



# ***LangChain*을 활용한 PDF 챗봇 구축**



## ■ contents

- 01. LangChain 이론
- 02. LangChain 구조
- 03. Prompt
- 04. RAG (Retrieval Augmented Generation)
- 05. PDF 챗봇 구축 전략
- 06. LM Studio 설치 및 환경 설정
- 07. 실습 (LangChain – PDF 챗봇 구축 )

# 01. LangChain 이론

---

## LangChain이란?



언어 모델로 구동되는 애플리케이션을 개발하기 위한 프레임 워크

대표적인 기능

- 데이터 인식 : 언어 모델이 다른 데이터 소스에 연결하는 기능
- 에이전트 기능 : 언어 모델이 환경과 상호작용 할 수 있도록 구성하는 기능

구체적인 방법론을 제공하지 않아도 알아서 생각해서 찾아내도록 하는 기능

**ex)** 기사 검색, 최신 동향, 인터넷 검색 등

# 01. LangChain 이론

## ChatGPT 한계



ChatGPT

### • 1. 정보 접근 제한

ChatGPT(GPT-3.5)는 2021년까지의 데이터를 학습한 LLM(초거대언어모델)이므로, 2022년부터의 정보에 대해서는 답변을 하지 못하거나, 거짓된 답변을 제공한다.

### • 2. 토큰 제한

ChatGPT에서 제공하는 모델인 GPT-3.5와 GPT-4는 각각 4096, 8192토큰이라는 입력 토큰 제한이 존재한다.

### • 3. 환각현상(Hallucination)

Fact에 대한 질문을 했을 때, 엉뚱한 대답을 하거나 거짓말을 하는 경우가 많다.

# 01. LangChain 이론

---

ChatGPT 한계점 극복 방법 => ChatGPT 개량

## 1. Fine-Tuning

기존 딥러닝 모델의 **weight**를 조정하여 원하는 용도의 모델로 업데이트 하는 방법

## 2. N-shot Learning

0개 ~ n개의 출력 예시를 제시하여, 딥러닝이 용도에 알맞은 출력을 하도록 조정하는 방법

## 3. In-Context Learning

문맥을 제시하고, 이 문맥 기반으로 모델이 출력하도록 조정하는 방법

내가 질문하고자 하는 정보를 미리 주어 주고 원하는 답을 얻는 방법

In-Context Learning을 도와주는 프레임워크가 **LangChain**

장점 : 어떤 분야에 잘 알고 있는 챗봇이랑 대화하고 싶다는 문제는 대부분 해결 가능

**ChatGPT**

### • 1. 정보 접근 제한

ChatGPT(GPT-3.5)는 2021년까지의 데이터를 학습한 LLM(초거대언어모델)이므로, 2022년부터의 정보에 대해서는 답변을 하지 못하거나, 거짓된 답변을 제공한다.



**Vectorstore 기반 정보 탐색 or Agent 활용한 검색 결합**

### • 2. 토큰 제한

ChatGPT에서 제공하는 모델인 GPT-3.5와 GPT-4는 각각 4096, 8192토큰이라는 입력 토큰 제한이 존재한다.



**TextSplitter를 활용한 문서 분할**

### • 3. 환각현상(Hallucination)

Fact에 대한 질문을 했을 때, 엉뚱한 대답을 하거나 거짓말을 하는 경우가 많다.

## 02. LangChain 구조



# LangChain

### LLM

: 초거대 언어모델로, 생성 모델의 엔진과 같은 역할을 하는 핵심 구성 요소

예시: GPT-3.5, PALM-2, LLAMA, StableVicuna, WizardLM, MPT, ...

### Prompts

: 초거대 언어모델에게 지시하는 명령문

요소: Prompt Templates, Chat Prompt Template, Example Selectors, Output Parsers

### Index

: LLM이 문서를 쉽게 탐색할 수 있도록 구조화 하는 모듈

예시: Document Loaders, Text Splitters, Vectorstores, Retrievers, ...

### Memory

: 채팅 이력을 기억하도록 하여, 이를 기반으로 대화가 가능하도록 하는 모듈

예시: ConversationBufferMemory, Entity Memory, Conversation Knowledge Graph Memory, ...

### Chain

: LLM 사슬을 형성하여, 연속적인 LLM 호출이 가능하도록 하는 핵심 구성 요소

예시: LLM Chain, Question Answering, Summarization, Retrieval Question/Answering, ...

### Agents

: LLM이 기존 Prompt Template으로 수행할 수 없는 작업을 가능케하는 모듈

예시: Custom Agent, Custom MultiAction Agent, Conversation Agent, ...

### 03. Prompt란



Day1. 사과와 빵이 들어간 음식을  
추천하고, 그것의 레시피를 알려줘.

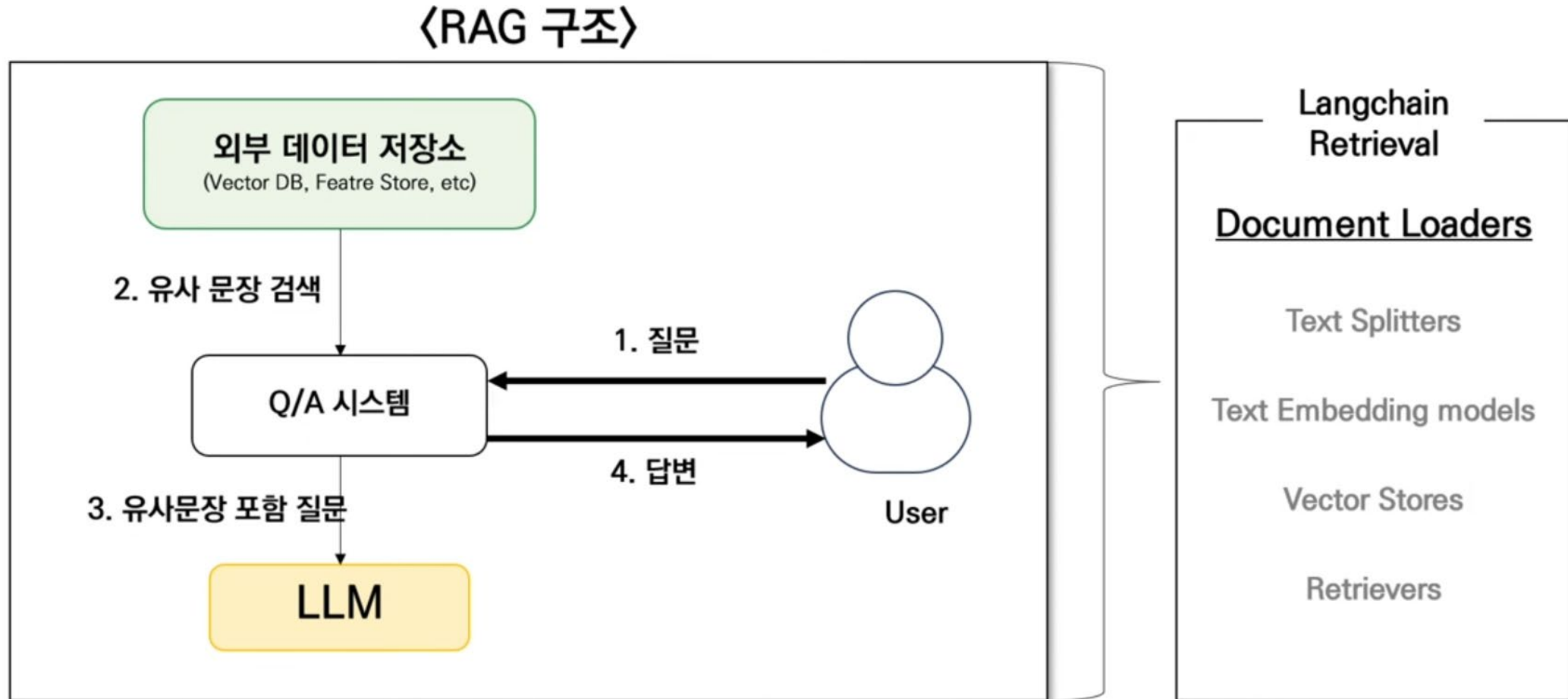
Day 2. 쿼와 초콜렛이 들어간 음식을  
추천하고, 그것의 레시피를 알려줘.

Day 3. 장어와 고추장이 들어간 음식을  
추천하고, 그것의 레시피를 알려줘.

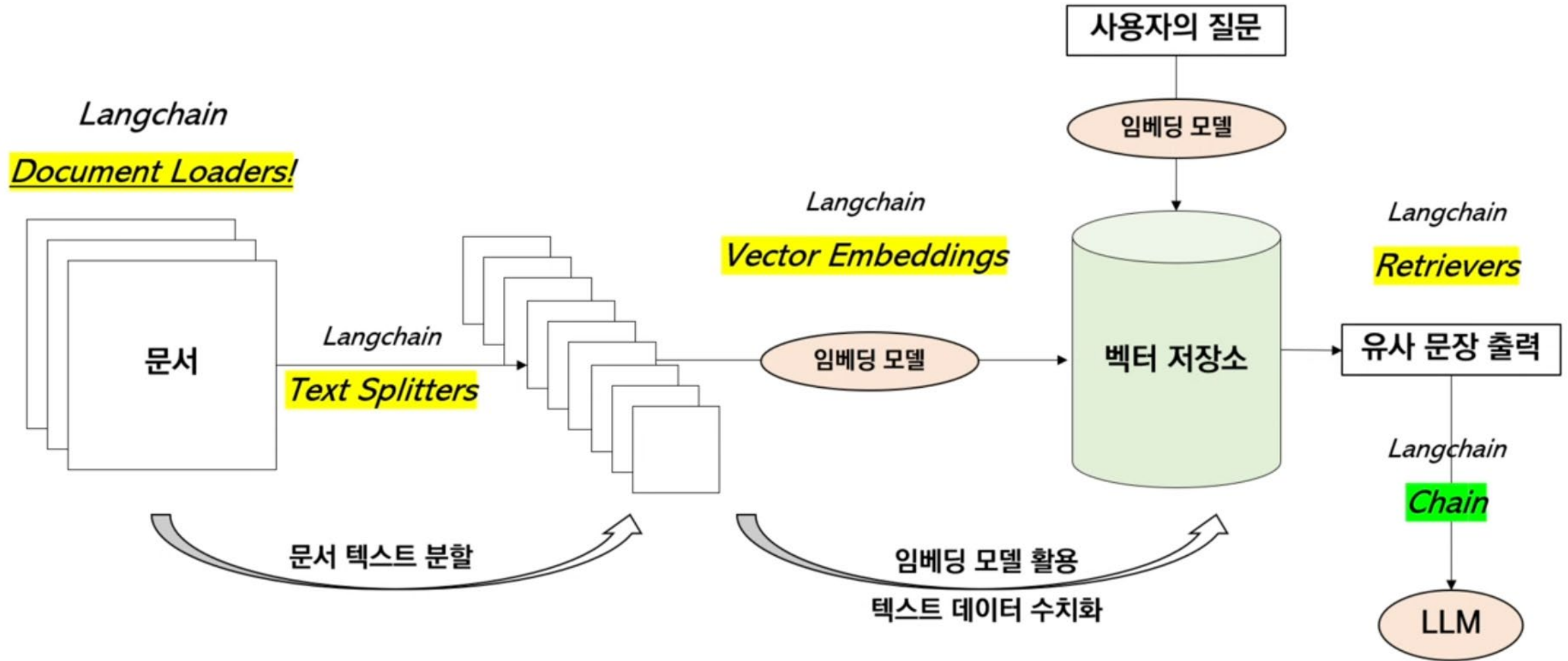
...



