LangChain 기반의 RAG 구축 사례

발표일자 2025.10.14

발 표 자 정일구

코메디닷컴

RAG (Retrieval-Augmented Generation) - 검색, 증강, 생성

■ AS-IS

현재의 ChatGPT가 가질 수 있는 문제점

1. **최신 정보**에 대하여 학습되어 있지 않다

2. 개인or회사에 제한되어 있는 **내부데이터**에 대한 **학습이 되어 있지 않다**.

3. 특정도메인에 대한 질문을 하면 기대하는 답변을 얻을 수 없다.

4. 할루시네이션(환각) 현상이 발생한다. 게다가 문서의 양이 많아지면 더욱 심해진다.

■ TO-BE

적합한 RAG를 적용했을 때는

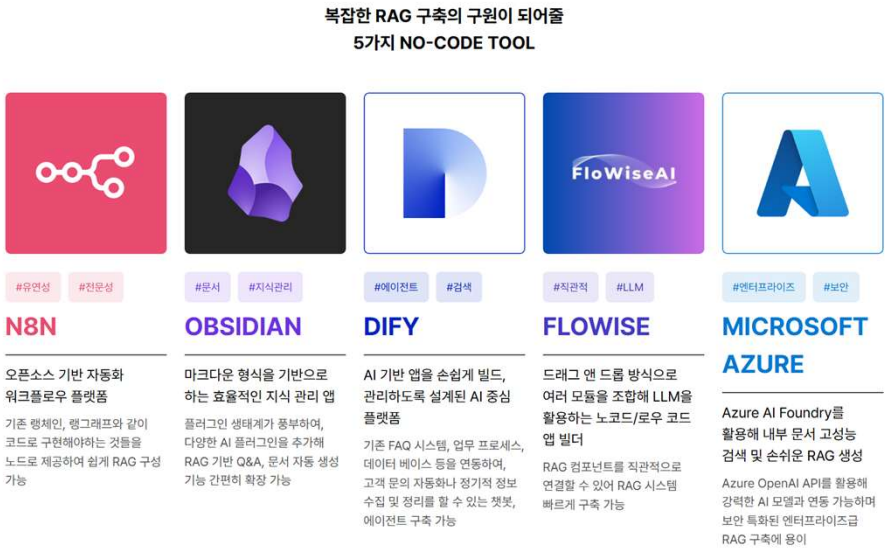
1. 최신 정보를 기반으로 답변할 수 있으며, 정보를 찾을 수 없는 경우 검색기능을 활용하여 답변할 수 있다.

2. 개인or회사에 제한되어 있는 내부데이터를 참고하여 답변할 수 있다.

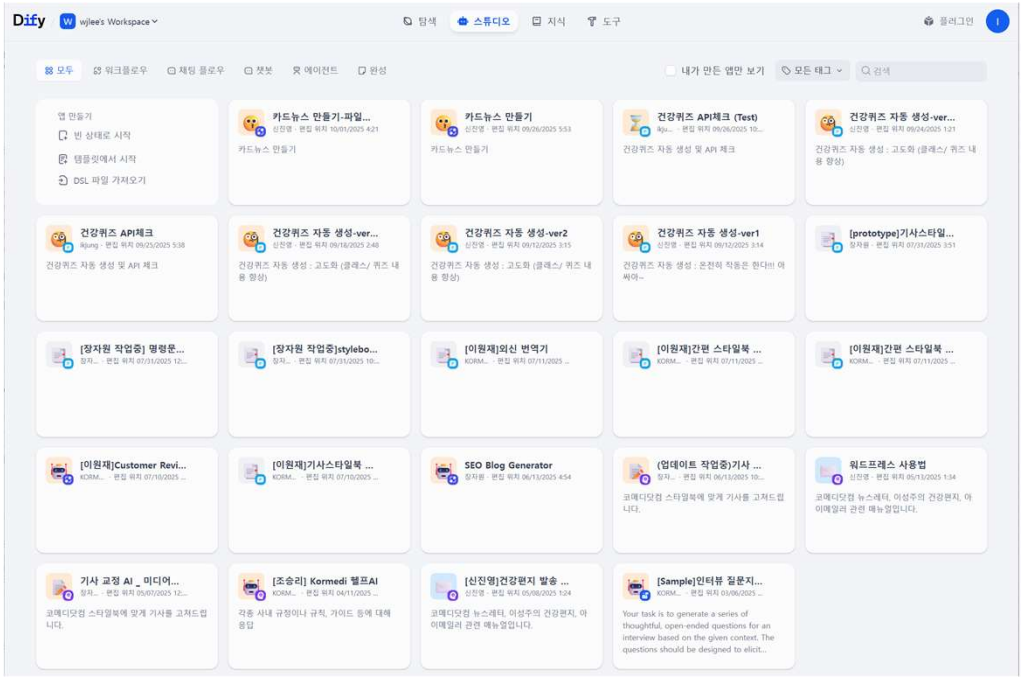
3. 문서를 내부 DB에 저장할 수 있고, DB에 내용을 축적할 수 있으며, 저장된 DB에서 원하는 정보를 검색하여 답변할 수 있다.

4. 답변에 대한 출처를 역으로 저장되어 있는 DB에서 검색 후 검증하는 방식으로 할루시네이션 현상을 줄일 수 있다.

RAG (Retrieval-Augmented Generation) - 검색, 증강, 생성



현재 사내 AI 에이전트인 NO-CODE 기반의 **Dify**가 있음 →



▶ 여기서는 LangChain에 대해 알아볼 예정

LangChain

■ 개요

▶ 언어 모델을 활용해 다양한 애플리케이션을 개발할 수 있는 프레임워크

▶ LangChain을 통해 언어 모델은 다음과 같은 기능을 수행할 수 있게 됩니다.

1) 문맥을 인식하는 기능
- 다양한 문맥 소스와 연결함.
- 프롬프트 지시사항, 소수의 예시, 응답에 근거한 내용 등이 포함됨.
- 이를 통해 제공된 정보를 기반으로 더 정확하고 관련성 높은 답변을 생성 가능.

2) 추론하는 기능
- 주어진 문맥을 바탕으로 어떠한 답변을 제공하거나, 어떤 조치를 취해야 할지를 스스로 추론할 수 있음.
- 단순히 정보를 재생산하는 것을 넘어서, 주어진 상황을 분석하고 적절한 해결책을 제시할 수 있음을 의미.

▶ LangChain 을 활용하면 이전에 언급한 기능을 바탕으로 **검색 증강 생성(RAG) 어플리케이션 제작**, **구조화된 데이터 분석**, **챗봇** 등을 만들 수 있습니다.

