM technology used to quickly process data in computers and other digital circuit devices. It is part of the SDRAM technology, one of the DRAMs, and has superior operation speed and power consumption compared to the previous DDR2 SDRAM products. Samsung Electronics developed it for the first time in the industry in 2005. The main advantage of DDR3 is that it can operate the input/output bus four times faster than the speed of memory cells, thereby implementing a faster bus speed than previous memory technologies. However, faster bus speeds and throughput can increase latency, which can reduce overall speed. Also, DDR3 has set the standard chip capacity from 512 megabits to 8 gigabits. Therefore, it can use up to 16 gigabytes of memory modules.

언어 번역 서비스예요~

번역을 원하는 텍스트를 입력하세요

전자 공학에서 DDR3 SDRAM은 컴퓨터와 다른 디지털 회로 장치에서 데이터를 빠르게 처리하는 데 쓰

번역

在电子工程中，DDR3 SDRAM是一种在计算机和其他数字电路设备中快速处理数据的RAM技术。它是SDRAM系列技术中的一种，其操作速度和功耗等方面优于前一代DDR2 SDRAM产品。三星电子于2005年首次开发了这项技术。DDR3的主要优点是其输入/输出总线的速度可以比内存单元的速度快4倍，因此可以实现比以前的内存技术更快的总线速度。然而，更快的总线速度和吞吐量可能会增加延迟，从而降低整体速度。此外，DDR3已将512兆比特到8千兆比特的芯片容量设为标准。因此，可以使用最大16千兆字节的内存模块。

메일 작성기 만들기

메일 작성기 만들기

- 메일을 작성해주는 애플리케이션
- 메일뿐만 아니라 원하는 어떤 것이든 텍스트 생성이 필요한 곳이라면 어디든 적용
- 필요한 라이브러리를 설치

```
!pip install streamlit  
!pip install langchain  
!pip install openai
```

- !pip install langchain-classic

메일 작성기 만들기

메일 작성기 만들기

- 구현을 위해 openAI 키를 가져오는 것

```
import streamlit as st
import os
from dotenv import load_dotenv
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

- 어떤 내용으로 이메일을 작성할지 사용자로부터 정보를 받음

```
st.set_page_config(page_title="이메일 작성 서비스예요~", page_icon=":robot:")
st.header("이메일 작성기")

def getEmail():
    input_text = st.text_area(label="메일 입력", label_visibility='collapsed',
                              placeholder="당신의 메일은...", key="input_text")
    return input_text

input_text = getEmail()
```


메일 작성기 만들기

메일 작성기 만들기

- 이메일 작성을 위한 템플릿(template)을 정의

```
# 이메일 변환 작업을 위한 템플릿 정의
query_template = """
    메일을 작성해주세요.
    아래는 이메일입니다:
    이메일: {email}
    """
```

- 템플릿을 사용하기 위해 PromptTemplate 인스턴스를 생성

```
from langchain_classic import PromptTemplate
# PromptTemplate 인스턴스 생성
prompt = PromptTemplate(
    input_variables=["email"],
    template=query_template,
)
```


메일 작성기 만들기

메일 작성기 만들기

- gpt-4 LLM을 호출

```
from langchain_classic.chat_models import ChatOpenAI
# 언어 모델을 호출합니다
def loadLanguageModel():
    llm = ChatOpenAI(temperature=0.0,model_name='gpt-4.1-mini')
    return llm
```

메일 작성기 만들기

메일 작성기 만들기

◦ 웹 화면에서 보여질 구성을 설정

예시 이메일을 표시

```
st.button("*예제를 보여주세요*", type='secondary', help="봇이 작성한 메일을 확인해보세요.")  
st.markdown("### 봇이 작성한 메일은:")
```

```
if input_text:
```

```
    llm = loadLanguageModel()
```

PromptTemplate 및 언어 모델을 사용하여 이메일 형식을 지정

```
    prompt_with_email = prompt.format(email=input_text)
```

```
    formatted_email = llm.invoke(prompt_with_email).content
```

서식이 지정된 이메일 표시