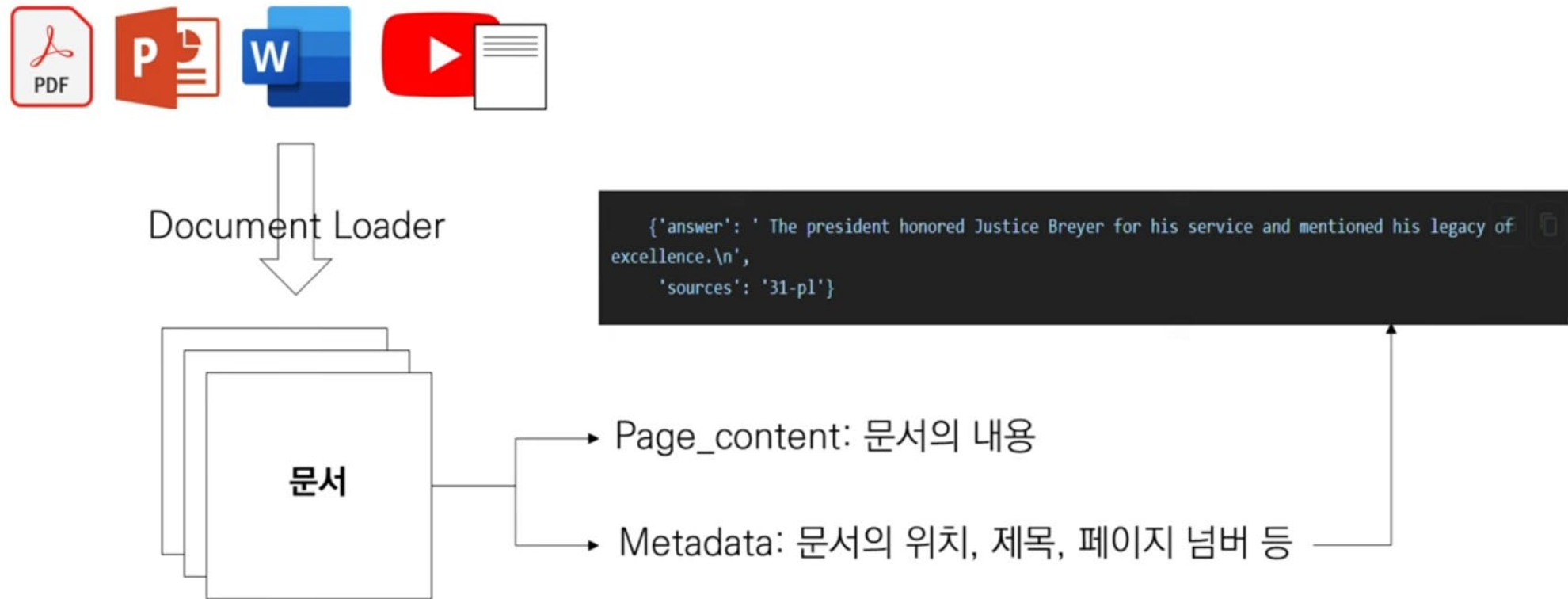
## 04. RAG (Retrieval Augmented Generation)



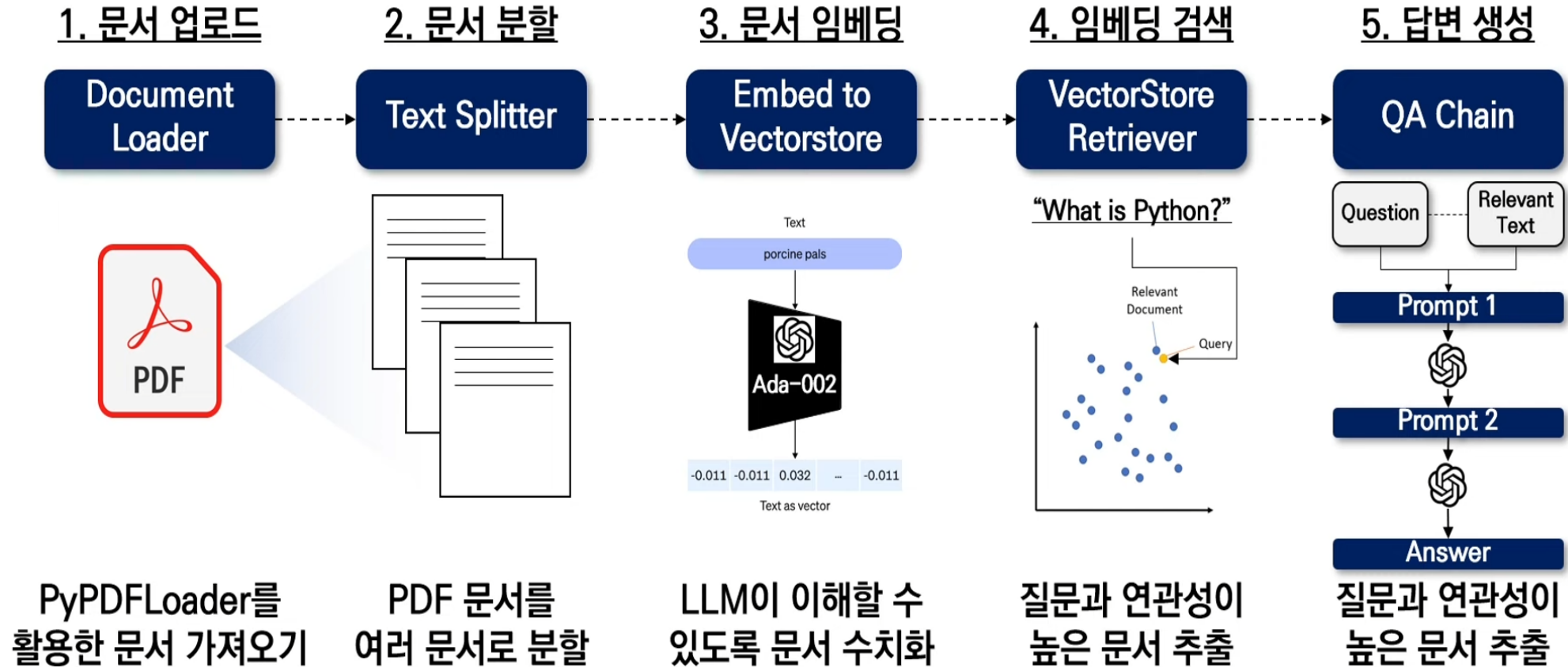
## 04. RAG (Retrieval Augmented Generation)



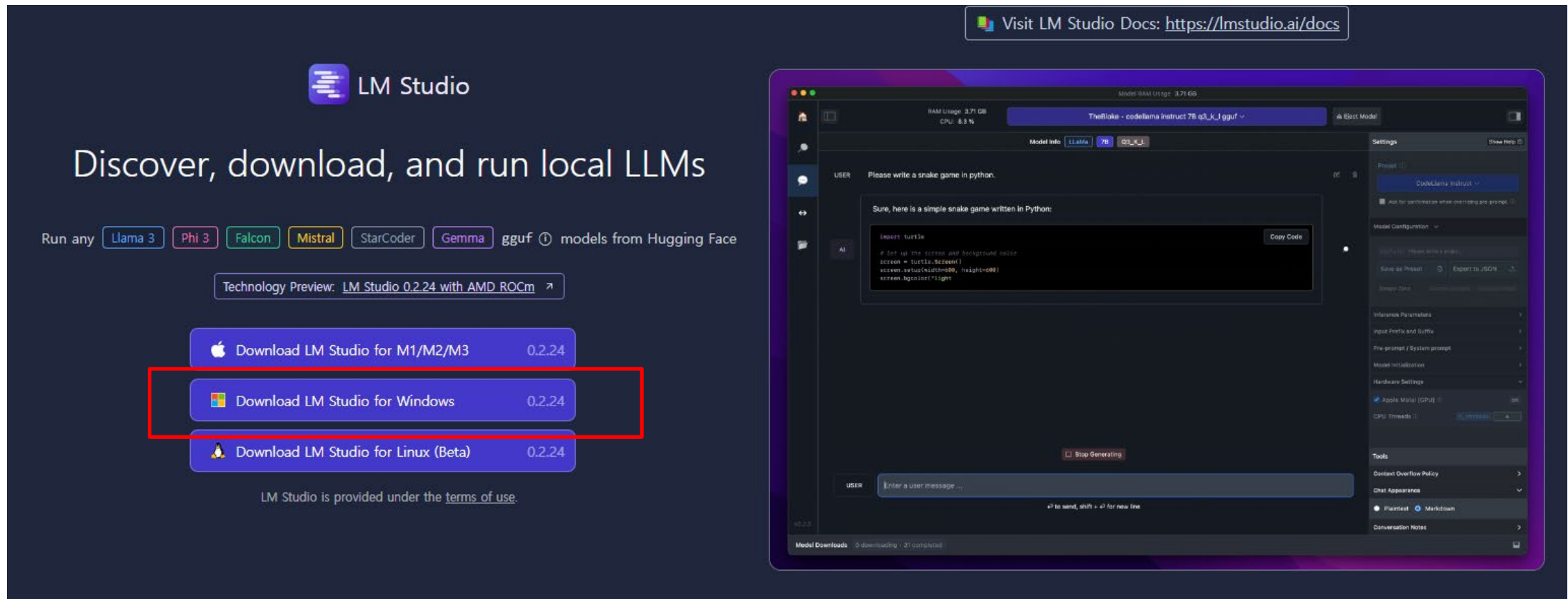
## 04. RAG (Retrieval Augmented Generation)



## 05. PDF 챗봇 구축 전략



## 06. LM Studio 설치 및 환경 설정

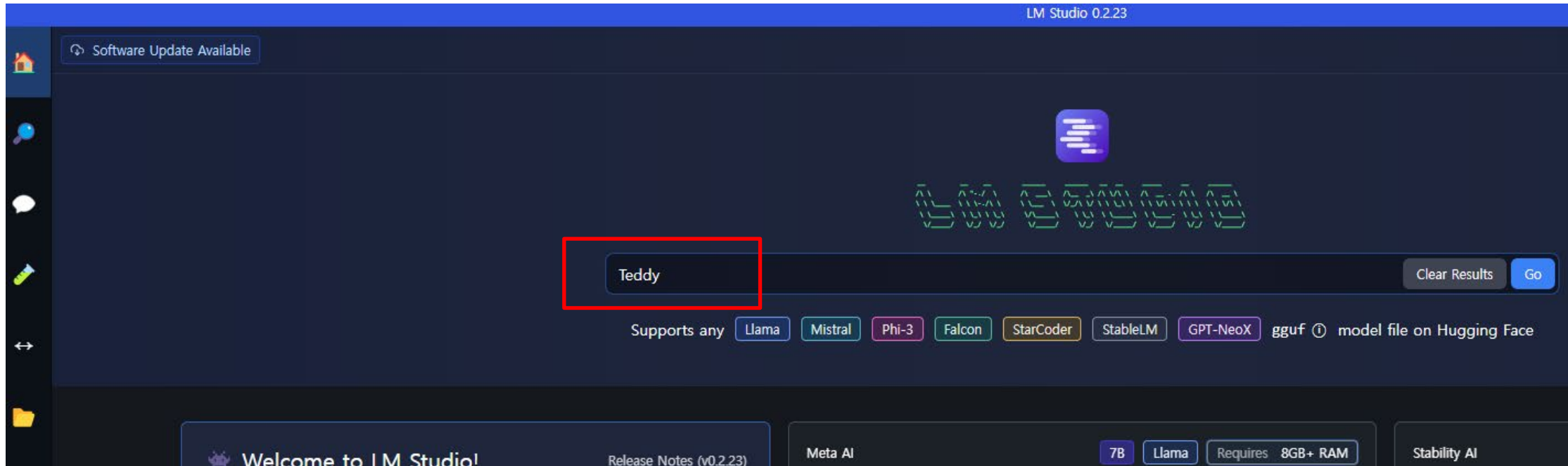


The image displays the LM Studio website and its application interface. The website on the left features the LM Studio logo and the text "Discover, download, and run local LLMs". It lists various models available for download: Llama 3, Phi 3, Falcon, Mistral, StarCoder, and Gemma, all in gguf format from Hugging Face. A "Technology Preview: LM Studio 0.2.24 with AMD ROCm" button is also present. Below this, three download buttons are shown for different operating systems: "Download LM Studio for M1/M2/M3", "Download LM Studio for Windows" (highlighted with a red box), and "Download LM Studio for Linux (Beta)". All three buttons show the version "0.2.24". At the bottom of the website, it states "LM Studio is provided under the terms of use."