LangChain 설치 환경

■ 로컬 리소스 (가정용 셋팅)

▶ Chatreey AN3 (미니PC)
- CPU : AMD Ryzen 7 7840
- RAM : 2xDDR5 So-dimm 4800MHz
- Storage : M.2 NVMe 2280 Slot
- USB4 포트
- WOL (Wake On LAN)
- **LangChain 환경 설정**

▶ AOSTAR AG02
- **eGPU Dock**
- 최대 64Gbps의 대역폭 제공
- USB4 포트

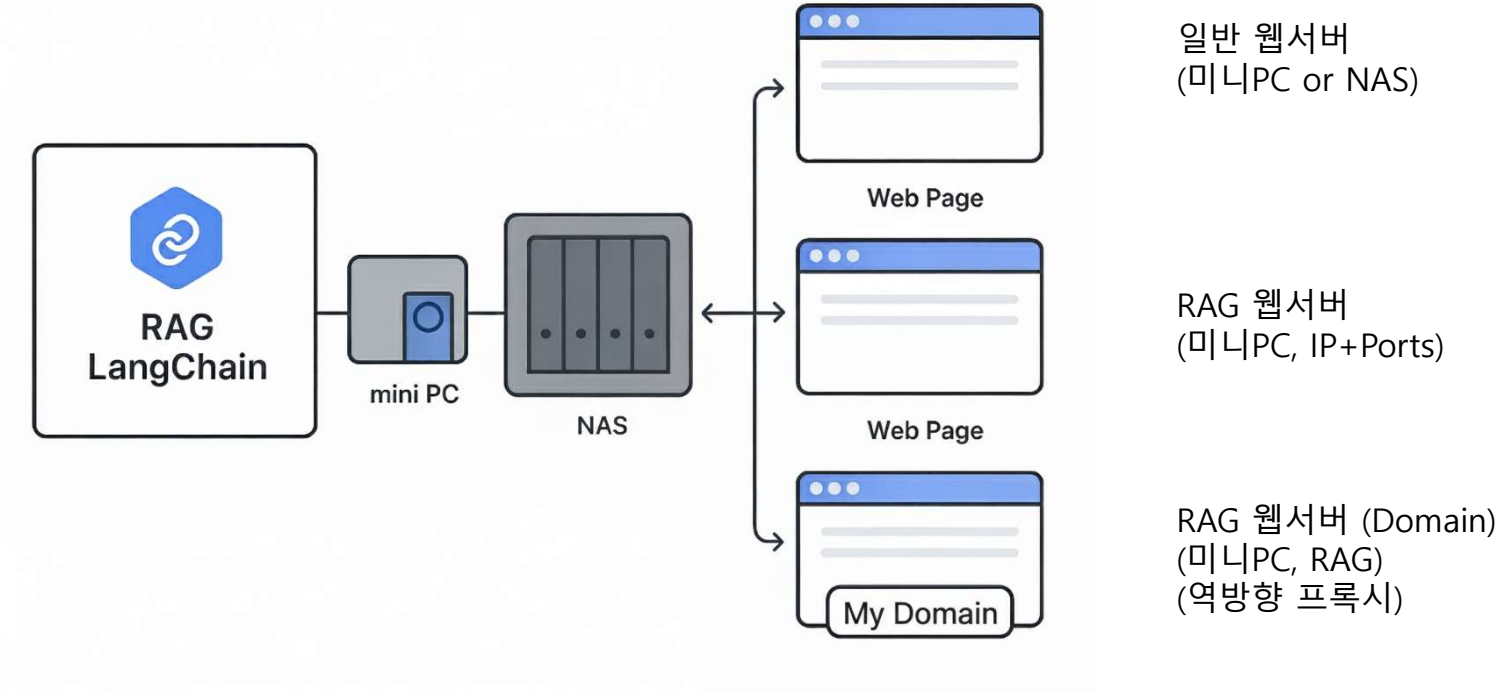
▶ GTX1080
- VRAM : 8GB GDDR5X
- **GPU 연산 작업**

▶ Synology NAS DS120j
- CPU : Marvell Armada 3700 88F3720 듀얼코어 800MHz (64비트)
- RAM : 512MB DDR3L
- 도메인 연결 및 역방향 프록시 설정 등
- **간단한 웹서비스 제공** (PHP, NodeJS, DB 등)

LangChain 설치 환경

▶ 구축 예시

※ 첨부된 이미지들은 전부 로컬 AI로 생성되어, 일부 오타가 있을 수 있음.



일반 웹서버
(미니PC or NAS)

RAG 웹서버
(미니PC, IP+Ports)

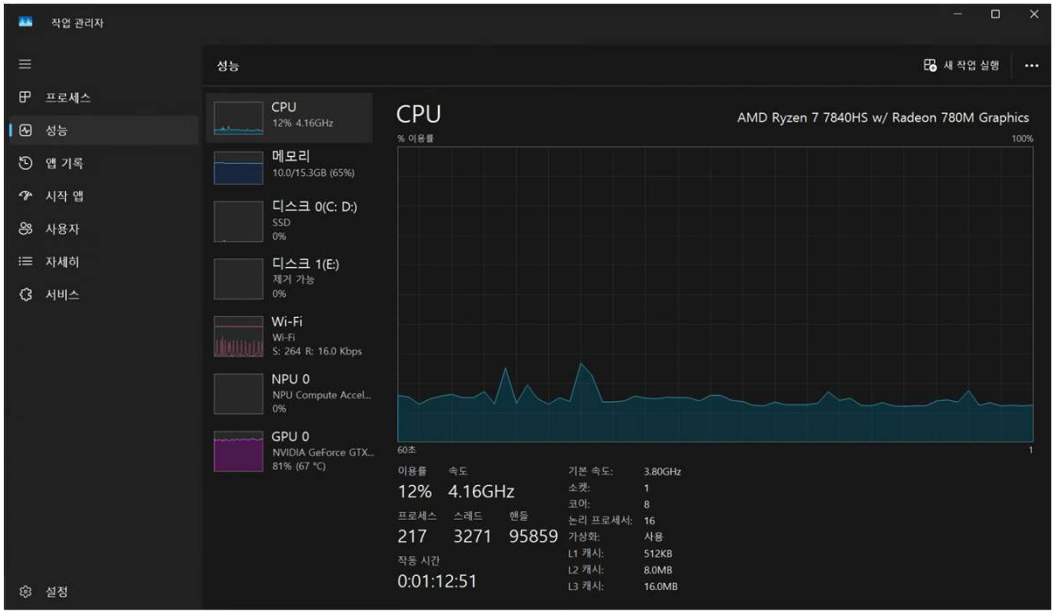
RAG 웹서버 (Domain)
(미니PC, RAG)
(역방향 프록시)

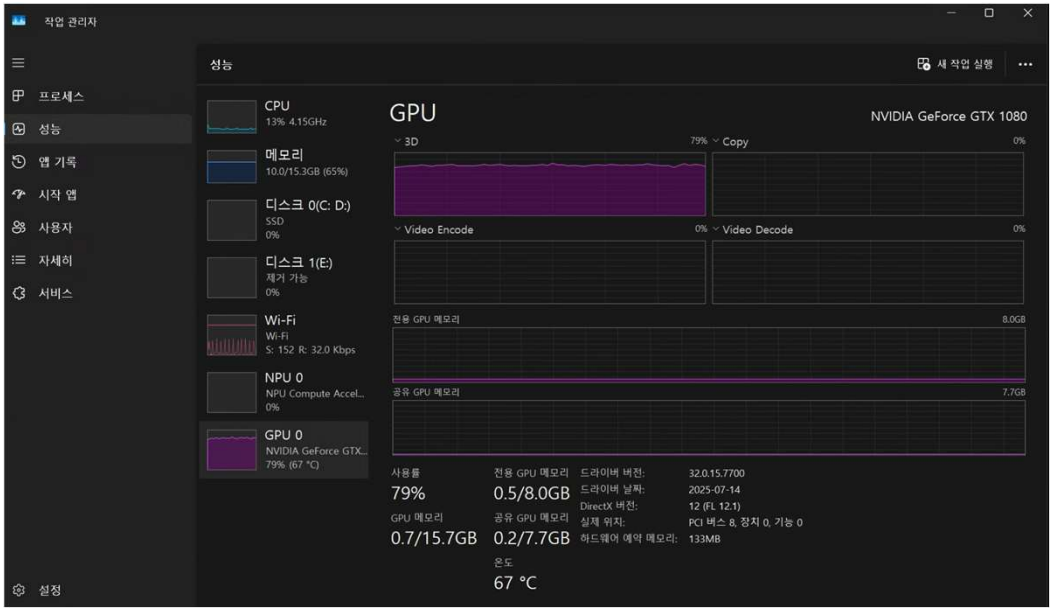
LangChain 동작 환경 (CPU, GPU)