```
    st.write(formatted_email)
```

메일 작성기 만들기

메일 작성기 만들기

- py확장자로 내려 받은 파일을 email.py로 이름을 변경
- 해당 파일 이동 후 쉘 창에서 다음의 명령어 실행
- streamlit run email.py

이메일 작성기

당신의 메일은...

예제를 보여주세요

봇이 작성한 메일은:

이메일 작성기

다음주 월요일에 오후 5시에 첫 번째 멘토님과의 zoom 회의가 예정되어 있고 그 주 수요일 오후 2시에 두 번째 멘토님과의 zoom 회의가 예정되어 있으므로 참여가 불가능한 인원은 미리 매니저님께 연락 부탁드립니다.

예제를 보여주세요

봇이 작성한 메일은: ↩

제목: 다음주 멘토님과의 zoom 회의 일정 안내

안녕하세요,

다음주 월요일과 수요일에 멘토님들과의 zoom 회의가 예정되어 있음을 알려드립니다.

첫 번째 회의는 월요일 오후 5시에, 두 번째 회의는 수요일 오후 2시에 진행될 예정입니다.

회의 참여가 어려운 분은 미리 매니저님께 연락해주시기 바랍니다.

감사합니다.

[귀하의 이름]

CSV 파일 분석하기

CSV 파일 분석하기

- 엑셀(혹은 CSV)파일에 기반한 분석
- 필요한 라이브러리를 설치

```
!pip install langchain-experimental  
!pip install tabulate  
!pip install pandas  
!pip install openai
```

- langchain_experimental은 외부 데이터 소스(예: 판다스를 사용할 때 필요)
- tabulate 라이브러리는 파이썬에서 테이블 형태의 데이터를 깔끔하고 읽기 쉬운 텍스트 형식으로 출력하는 데 사용

CSV 파일 분석하기

CSV 파일 분석하기

- 파일 불러오기

```
import pandas as pd #파이썬 언어로 작성된 데이터를 분석 및 조작하기 위한 라이브러리

#csv 파일을 데이터프레임으로 가져오기
df = pd.read_csv('booksv_02.csv') #booksv_02.csv 파일이 위치한 경로 지정
df.head()
```

CSV 파일 분석하기

CSV 파일 분석하기

- LLM에 CSV 파일의 내용을 전달
- LLM은 gpt-4.1-mini 모델을 사용

```
from langchain_experimental.agents.agent_toolkits import create_pandas_dataframe_agent
from langchain_classic.chat_models import ChatOpenAI
from langchain_classic.agents.agent_types import AgentType

#에이전트 생성
agent = create_pandas_dataframe_agent(
    ChatOpenAI(temperature=0, model='gpt-4.1-mini'),
    df,          #데이터가 담긴 곳
    verbose=False, #추론 과정을 출력하지 않음
    agent_type=AgentType.OPENAI_FUNCTIONS,
    allow_dangerous_code=True,
)
```


CSV 파일 분석하기

CSV 파일 분석하기

- CSV 파일에서 어떤 책의 ratings_count가 제일 높은지 질문

```
agent.run('어떤 제품의 ratings_count가 제일 높아?')
```

```
'ratings_count가 가장 높은 제품은 "One Hundred Years of Solitude"이며, ratings_count는 547,207입니다.'
```

- 가장 최근에 출간된 책이 무엇인지 추가 질문

```
agent.run ( '가장 최근에 출간된 책은?' )
```

```
'가장 최근에 출간된 책은 2007년에 출간된 "The Yiddish Policemen\'s Union"입니다.  
저자는 Michael Chabon입니다.'
```

정리

정리

- 간단한 챗봇 만들기
- RAG 기반의 챗봇 만들기
- PDF 요약 웹사이트 만들기
- 독립형 질문 챗봇 만들기
- 대화형 챗봇 만들기
- 번역 서비스 만들기
- 메일 작성기 만들기
- CSV 파일 분석하기