The application interface on the right shows the LM Studio desktop application. It features a dark theme and a sidebar on the left with icons for home, chat, and settings. The main area displays a chat conversation. The user's message is "Please write a snake game in python." and the AI's response is "Sure, here is a simple snake game written in Python:" followed by a code block containing Python code for a snake game using the turtle module. A "Copy Code" button is next to the code. The bottom of the interface has a text input field for the user and a "Stop Generating" button. The right sidebar contains various settings and tools, including "Model Configuration", "Inference Parameters", "Hardware Settings", and "Tools".

다운로드 링크 [<https://lmstudio.ai/>]

## 06. LM Studio 설치 및 환경 설정



Teddy 검색

## 06. LM Studio 설치 및 환경 설정

The screenshot displays the LM Studio application interface. At the top, the user 'Teddy' is logged in, and system information shows 'Estimated RAM capacity: 31.78 GB' and 'Estimated VRAM capacity: 12.00 GB'. The left sidebar shows search results for 'teddylee777/EEVE-Korean-Instruct-10.8B-v1.0-gguf', which is highlighted in blue and enclosed in a red box. The right panel shows the details for this model, including a list of 11 available files with download buttons and compatibility information.

teddylee777/EEVE-Korean-Instruct-10.8B-v1.0-gguf

GGUF 8B Architecture Llama

Open Model Card in Browser

README.md

11 Available Files Filter by: Compatibility Guess Show All

File Name	Model File	Compatibility	Size	Action
EEVE-Korean-Instruct-10.8B-v1.0-Q4_0.gguf	MODEL FILE Q4_0	FULL GPU OFFLOAD POSSIBLE	6.12 GB	Download
EEVE-Korean-Instruct-10.8B-v1.0-Q4_K_S.gguf	MODEL FILE Q4_K_S	FULL GPU OFFLOAD POSSIBLE	6.17 GB	Download
EEVE-Korean-Instruct-10.8B-v1.0-Q4_K_M.gguf	MODEL FILE Q4_K_M	FULL GPU OFFLOAD POSSIBLE	6.51 GB	Download
EEVE-Korean-Instruct-10.8B-v1.0-Q4_1.gguf	MODEL FILE Q4_1	FULL GPU OFFLOAD POSSIBLE	6.79 GB	Download
EEVE-Korean-Instruct-10.8B-v1.0-Q5_0.gguf	MODEL FILE Q5_0	FULL GPU OFFLOAD POSSIBLE	7.45 GB	Download
EEVE-Korean-Instruct-10.8B-v1.0-Q5_K_S.gguf	MODEL FILE Q5_K_S	FULL GPU OFFLOAD POSSIBLE	7.45 GB	Download
EEVE-Korean-Instruct-10.8B-v1.0-Q5_K_M.gguf	MODEL FILE Q5_K_M	FULL GPU OFFLOAD POSSIBLE	7.65 GB	Download

EEVE-Korean-Instruct-10.8B-v1.0-gguf 선택

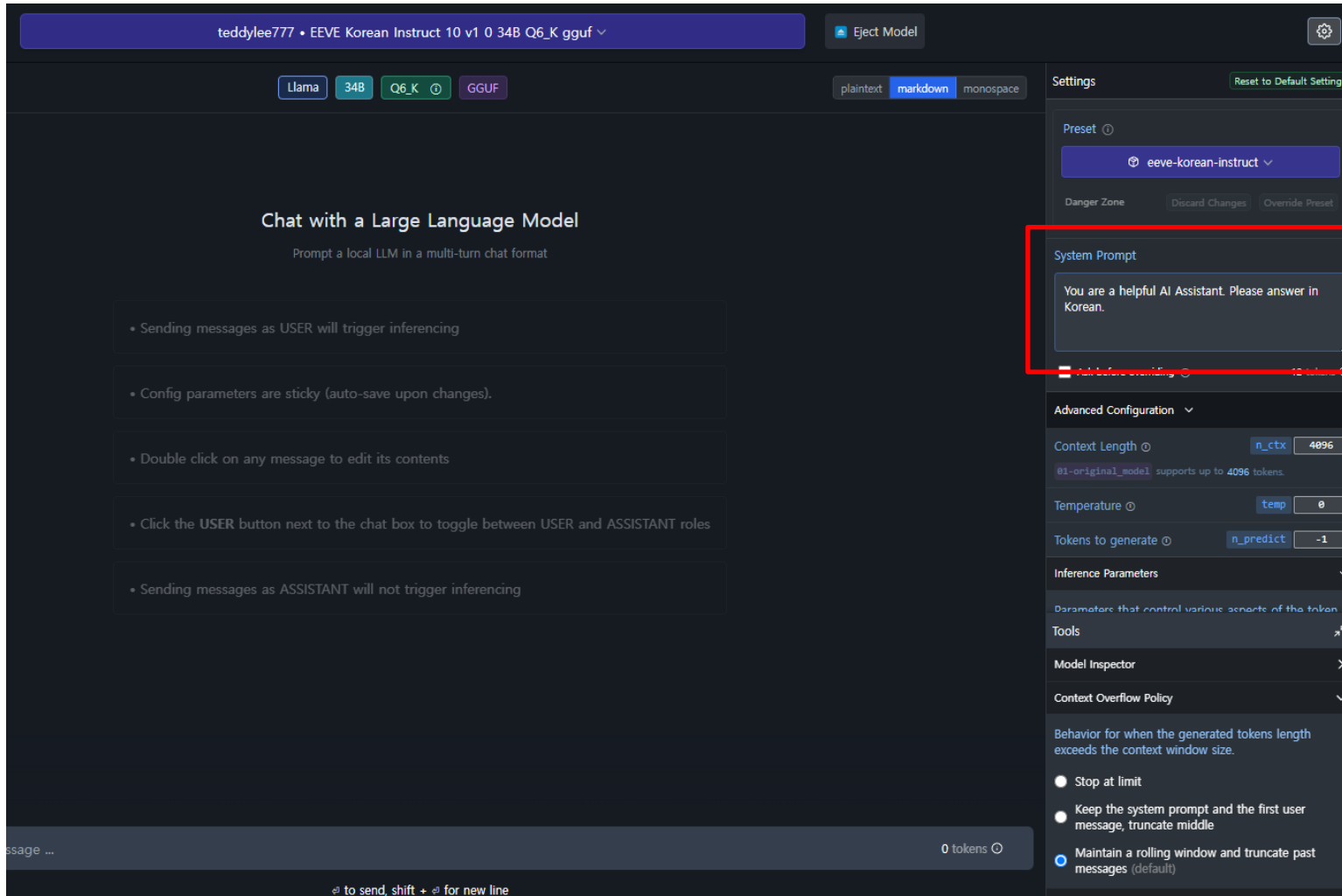
## 06. LM Studio 설치 및 환경 설정



EEVE-Korean-Instruct-10.8B-v1.0-gguf 선택



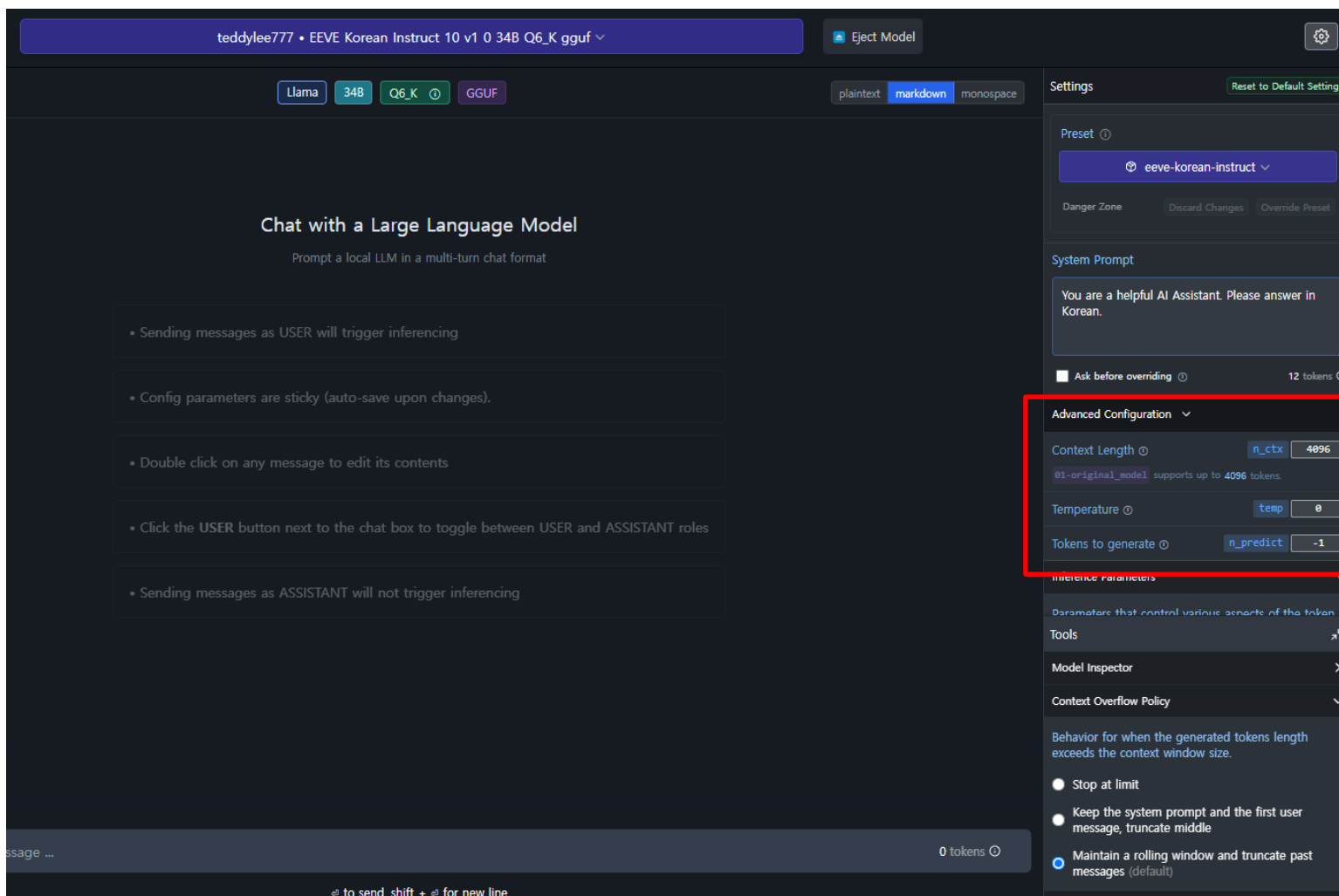
## 06. LM Studio 설치 및 환경 설정



### [System Prompt 내용]

You are a helpful AI Assistant.  
Please answer in Korean.