▶ nvidia-smi
현재 langchain 환경에서 gpu를 사용할 수 있는지를 체크.
(실행 결과가 없거나 표시되지 않으면 cpu로만 수행하게 됨)

▶ CPU/GPU 수행 차이
임베딩 / 벡터DB / LLM 추론 단계에서
GPU가 CPU보다 수십~수백배 가량 빠르게 처리함.

```
C:\Users\fearat>nvidia-smi
Wed Sep 17 13:03:46 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 577.00                 Driver Version: 577.00         CUDA Version: 12.9     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce GTX 1080      WDDM  |   00000000:08:00.0 Off |                  N/A |
|  0%   49C    P0             37W /  200W |     153MiB /   8192MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                      GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|    0   N/A  N/A            5152    C+G   ...yb3d8bbwe\WindowsTerminal.exe      N/A      |
|    0   N/A  N/A            8708    C+G   ...indows\System32\ShellHost.exe      N/A      |
|    0   N/A  N/A            8732    C+G   C:\Windows\explorer.exe               N/A      |
|    0   N/A  N/A            9644    C+G   ...cw5n1h2txyewy\SearchHost.exe       N/A      |
|    0   N/A  N/A            9680    C+G   ...y\StartMenuExperienceHost.exe      N/A      |
|    0   N/A  N/A           11368    C+G   ...App_cw5n1h2txyewy\LockApp.exe      N/A      |
|    0   N/A  N/A           12420    C+G   ...8bbwe\PhoneExperienceHost.exe      N/A      |
|    0   N/A  N/A           12832    C+G   ...s\PowerToys.AdvancedPaste.exe      N/A      |
|    0   N/A  N/A           12956    C+G   ...5n1h2txyewy\TextInputHost.exe      N/A      |
|    0   N/A  N/A           13380    C+G   ...UI3Apps\PowerToys.Peek.UI.exe      N/A      |
|    0   N/A  N/A           17108    C+G   ...ntrolPanel\SystemSettings.exe      N/A      |
+-----------------------------------------------------------------------------------------+
```





LangChain 설치 과정 (step 1/2)

■ 미니PC 내 설정 (Win11)

▶ D드라이브 (예시 경로 D:₩langchain_rag₩)를 기준으로 함.

▼ 명령 프롬프트에서 실행

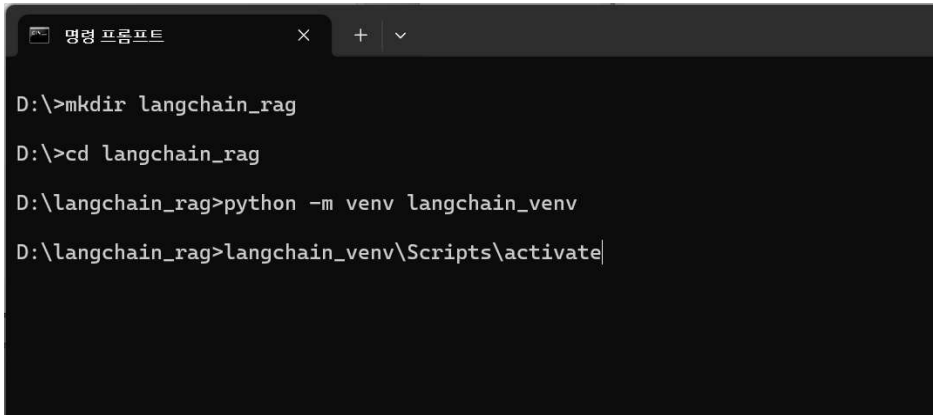
> d:

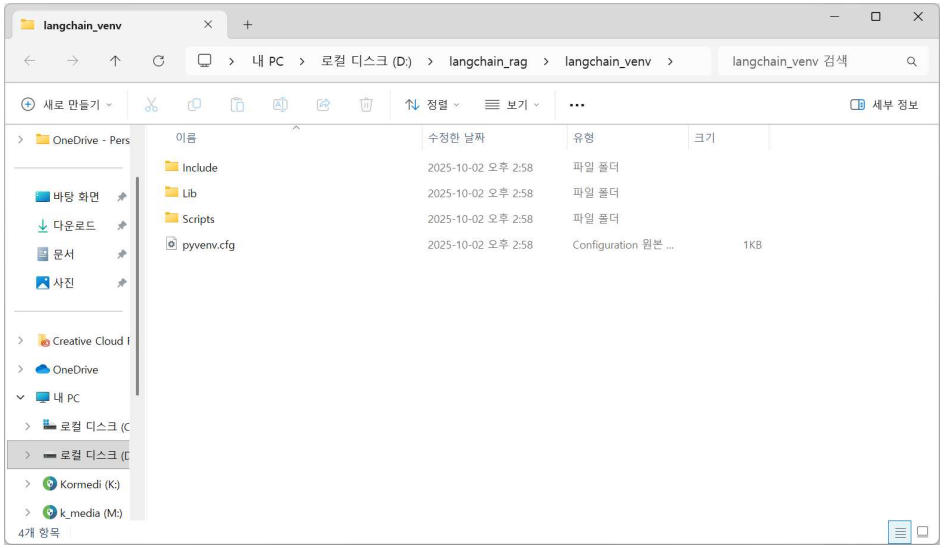
> mkdir langchain_rag

> cd langchain_rag

> python -m venv **langchain_venv**

> .₩**langchain_venv**₩Scripts₩activate





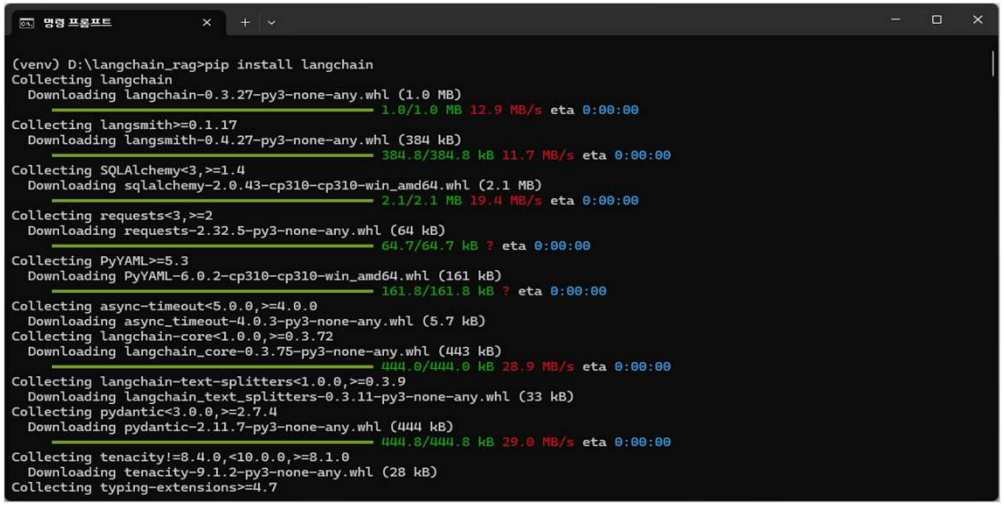
LangChain 설치 과정 (step 2/2)