## 06. LM Studio 설치 및 환경 설정



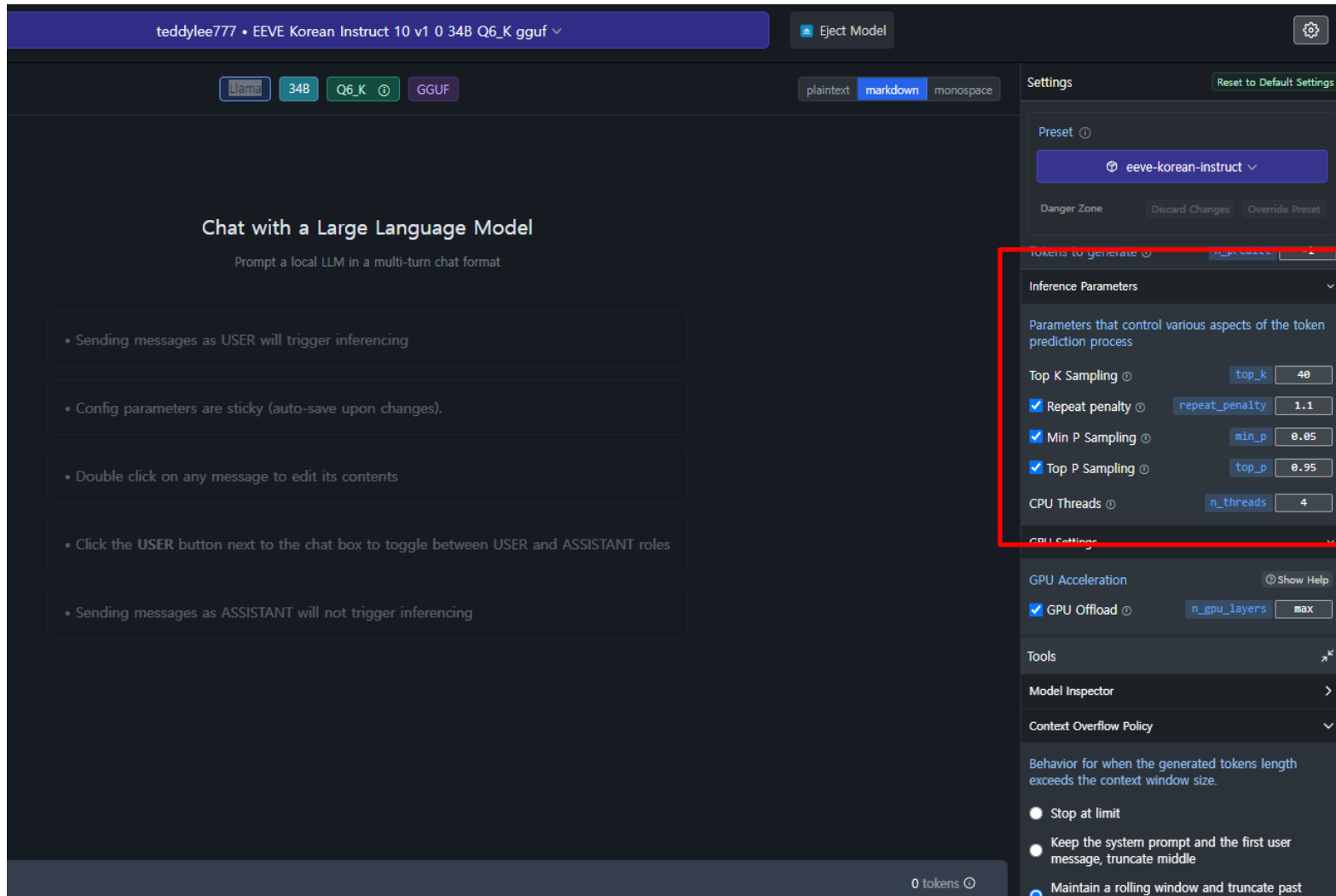
### [Advanced Configuration]

Context Length: 4096

Temperature: 0

Tokens to generate: -1

## 06. LM Studio 설치 및 환경 설정

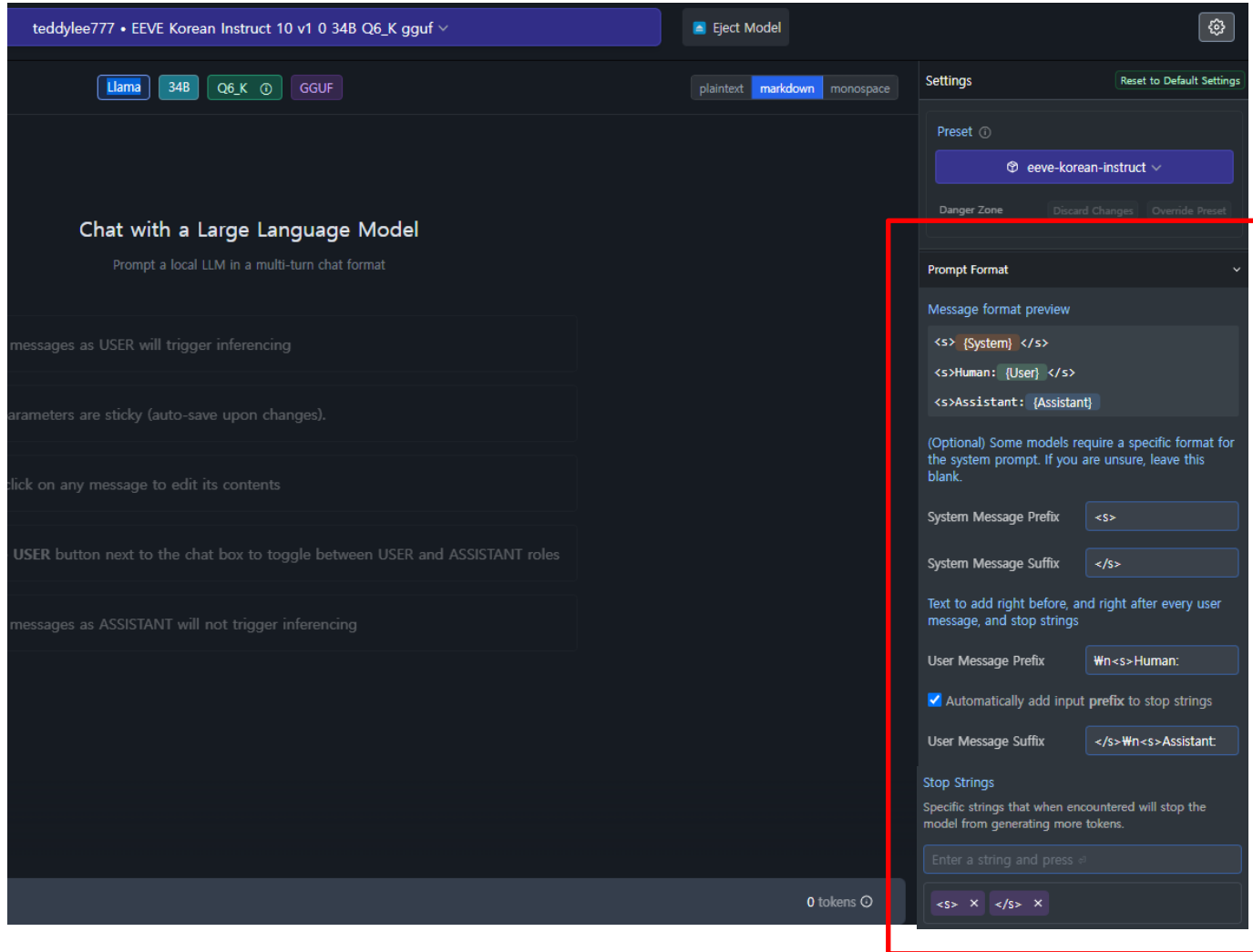


### [Inference Parameters]

기본 설정 값 사용

필요에 따라 조정해서  
사용하기

## 06. LM Studio 설치 및 환경 설정



### [Prompt Format]

System Message Prefix: <s>

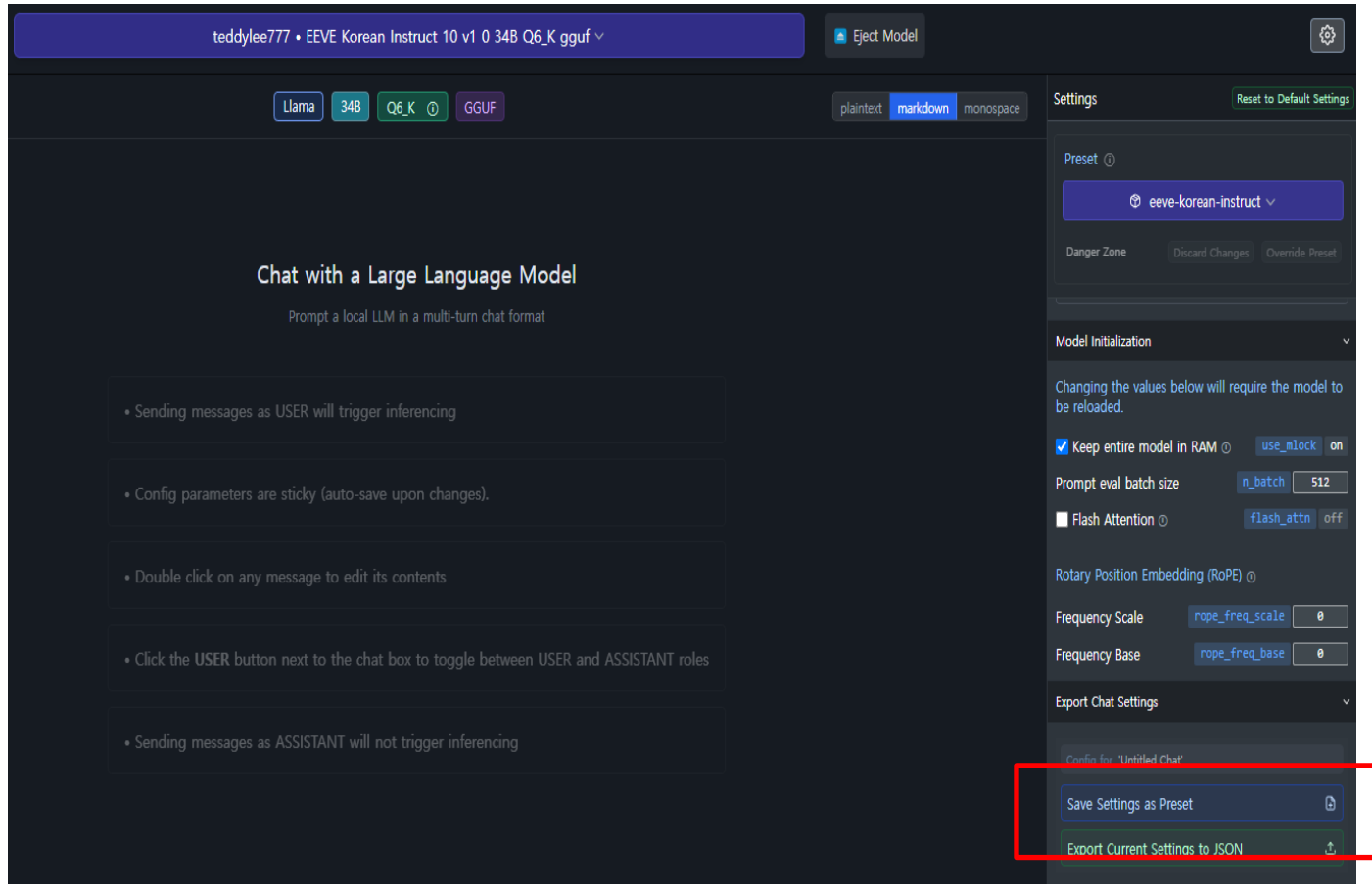
System Message Suffix: </s>

User Message Prefix: \n<s>Human:

User Message Suffix: </s>\n<s>Assistant:

Stop Strings: <s>, </s>

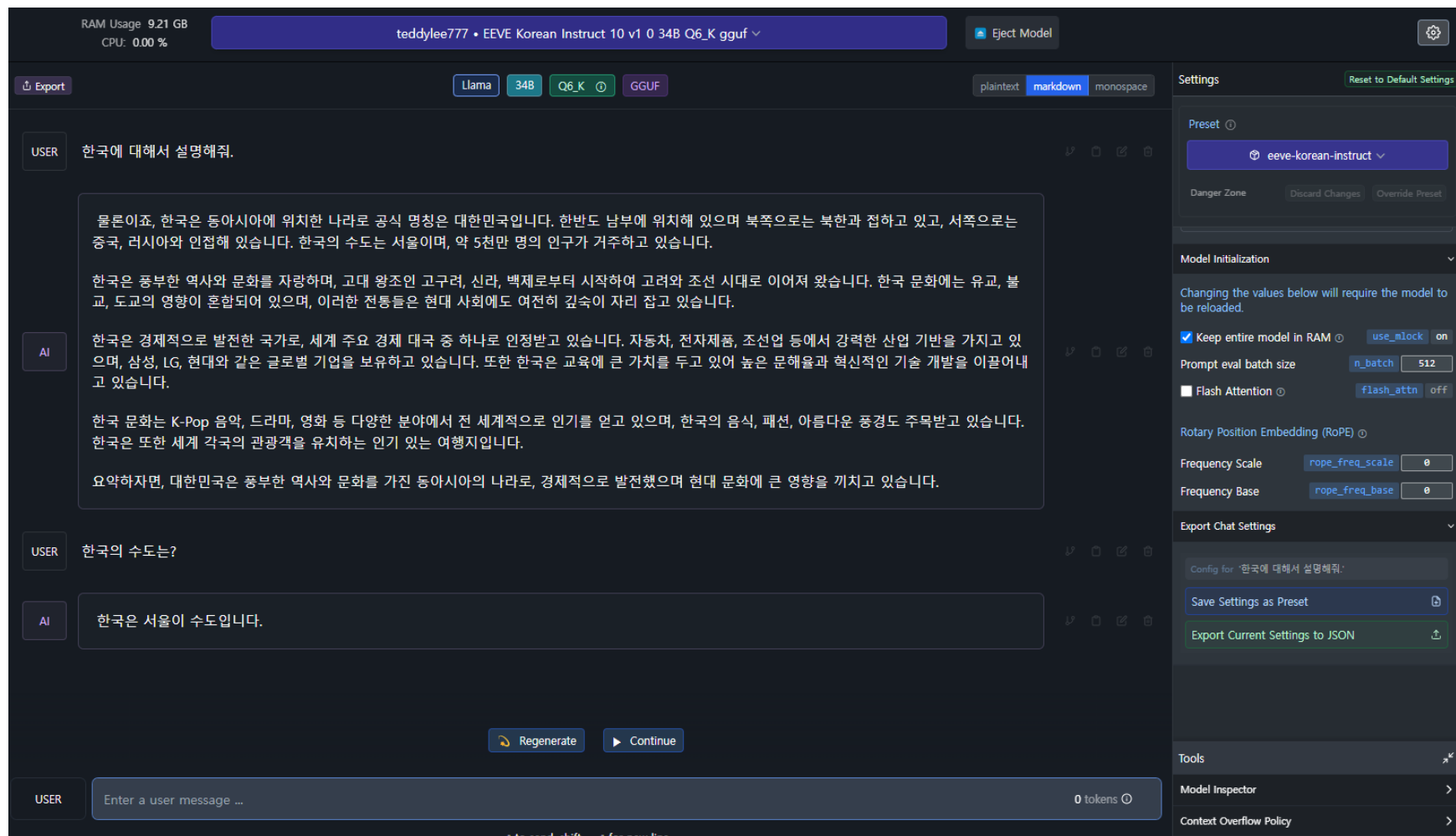
## 06. LM Studio 설치 및 환경 설정



### [Save Settings as Preset]

Preset 설정 저장하기

## 06. LM Studio 설치 및 환경 설정



## 06. LM Studio 설치 및 환경 설정

The screenshot displays the LM Studio application interface. At the top, a status bar shows RAM Usage at 9.21 GB and CPU usage at 0.00 %. A red box highlights the model selection dropdown, which is set to "teddylee777 • EEVE Korean Instruct 10 v1 0 34B Q6\_K gguf". Below this, the "Local Inference Server" section provides instructions on starting a local HTTP server that mimics OpenAI API endpoints. It lists supported endpoints: /v1/chat/completions, /v1/embeddings, and /v1/models. A red box highlights the "Configuration" tab in the left sidebar. The main configuration area shows the "Server Port" set to 1234 and several toggle switches for "Cross-Origin-Resource-Sharing (CORS)", "Request Queuing", "Verbose Server Logs", and "Apply Prompt Formatting", all of which are currently turned ON. A red box highlights the "Start Server" button. The "Embedding Model Settings" section is also visible, showing the "nomic-ai/nomic-embed-text-v1.5" model. The "Examples" section contains a code block for using the OpenAI client library. The right sidebar shows "Server Model Settings" with a red box highlighting the "Preset" dropdown, which is set to "Llama-3-ko". Other settings like "System Prompt", "Context Length", "Temperature", and "Tokens to generate" are also visible.

RAM Usage 9.21 GB  
CPU: 0.00 %

teddylee777 • EEVE Korean Instruct 10 v1 0 34B Q6\_K gguf

Eject Model

Community

### Local Inference Server

Start a local HTTP server that mimics select OpenAI API endpoints.

Supported endpoints: /v1/chat/completions, /v1/embeddings, /v1/models

API Documentation: [Local Server API](#) (↗ open in browser)

### Configuration

Server Port: 1234

Cross-Origin-Resource-Sharing (CORS) ☒ ON ☐ OFF

Request Queuing ☒ ON ☐ OFF

Verbose Server Logs ☒ ON ☐ OFF

Apply Prompt Formatting ☒ ON ☐ OFF

Reload to Apply Changes ↻

Start Server

Stop Server

### Embedding Model Settings

Learn More About Embeddings ↗

Load a text embedding model and utilize it through the `POST /v1/embeddings` endpoint.

nomic-ai/nomic-embed-text-v1.5  
A recent high-performing text embedding model by Nomic AI (apache-2.0)

Download (146.15 MB)

### Examples

hello world (curl) chat (python) ai assistant (python) vision (python) embeddings (python)

```
# Example: reuse your existing OpenAI setup
from openai import OpenAI

# Point to the local server
client = OpenAI(base_url="http://localhost:1234/v1", api_key="lm-studio")

completion = client.chat.completions.create(
    model="teddylee777/EEVE-Korean-Instruct-10.8B-v1.0-gguf",
    messages=[
        {"role": "system", "content": "Always answer in rhymes."},
        {"role": "user", "content": "Introduce yourself."}
    ]
)
```

Copy Code

### Server Model Settings

Reset to Default Settings

Preset ☒ Llama-3-ko 11 changes

System Prompt

You are a helpful AI Assistant. Please answer in Korean.

☒ Ask before overriding 12 tokens

### Advanced Configuration

Context Length ☒ n\_ctx 8192  
#1-original\_model supports up to 4096 tokens.

Temperature ☒ temp 0

Tokens to generate ☒ n\_predict -1

### GPU Settings

Prompt Format

### Message format preview

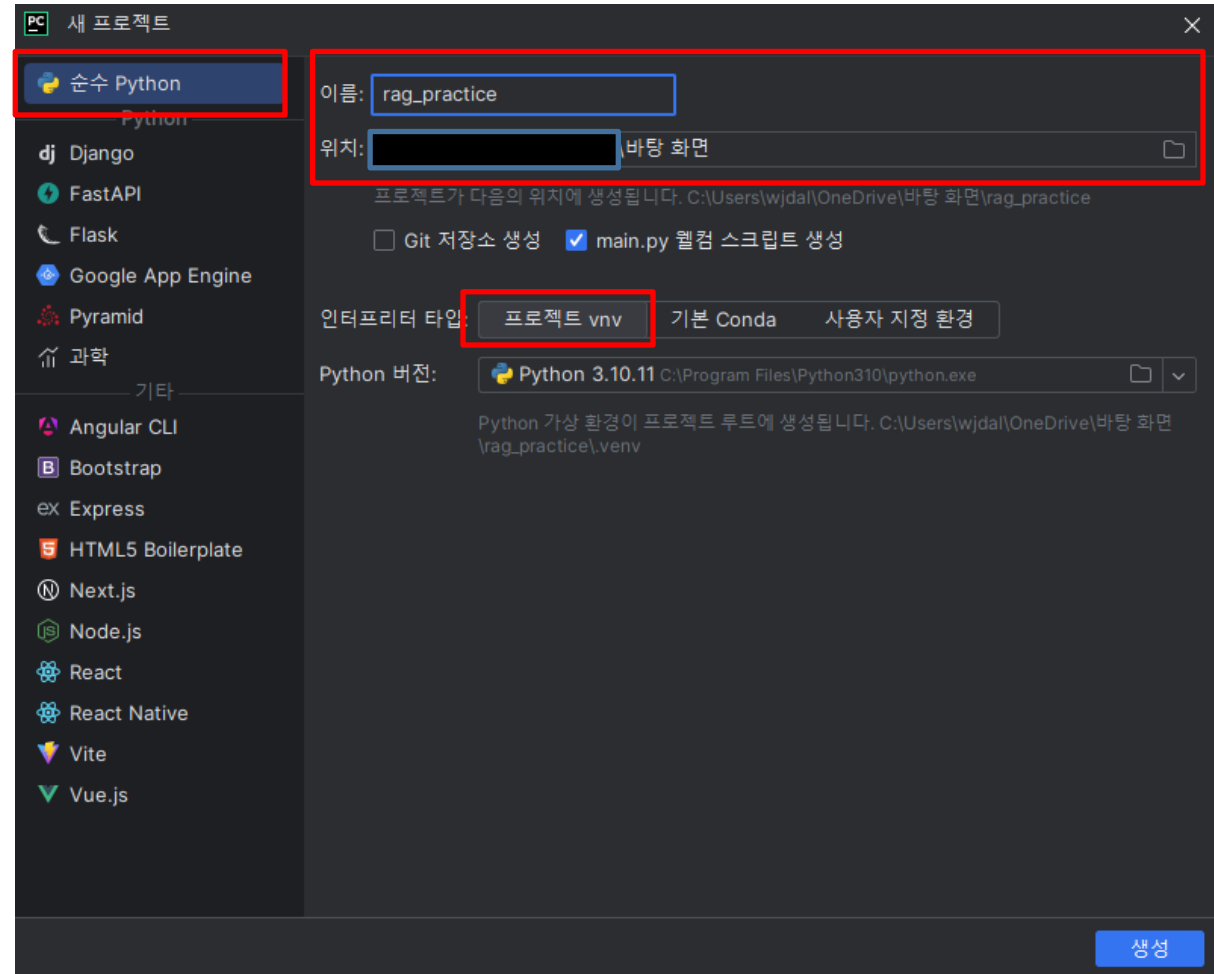
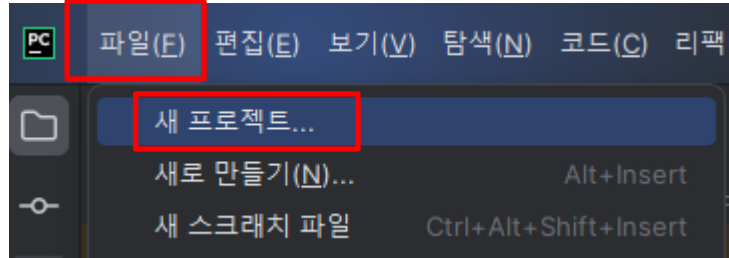
```
<s> [System] </s>
<s>Human: [User] </s>
<s>Assistant: [Assistant]
```

(Optional) Some models require a specific format for the system prompt. If you are unsure, leave this blank.

Server logs Server not running (logs are saved into /tmp/lmstudio-server-log.txt)

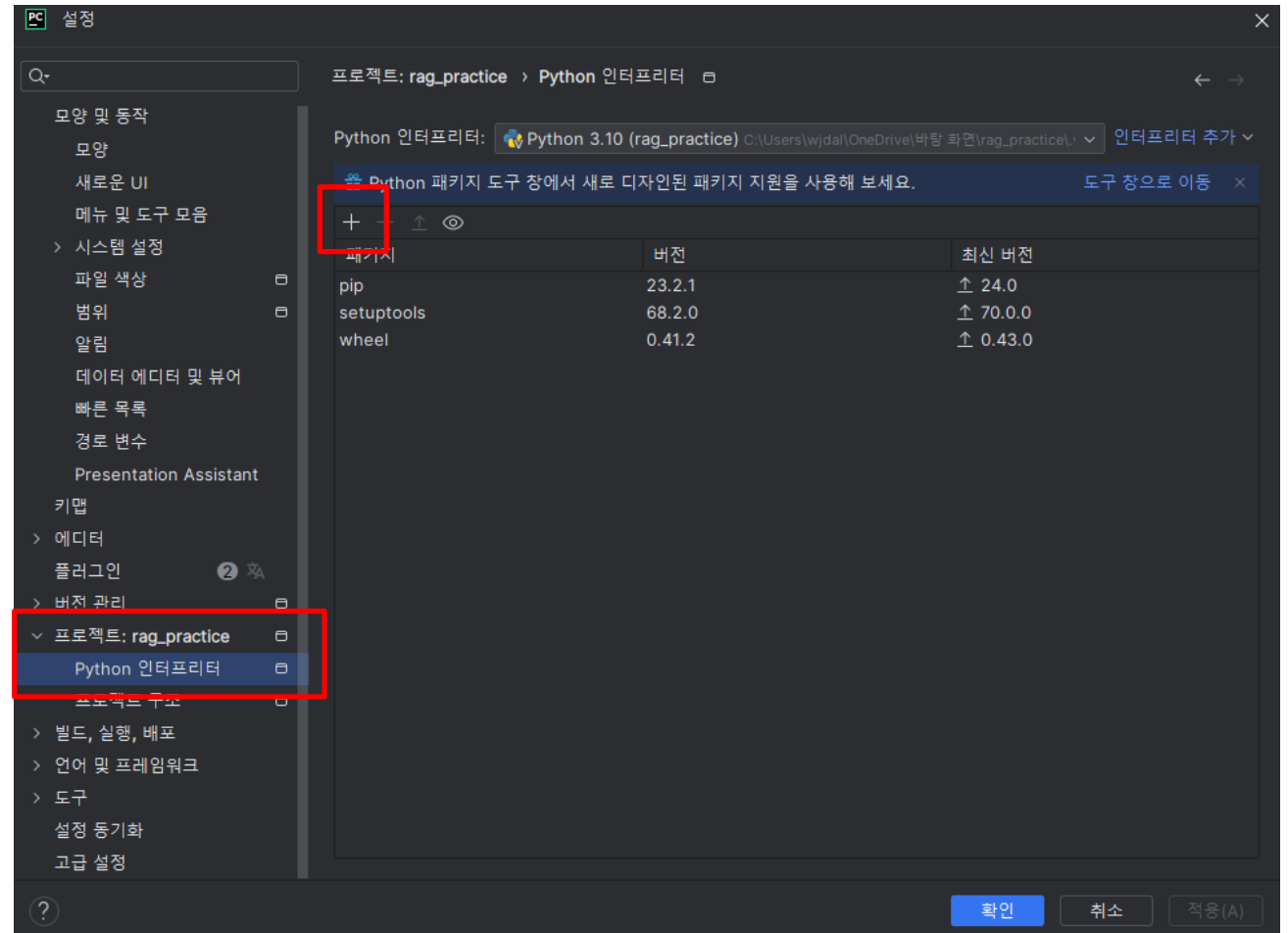
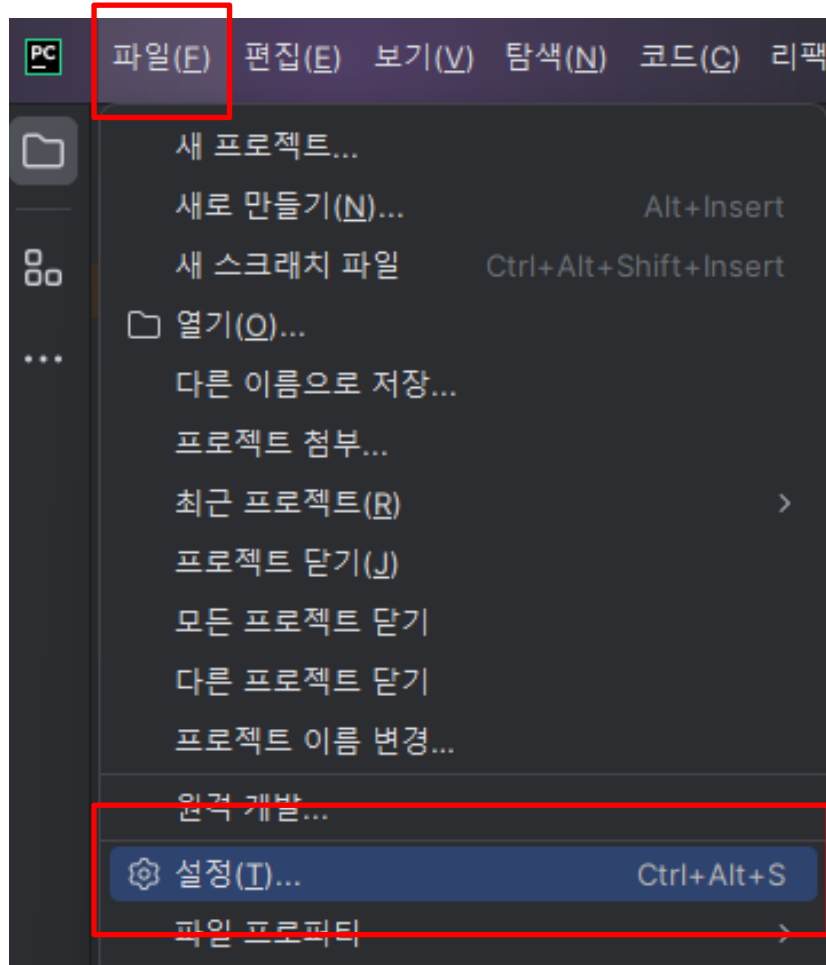
Filter logs... Open Logs Clear (Ctrl+K)

## 07. 실습 (LangChain – PDF 챗봇 구축 )

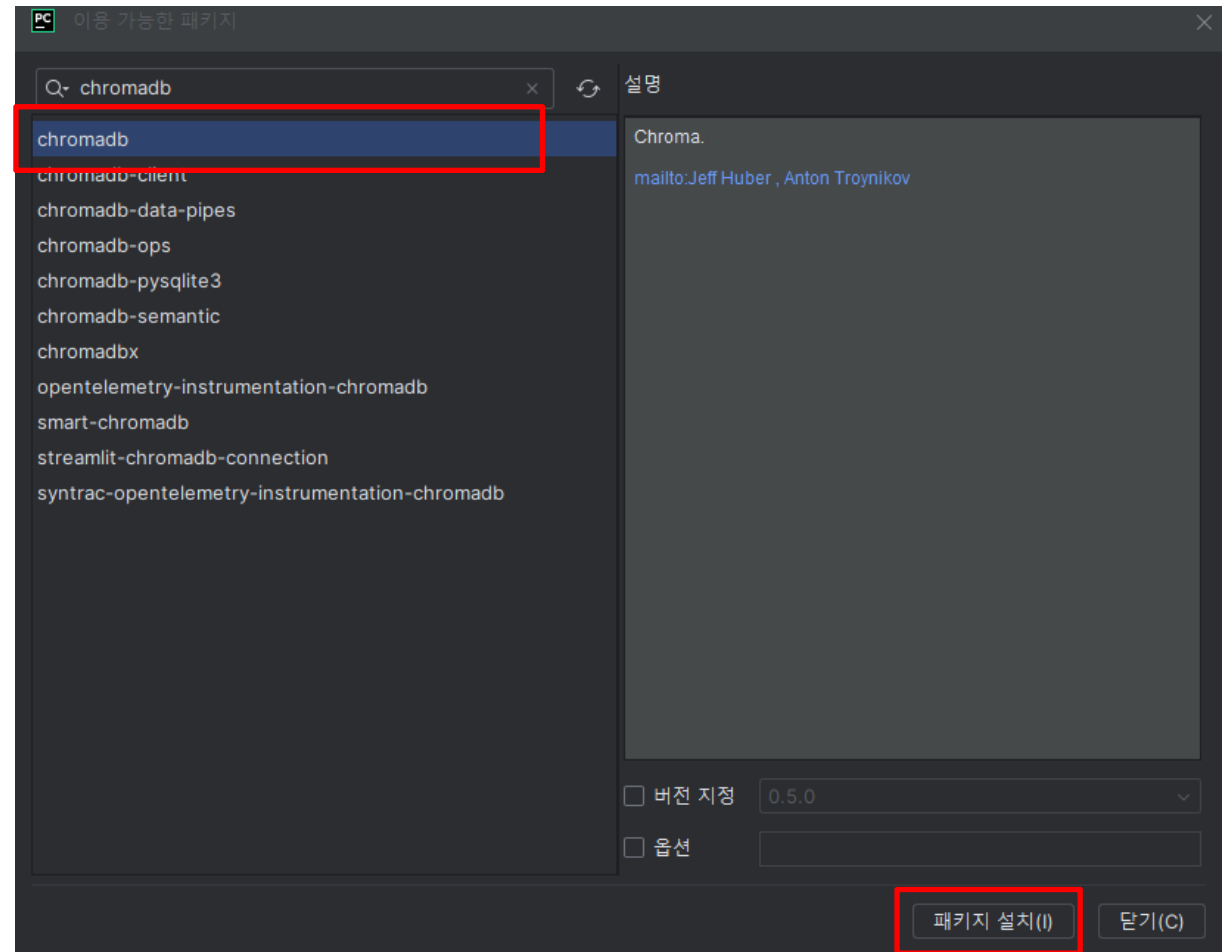
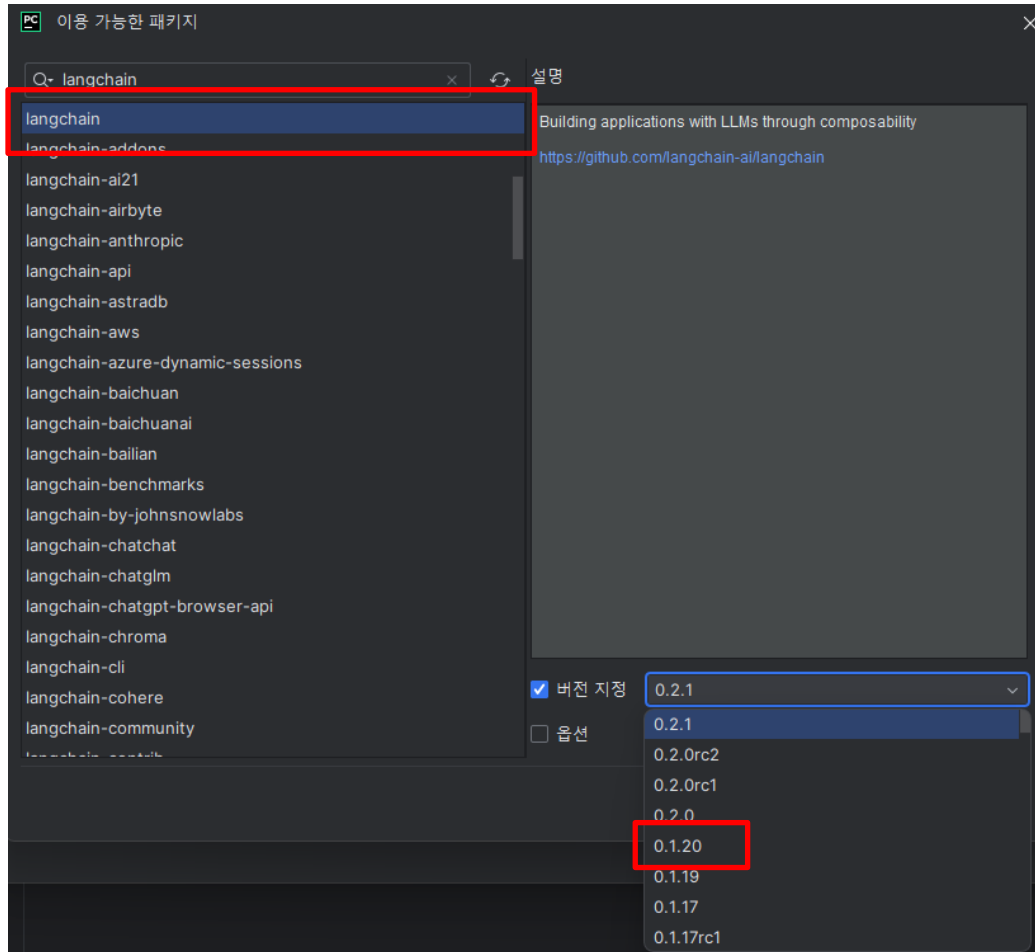