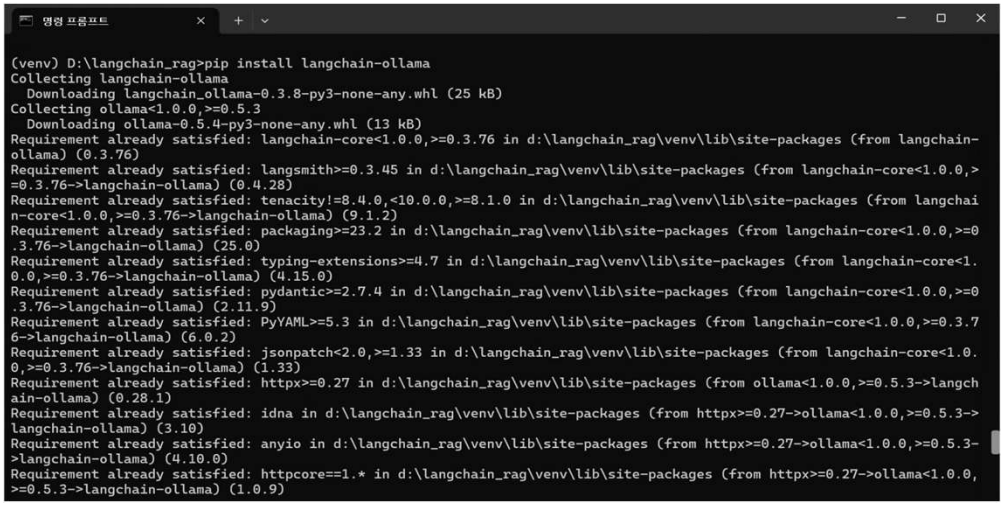
■ langchain 패키지 설치

▼ 명령 프롬프트의 venv 환경에서 실행

> python.exe -m pip install --upgrade pip
> pip install python-dotenv pypdf html2text pymysql
> pip install Flask fastapi unicorn
> pip install langchain langchain-community langchain-ollama
> pip install "langchain[cli]"
> pip install langchain-huggingface sentence-transformers
> pip install -U langchain-ollama
> pip install qdrant-client
> pip install torch qdrant-client
> pip install -U langchain-qdrant

(필요 시 추가 설치 진행)





▼ requirements.txt 같은 형태로 일괄 설치도 가능함. (예시)
> pip install -r https://gpt.fearat.kr/assets/langchain/**requirements.txt**
> pip install -r https://gpt.fearat.kr/assets/langchain/**requirements-mini.txt** (경량화 버전)

LLM 설치 과정 (ollama)

■ ollama 설치

▶ ollama 어플리케이션 다운로드

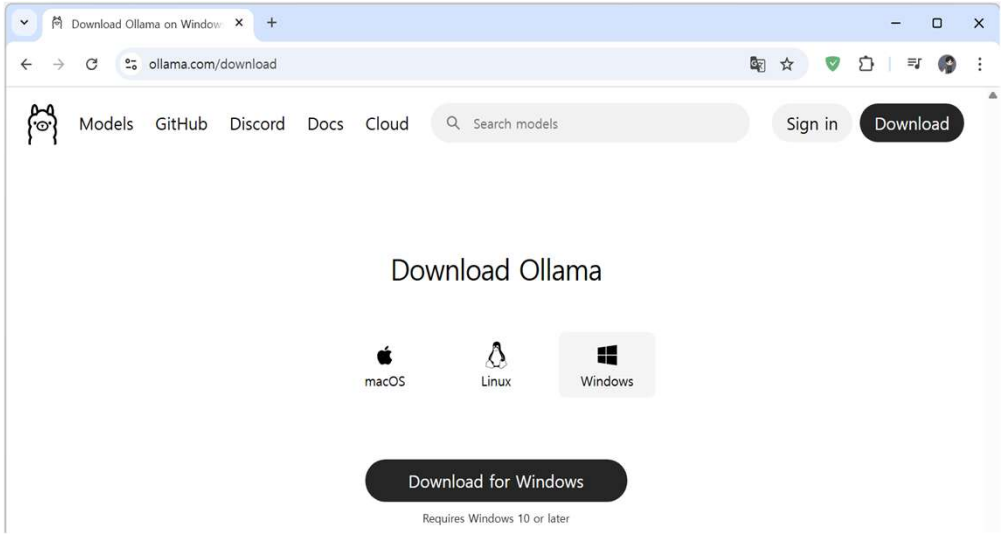
https://www.ollama.com/download
(여기서는 windows 버전으로 다운로드)

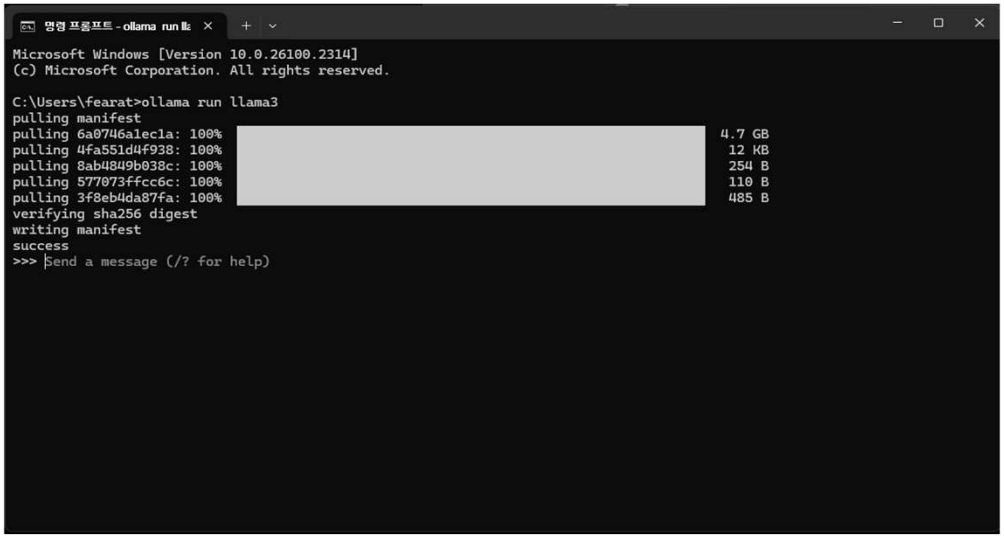
▶ ollama 어플리케이션 설치

▶ ollama에서 사용할 LLM(llama3) 다운로드 및 실행

> ollama run llama3
(model이 없으면 다운로드를 진행함.)