## 07. 실습 (LangChain – PDF 챗봇 구축 )



## 07. 실습 (LangChain – PDF 챗봇 구축 )



## 07. 실습 (LangChain – PDF 챗봇 구축 )

---

### [라이브러리]

`langchain==0.1.20`

`pypdf==3.16.2`

`chromadb==0.5.0`

`pdf2image==1.16.3`

`load-dotenv==0.1.0`

`pdfminer==20191125`

`openai==1.30.5`

`tiktoken==0.7.0`

`sentence-transformers==3.0.0`

---

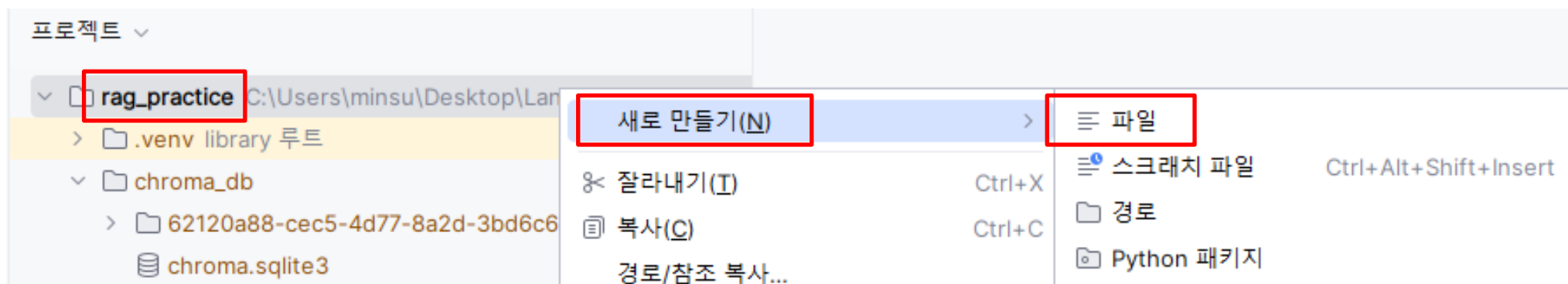
## 07. 실습 (LangChain – PDF 챗봇 구축 )

### [말뭉치 저장 경로 추가]

경로 이름: **Corpus**

PDF 파일 다운로드 링크: <https://m.site.naver.com/1pZ7F>

치과교정용 스마트 페이스마스크를 활용한 스마트 교정 관리.pdf 파일 추가

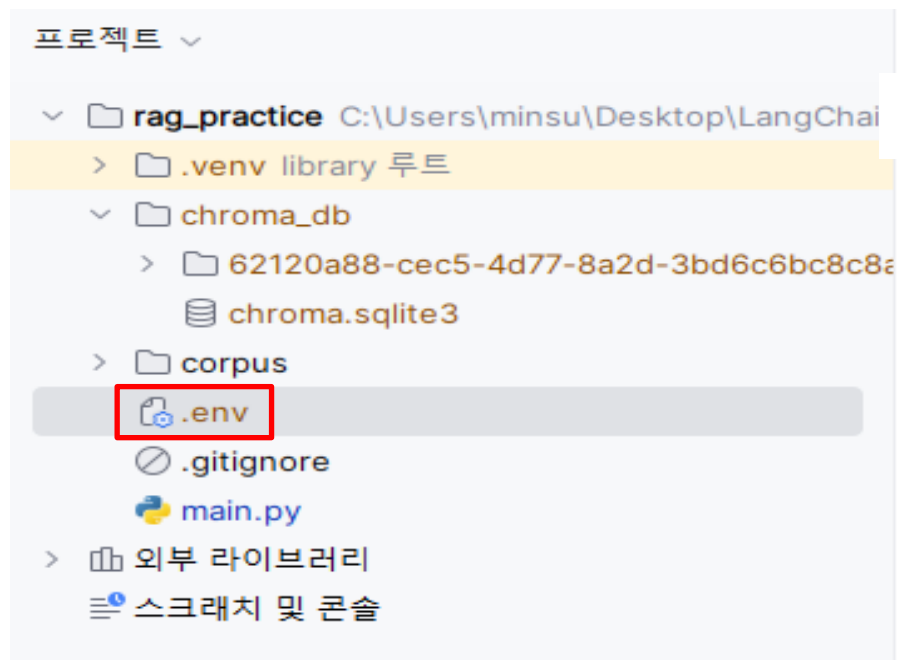
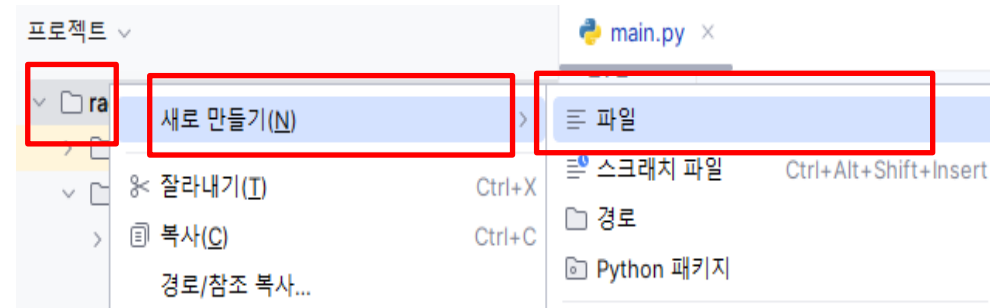


## 07. 실습 (LangChain – PDF 챗봇 구축 )

### [.env 파일 내용]

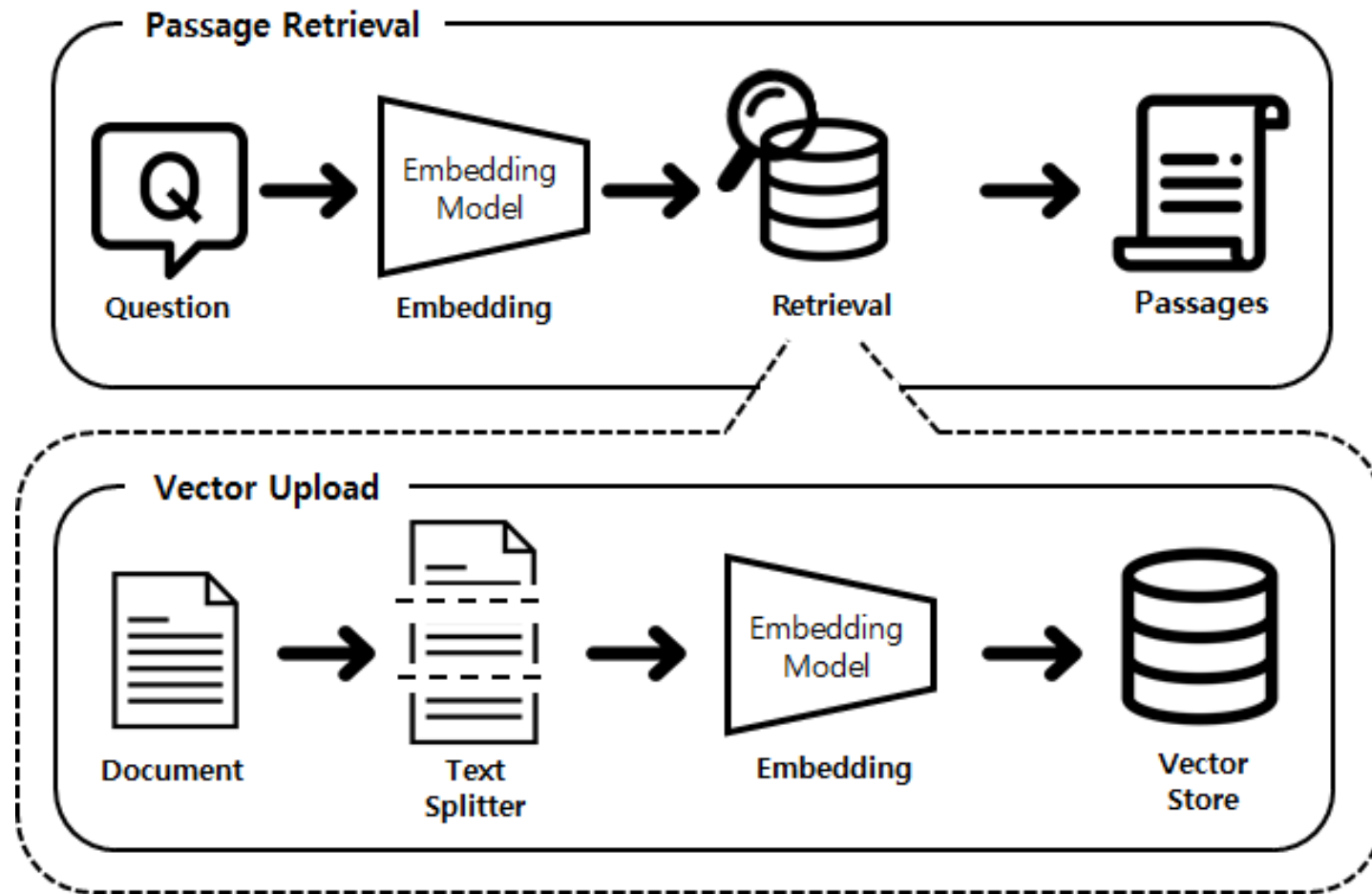
파일 이름: .env

LM\_LOCAL\_URL="http://localhost:1234/v1"



```
1 OPENAI_API_KEY="본인 API KEY 입력하기"
```

## 07. 실습 (LangChain – PDF 챗봇 구축 )



## 07. 실습 (LangChain – PDF 챗봇 구축 )

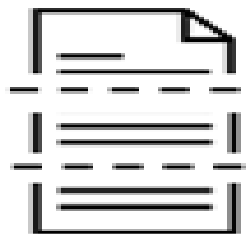
---



Document

```
1  def docs_load():
2      """
3      문서 읽는 함수
4      """
5
6      from langchain.document_loaders import PyPDFLoader
7
8      loader = PyPDFLoader("corpus/치과교정용 스마트 페이스마스크를 활용한 스마트 교정 관리.pdf").load()
9
10     print(loader)
11
12     return loader
```

## 07. 실습 (LangChain – PDF 챗봇 구축 )



**Text  
Splitter**

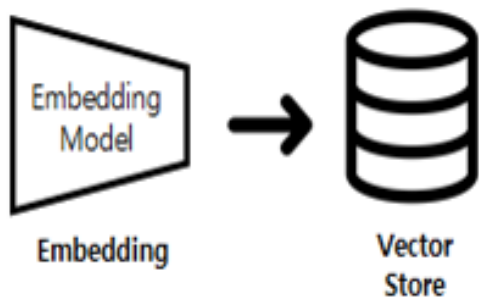
```

15  ▼ def rc_text_split(corpus):
16  ▼     """
17      CharacterTextSplitter를 사용하여 문서를 분할하도록 하는 함수
18      :param corpus: 전처리 완료된 말뭉치
19      :return: 분리된 청크
20      """
21
22      from langchain.text_splitter import RecursiveCharacterTextSplitter
23
24      rc_text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
25          separators=["\n\n", "\n", " ", ""],
26          chunk_size=2000,
27          chunk_overlap=500,
28          model_name="gpt-4o" # o200k_base
29      )
30
31      text_documents = rc_text_splitter.split_documents(corpus)
32
33      return text_documents

```



## 07. 실습 (LangChain – PDF 챗봇 구축 )

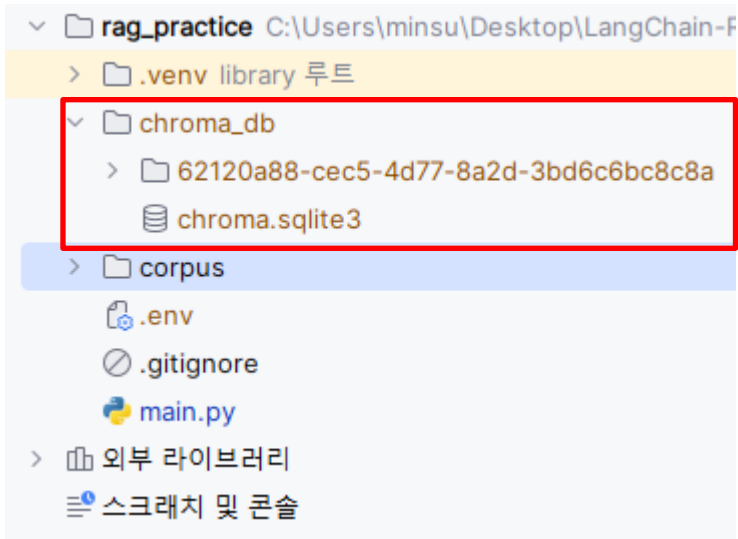
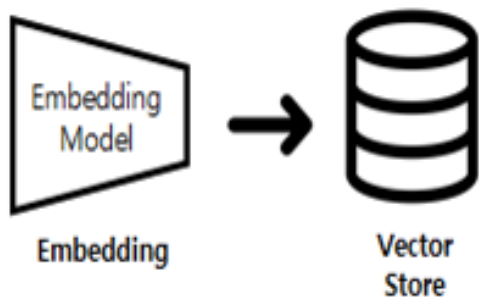


```

36 def embedding_model():
37     """
38     문서 임베딩 모델을 생성하는 함수
39     :return: 허깅페이스 임베딩 모델
40     """
41
42     from langchain.embeddings import HuggingFaceEmbeddings
43
44     model_name = "jhgan/ko-sroberta-multitask" # 한국어 모델
45     model_kwargs = {'device': 'cpu'} # cpu를 사용하기 위해 설정
46     encode_kwargs = {'normalize_embeddings': True}
47     model = HuggingFaceEmbeddings(
48         model_name=model_name,
49         model_kwargs=model_kwargs,
50         encode_kwargs=encode_kwargs
51     )
52
53     return model

```

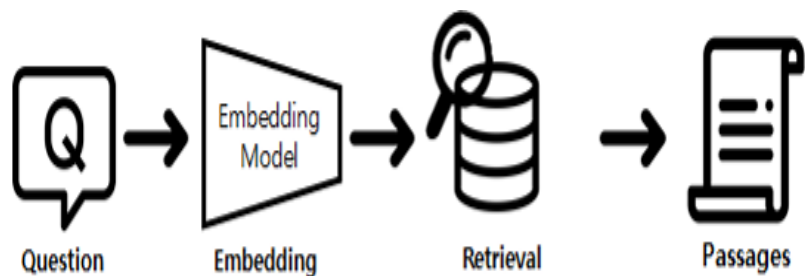
## 07. 실습 (LangChain – PDF 챗봇 구축 )



```

56 def document_embedding(docs, model, save_directory):
57     """
58     Embedding 모델을 사용하여 문서 임베딩하여 Chroma 벡터저장소(VectorStore)에 저장하는 함수
59     :param model: 임베딩 모델 종류
60     :param save_directory: 벡터저장소 저장 경로
61     :param docs: 분할된 문서
62     :return:
63     """
64
65     from langchain_community.vectorstores import Chroma
66     import os
67     import shutil
68
69     print("\n잠시만 기다려주세요.\n\n")
70
71     # 벡터저장소가 이미 존재하는지 확인
72     if os.path.exists(save_directory):
73         shutil.rmtree(save_directory)
74         print(f"디렉토리 {save_directory}가 삭제되었습니다.\n")
75
76     print("문서 벡터화를 시작합니다. ")
77     db = Chroma.from_documents(docs, model, persist_directory=save_directory)
78     print("새로운 Chroma 데이터베이스가 생성되었습니다.\n")
79
80     return db
  
```

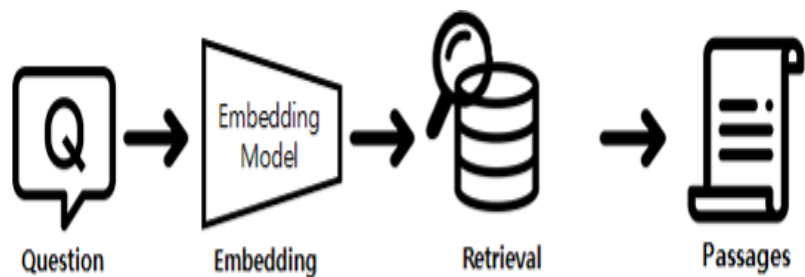
## 07. 실습 (LangChain – PDF 챗봇 구축 )



```

83 def chat_llm():
84     """
85     채팅에 사용되는 거대언어모델 생성 함수
86     :return: 답변해주는 거대언어모델
87     """
88
89     import os
90     from dotenv import load_dotenv
91     from langchain.chat_models import ChatOpenAI
92     from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
93
94     load_dotenv('.env')
95
96     # OpenAI API 호출을 통해 구동 시 사용
97     llm = ChatOpenAI(
98         model="gpt-4o-mini",
99         api_key=os.getenv("OPENAI_API_KEY"),
100         temperature=0,
101         streaming=True,
102         callbacks=[StreamingStdOutCallbackHandler()],
103     )
104
105     return llm
  
```

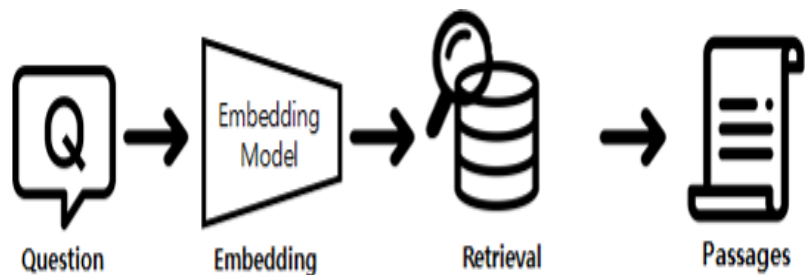
## 07. 실습 (LangChain – PDF 챗봇 구축 )



```

109  def qna(llm, db):
110      qna_result = []
111
112      check = 'Y'  # 0이면 질문 가능
113
114      while check == 'Y' or check == 'y':
115          query = input("질문을 입력하세요 : ")
116          response = db_qna(llm, db, query)  # 기본 검색기
117
118          qna_result.append({'query': query, 'response': response})
119
120          check = input("\n\nY: 계속 질문한다.\nN: 프로그램 종료\n입력: ")
121
122      return qna_result
  
```

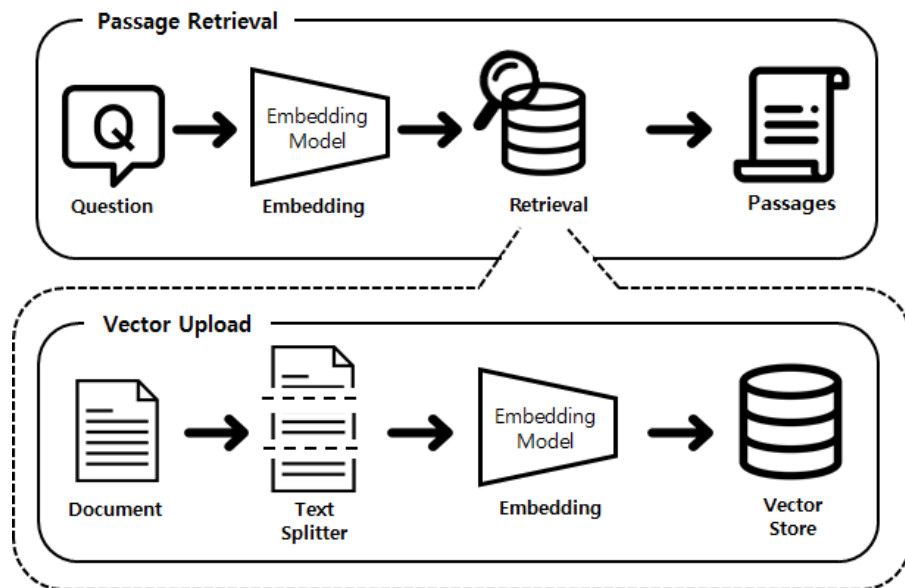
## 07. 실습 (LangChain – PDF 챗봇 구축 )



```

125 def db_qna(llm, db, query, ):
126     """
127     벡터저장소에서 질문을 검색해서 적절한 답변을 찾아서 답하도록 하는 함수
128     :param llm: 거대 언어 모델
129     :param db: 벡터스토어
130     :param query: 사용자 질문
131     :return: 거대언어모델(LLM) 응답 결과
132     """
133
134     from langchain.prompts import ChatPromptTemplate
135     from langchain.schema.runnable import RunnableLambda, RunnablePassthrough
136     from langchain_core.output_parsers import StrOutputParser
137
138     db = db.as_retriever(
139         search_type="mmr",
140         search_kwargs={'k': 3, 'fetch_k': 5}
141     )
142     prompt = ChatPromptTemplate.from_messages(
143         [
144             (
145                 "system",
146                 """
147                 You are a specialized AI for question-and-answer tasks.
148                 You must answer questions based solely on the Context provided.
149
150                 Context: {context}
151                 """,
152             ),
153             ("human", "Question: {question}"),
154         ]
155     )
156
157     chain = {
158         "context": db | RunnableLambda(format_docs),
159         "question": RunnablePassthrough()
160     } | prompt | llm | StrOutputParser()
161
162     response = chain.invoke(query)
163
164     return response
165
169 def format_docs(docs):
170     return "\n\n".join(document.page_content for document in docs)
  
```

## 07. 실습 (LangChain – PDF 챗봇 구축 )



```

173 def run():
174     """
175     챗봇 시작
176     Document Load -> Text Splitter -> Document Embedding -> VectorStore save -> QA
177     """
178
179     # 문서 업로드
180     loader = docs_load()
181
182     # 문서 분할
183     chunk = rc_text_split(loader)
184
185     print(chunk)
186
187     # 임베딩 모델 생성
188     model = embedding_model()
189
190     # 문서 임베딩
191     db = document_embedding(chunk, model, save_directory="./chroma_db")
192
193     # 채팅에 사용할 거대언어모델(LLM) 선택
194     llm = chat_llm()
195
196     # 질의응답
197     qna_list = qna(llm, db)
198
199     print(qna_list)
200
201
202 > if __name__ == "__main__":
203     run()
  
```



**Thank You**  
**For Your Attention.**