※ 시스템 환경변수 설정을 통해 C가 아닌 D드라이브에 설치도 가능.
(이후 예제에서는 D드라이브를 기준)

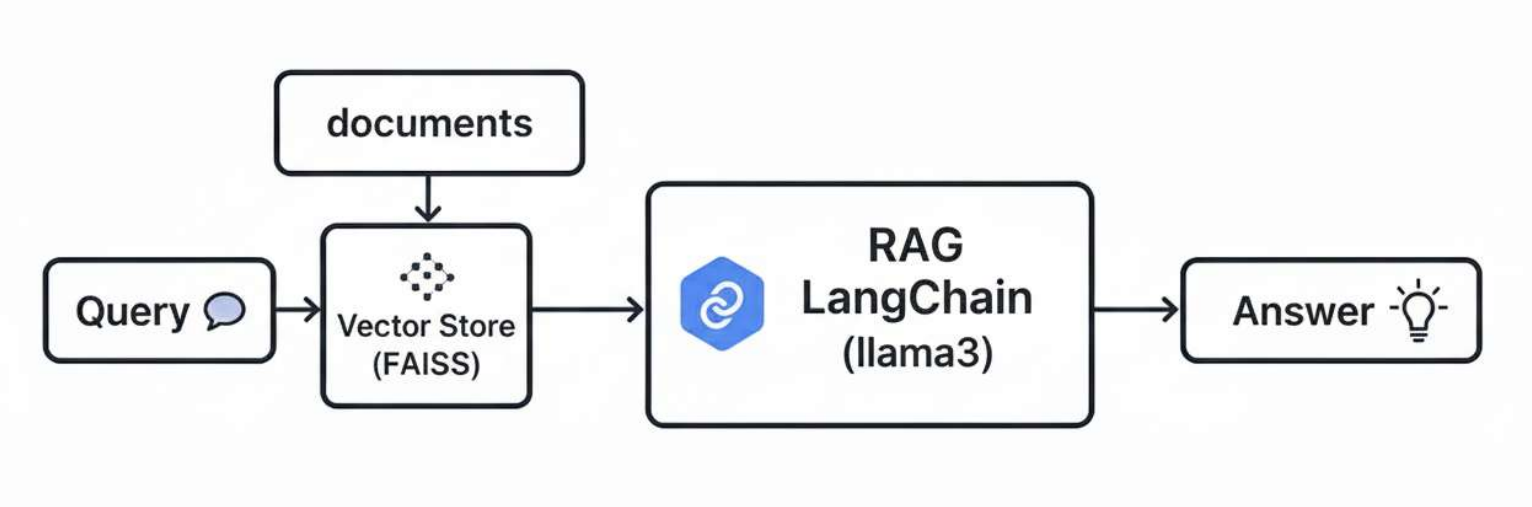




LangChain 예제1 (1/2)

■ 내부 데이터를 기준으로 답변하는 예시

▶ 단일 스크립트 내에서 질문 → 가상의 문서 데이터 → 벡터 스토어 생성 → RAG 체인수행 → 답변 하는 과정



LangChain 예제1 (2/2)

■ 내부 데이터를 기준으로 답변하는 예시

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python rag_test.py

※ 임베딩 시 사용하는 GPU, LLM에 따라 벡터의 품질이 결정되고, 이는 최종 답변의 정확도 및 속도에 영향을 줌.
(GTX1080 기준 15초 가량 소요)

```python
# --- RAG 로직 설정 ---

# Ollama 모델 및 임베딩 설정
embeddings = OllamaEmbeddings(model="nomic-embed-text", base_url="http://127.0.0.1:11434")
llm = OllamaLLM(model="llama3", base_url="http://127.0.0.1:11434")

# 프롬프트 템플릿 설정
prompt_template = """
당신은 게임 공지에 대해 답변하는 친절한 AI 어시스턴트입니다.
제공된 정보를 바탕으로 질문에 **반드시 한국어로** 답변하세요.
만약 정보에 질문에 대한 내용이 없다면, "관련 정보를 찾을 수 없습니다."라고 답변하세요.

{context}

질문: {question}
답변:
"""
PROMPT = PromptTemplate(
    template=prompt_template, input_variables=["context", "question"]
)

# 문서 데이터 및 벡터 스토어 생성
documents = [
    "리그 오브 레전드 시즌 15 시작! 새로운 랭크 시스템과 아이템 업데이트가 적용됩니다.",
    "로스트아크 '카제로스 레이드'가 10월 27일 업데이트됩니다. 신규 유물 장비와 보상이 추가됩니다.",
    "피파 온라인 4, 11월 1일 점검 안내. 서버 안정화 및 신규 선수 로스터 업데이트가 예정되어 있습니다.",
    "발로란트 7.10 패치 노트: 제트와 바이퍼 스킬 조정. 11월 2일 적용됩니다.",
    "오버워치 2 신규 영웅 '벤처' 공개. 벤처는 드릴을 사용하는 공격형 영웅입니다."
]
vectorstore = FAISS.from_texts(documents, embeddings)
```

```python
# RAG 체인 생성
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=vectorstore.as_retriever(),
    chain_type_kwargs={"prompt": PROMPT}
)

# --- 자동 질문 및 답변 ---
def main():
    questions = [
        "오버워치에 새로 추가된 영웅의 이름과 특징은 무엇인가?",
        "리그 오브 레전드의 새 시즌은 몇인가?",
        "피파 온라인 4의 점검일과 점검 내용은?"
    ]

    # 랜덤으로 하나의 질문 선택
    question = random.choice(questions)

    print(f"🎮 선택된 질문: {question}\n")

    # 답변 생성
    response = qa_chain.invoke({"query": question})
    answer = response['result']

    print(f"🧠 답변: {answer}\n")
```

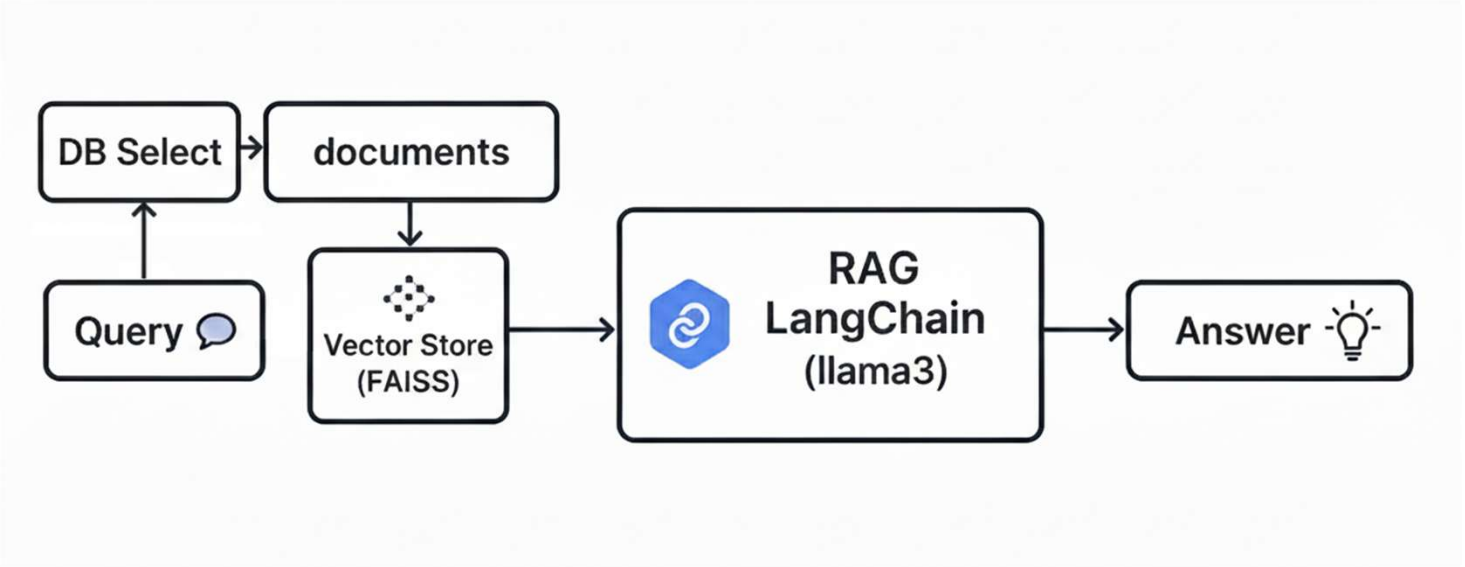
```
(langchain_venv) D:\langchain_rag>python rag_test.py
🎮 선택된 질문: 리그 오브 레전드의 새 시즌은 몇인가?

🧠 답변: 리그 오브 레전드의 새 시즌은 시즌 15입니다! 🎉
```

<실행 예시 : 랜덤 질문 1개에 대해 답변 중>

LangChain 예제2 (1/2)

■ DB에서 조회한 데이터를 기준으로 답변하는 예시

▶ 단일 스크립트 내에서 질문 → DB에서 조회된 문서 데이터 → 벡터 스토어 생성 → RAG 체인수행 → 답변 하는 과정



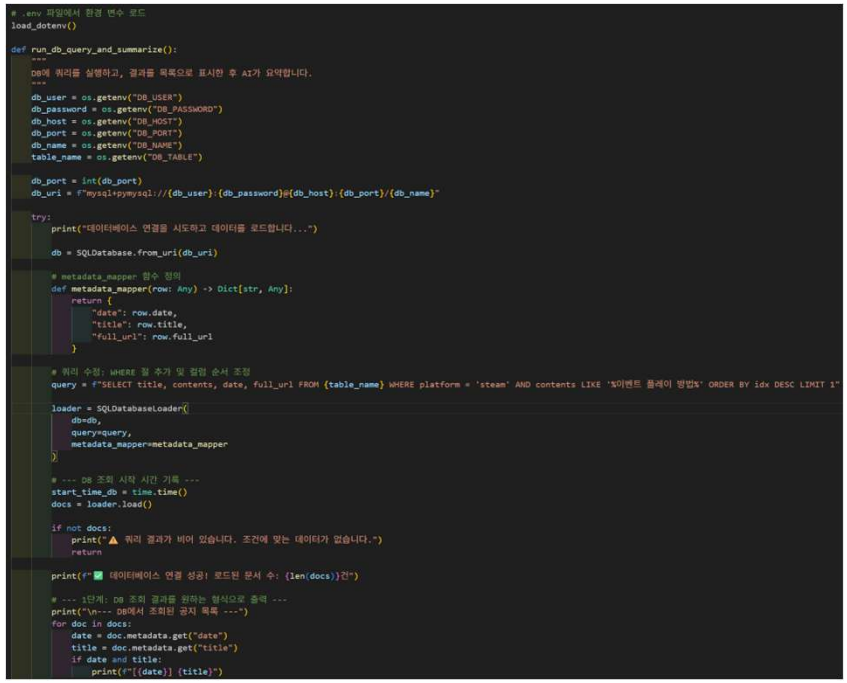
LangChain 예제2 (2/2)

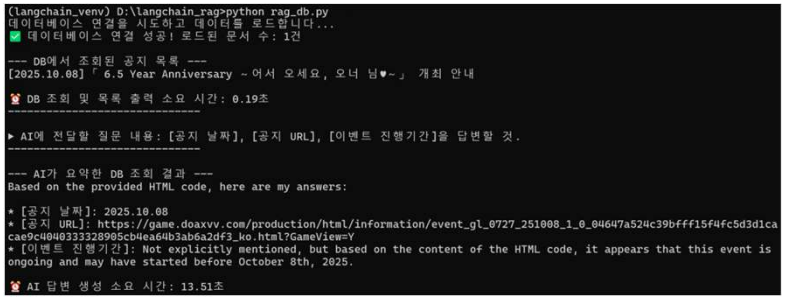
■ DB에서 조회한 데이터를 기준으로 답변하는 예시

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python rag_db.py

※ 임베딩 시 사용하는 LLM에 따라 다중 row를 제대로 답변하지 못하기도 함.



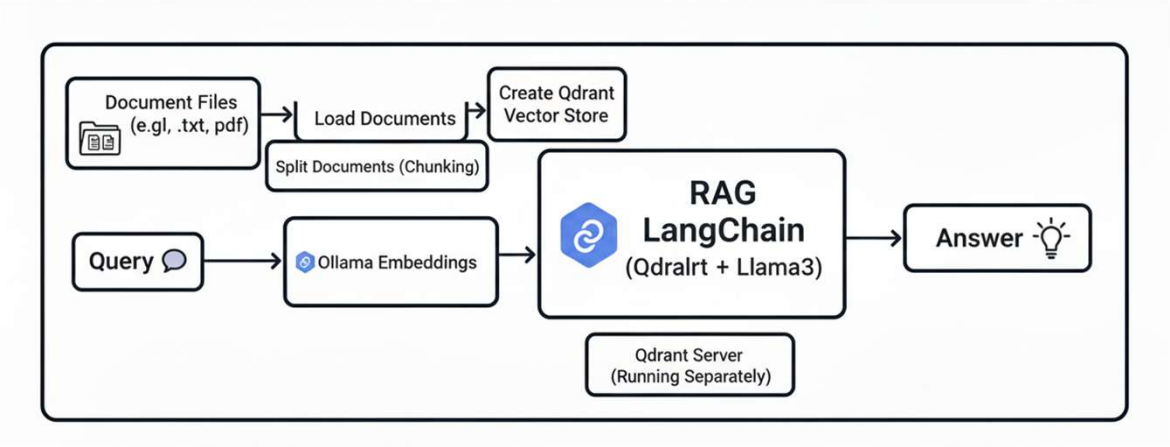


<실행 예시 : DB에서 조회된 내용으로 답변 중>

LangChain 예제3 (1/2)

■ 외부 파일을 조회하여 답변하기

▶ 단일 스크립트 내에서 질문 → 특정 경로의 파일 조회 → RAG 체인수행 → 답변 하는 과정



LangChain 예제3 (2/2)

■ 외부 파일을 조회하여 답변하기

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python rag_by_files.py

```python
def load_recent_html_files(base_path, limit=5):
    """지정된 폴더 내에서 가장 최근 생성된 HTML 파일 n개 로드"""
    html_files = []
    for root, _, files in os.walk(base_path):
        for f in files:
            if f.lower().endswith(".html"):
                full_path = os.path.join(root, f)
                created = os.path.getctime(full_path)
                html_files.append((created, full_path))

    # 최신 limit개 파일 선택
    latest_files = heapq.nlargest(limit, html_files, key=lambda x: x[0])
    return [path for _, path in latest_files]

def extract_text_from_html(file_path):
    """HTML 파일에서 본문 텍스트 추출"""
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            soup = BeautifulSoup(f, "html.parser")
        text = soup.get_text(separator="\n", strip=True)
        return text
    except Exception as e:
        print(f"⚠️ {file_path} 파싱 실패: {e}")
        return ""
```

```python
def run_html_summary():
    load_dotenv()  # .env 환경 변수 로드

    base_path = r"N:\eclipse_workspace\doaxvv\assets\notice\dmm"
    print(f"📂 HTML 파일 경로: {base_path}")

    start_time = time.time()
    limit = 5
    html_files = load_recent_html_files(base_path, limit)

    if not html_files:
        print("❌ HTML 파일을 찾을 수 없습니다.")
        return

    print(f"✅ 최신 HTML 파일 {limit}개를 불러왔습니다:")
    for f in html_files:
        print(" -", os.path.basename(f))

    # HTML 텍스트 로드 및 Document 생성
    start_load = time.time()
    docs = []
    for path in html_files:
        text = extract_text_from_html(path)
        if text:
            docs.append(Document(page_content=text, metadata={"source": os.path.basename(path)}))
    load_time = time.time() - start_load

    print(f"\n📄 문서 로드 완료. 총 {len(docs)}개 파일에서 텍스트 추출됨.")
    print(f"⏱ HTML 파일 로드 소요 시간: {load_time:.2f}초")
```

```python
# 텍스트 분할 (길이 제한 방지)
splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
split_docs = splitter.split_documents(docs)

# --- FAISS (메모리 내 벡터스토어) ---
vectorstore = FAISS.from_documents(split_docs, embeddings)

# --- RAG 체인 ---
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=vectorstore.as_retriever(search_kwargs={"k": 5}),
    chain_type="stuff",
    chain_type_kwargs={"prompt": PROMPT},
)

query_to_ai = f"총 {len(html_files)}개의 HTML 공지 내용을 각각 1~2문장으로 요약해줘."
print(f"\n▶ AI에 전달할 질문 내용: {query_to_ai}")

print("\n🧠 AI 요약 생성 중...")
start_ai = time.time()
response = qa_chain.invoke({"query": query_to_ai})
elapsed_ai = time.time() - start_ai

print("\n--- 📝 요약 결과 ---")
print(response["result"])
print(f"\n⏰ AI 요약 소요 시간: {elapsed_ai:.2f}초")

total_time = time.time() - start_time
print(f"✅ 전체 실행 완료 (총 {total_time:.2f}초)")
```

```
(langchain_venv) D:\langchain_rag>python rag_by_files.py
📂 HTML 파일 경로: N:\eclipse_workspace\doaxvv\assets\notice\dmm
✅ 최신 HTML 파일 5개를 불러왔습니다:
 - 49962.html
 - 49963.html
 - 49965.html
 - 49966.html
 - 49968.html

📄 문서 로드 완료. 총 5개 파일에서 텍스트 추출됨.
⏱HTML 파일 로드 소요 시간: 0.32초

▶ AI에 전달할 질문 내용: 총 5개의 HTML 공지 내용을 각각 1~2문장으로 요약해줘.

🧠 AI 요약 생성 중...

--- 📝 요약 결과 ---
Contents:
1. "DEAD OR ALIVE Xtreme Venus Vacation" official website notice
Summary: The official website of "DEAD OR ALIVE Xtreme Venus Vacation" has been updated, and the event period has been a
nnounced.

2. SSR swimsuit can be obtained by using the ticket gacha
Summary: SSR swimsuits are available through the ticket gacha, and there is a chance to obtain them during the event per
iod.

3. Event overview for the login bonus campaign
Summary: The "なつかし10連ｶ ﾁｹｯﾄ(月影)" login bonus campaign will be held from October 7th to October 14th, and it wil
l award up to 70 consecutive draws of SSR swimsuits.

4. Four "なつかしコーデガチャ" gachas share a common step system
Summary: The four "なつかしコーデガチャ" gachas have a shared step system, where the steps can be increased by using the
 ticket gacha and completing the gatcha draws.

5. The "なつかし10連ｶ ﾁｹｯﾄ(月影)" login bonus campaign will end on October 14th
Summary: The "なつかし10連ｶ ﾁｹｯﾄ(月影)" login bonus campaign will end on October 14th, and the SSR swimsuits obtained
during this period can be used until then.

⏰ AI 요약 소요 시간: 18.25초
✅ 전체 실행 완료 (총 21.98초)
```

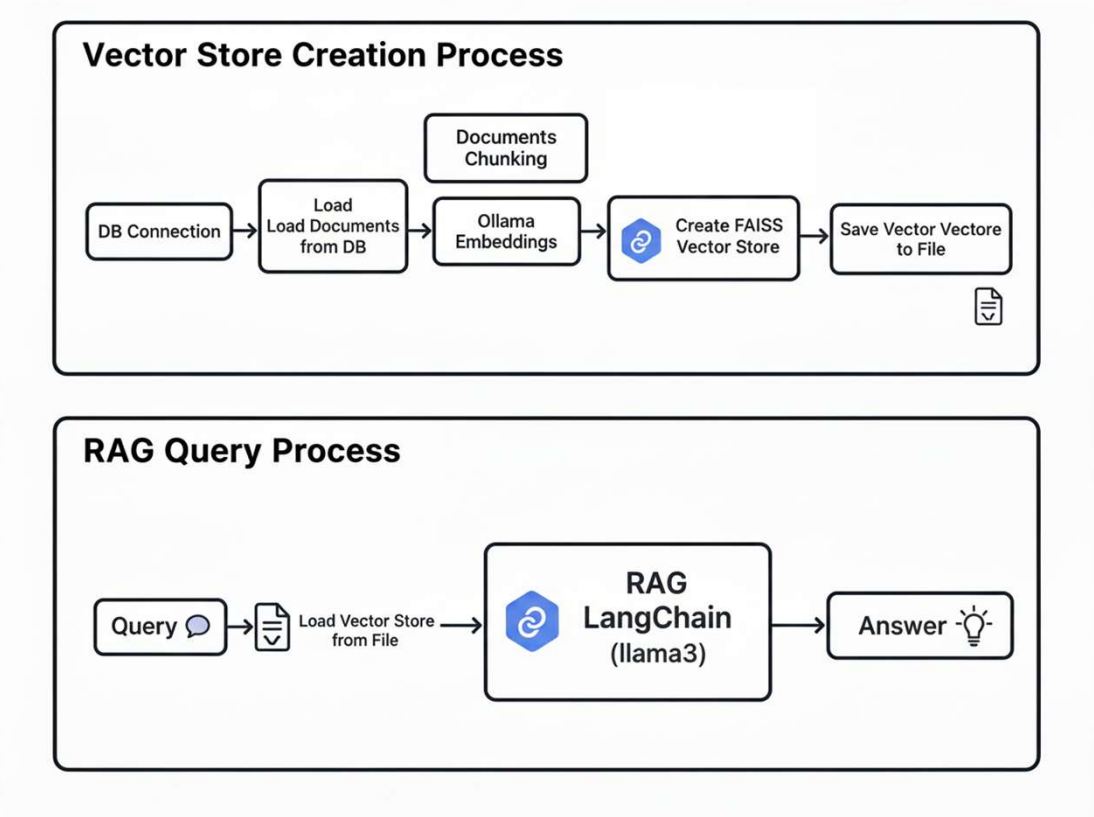
<실행 예시 : 특정 경로의 파일로부터 질문에 대해 답변>

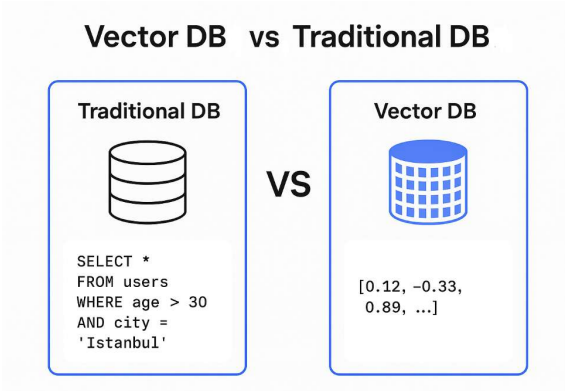
LangChain 예제4 (1/3)

■ 벡터DB 생성 후, 벡터DB에서 조회하여 답변하기

▶ documents의 내용을 DB에서 조회하여, Embedding 과정 후 Vector Store로 파일 저장

▶ 단일 스크립트 내에서 질문 → 저장된 Vector Store 파일 → RAG 체인수행 → 답변 하는 과정





<Vector DB와 기존 DB 구문의 비교>

Firecrawl

<Vector DB에 활용하기 좋은 예시 : Firecrawl>

LangChain 예제4 (2/3)

■ 벡터DB 생성 후, 벡터DB에서 조회하여 답변하기

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python create_vector_store.py
(venv) > python rag_faiss.py

※ 벡터DB 생성 시, 5000 rows 기준 속도
- gpu : 4시간 가량 소요
- cpu : 1개월 이상 소요.

```python
def create_and_save_vector_store():
    print("1. 데이터베이스 연결을 시도 중...")
    try:
        db_user = os.getenv("DB_USER")
        db_password = os.getenv("DB_PASSWORD")
        db_host = os.getenv("DB_HOST")
        db_port = int(os.getenv("DB_PORT"))
        db_name = os.getenv("DB_NAME")
        table_name = os.getenv("DB_TABLE")
        db_uri = f"mysql+pymysql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}"

        db = SQLDatabase.from_uri(db_uri)
        print("2. SQLDatabase 객체 생성 완료.")

        def metadata_mapper(row: Any) -> Dict[str, Any]:
            return {"date": row.date, "title": row.title, "contents": row.contents, "full_url": row.full_url, "platform": row.platform}

        query = f"SELECT title, contents, date, full_url, platform FROM {table_name}"
        loader = SQLDatabaseLoader(db=db, query=query, metadata_mapper=metadata_mapper)

        print("3. DB에서 문서 로딩 시작...")
        docs = loader.load()

        if not docs:
            print("⚠️ 로드할 문서가 없습니다. 벡터 스토어를 생성하지 않고 종료합니다.")
            return

        print(f"✅ DB 문서 로드 성공. 총 {len(docs)}건.")

        print("4. 문서 chunking 시작...")
        #splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
        splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=100)
        split_docs = splitter.split_documents(docs)
        print(f"✅ 문서 chunking 완료. 총 {len(split_docs)}개 청크 생성.")

        print("5. Ollama Embeddings 모델에 연결 중...")
        # model="nomic-embed-text"이 너무 부정확해서 llama3로 변경
        embeddings = OllamaEmbeddings(
            model="llama3",
            base_url="http://127.0.0.1:11434"
        )

        print("6. FAISS 벡터 스토어 생성 중...")
        vectorstore = FAISS.from_documents(split_docs, embeddings)

        os.makedirs("faiss_index", exist_ok=True)
        print("7. 생성된 벡터 스토어를 'faiss_index' 디렉터리에 저장 중...")
        vectorstore.save_local("faiss_index")

        print("🎉 FAISS 벡터 스토어를 'faiss_index' 디렉터리에 성공적으로 저장했습니다.")

    except SQLAlchemyError as e:
        print(f"❌ 데이터베이스 연결 또는 쿼리 실패: {e}")
    except Exception as e:
        print(f"❌ 예기치 않은 오류가 발생했습니다: {e}")
```

LangChain 예제4 (3/3)

■ 벡터DB 생성 후, 벡터DB에서 조회하여 답변하기

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python create_vector_store.py
(venv) > python rag_faiss.py

※ 답변 정확도는 DB 때와 마찬가지로 떨어지는 편

```python
def run_faiss_query_and_summarize():
    """
    faiss_index에서 벡터 데이터를 불러와 AI가 요약합니다.
    """
    try:
        print("FAISS 벡터 스토어를 불러오는 중...")

        # Ollama Embeddings 및 LLM 설정
        embeddings = OllamaEmbeddings(
            model="nomic-embed-text",
            base_url="http://127.0.0.1:11434"
        )
        llm = OllamaLLM(
            model="llama3",
            base_url="http://127.0.0.1:11434"
        )

        # FAISS 인덱스 로드
        start_time_load = time.time()
        vectorstore = FAISS.load_local(
            "faiss_index",
            embeddings,
            allow_dangerous_deserialization=True  # pickle 사용 허용
        )
        end_time_load = time.time()
        print(f"✅ FAISS 벡터 스토어 로드 완료 (소요 시간: {end_time_load - start_time_load:.2f}초)")

        # 프롬프트 템플릿을 파일에서 로드
        try:
            with open("prompts/doaxvv_notice.txt", "r", encoding="utf-8") as f:
                prompt_template = f.read()
        except FileNotFoundError:
            print("❌ 'prompts/doaxvv_notice.txt' 파일을 찾을 수 없습니다. 경로를 확인하세요.")
            return
```

```python
        PROMPT = PromptTemplate(
            template=prompt_template,
            input_variables=["context", "question"]
        )

        # RAG 체인 설정
        qa_chain = RetrievalQA.from_chain_type(
            llm=llm,
            chain_type="stuff",
            retriever = vectorstore.as_retriever(search_kwargs={"k": 10}),
            chain_type_kwargs={"prompt": PROMPT}
        )

        # AI 답변 생성
        start_time_ai = time.time()
        query_to_ai = "최근 문서들의 주요 내용을 요약해줘."
        response = qa_chain.invoke({"query": query_to_ai})
        end_time_ai = time.time()

        print("\n--- AI가 요약한 FAISS 벡터 데이터 ---")
        print(response["result"])
        print(f"\n⏰ AI 답변 생성 소요 시간: {end_time_ai - start_time_ai:.2f}초")

    except Exception as e:
        print(f"❌ 예기치 않은 오류가 발생했습니다: {e}")
```

```
(langchain_venv) D:\langchain_rag>python rag_faiss.py
FAISS 벡터 스토어를 불러오는 중...
✅ FAISS 벡터 스토어 로드 완료 (소요 시간: 2.05초)

▶ AI에 전달할 질문 내용: 최근 문서들의 주요 내용을 요약해줘.

--- AI가 요약한 FAISS 벡터 데이터 ---
관련 정보를 찾을 수 없습니다.

⏰ AI 답변 생성 소요 시간: 8.16초
```

<실행 예시 : 생성한 벡터DB로 답변 중 (실패)>

LangChain 예제5

■ 성능 개선을 위해 HuggingFace로 벡터DB 생성 후, 답변하기

▼ 예제 3의 create_vector_store.py과 대부분 동일.

※ 일반 윈도우 환경에서는 faiss-gpu를 사용할 수 없어 WSL2를 설치 후 적용
※ faiss-gpu를 제대로 사용하려면 Docker 환경에서 진행해야 함.
※ Docker에서 진행 시 GTX1080은 sm_61를 쓰고 있어 PyTorch 2.x와 호환 불가
(sm_61에 맞게 버전을 강제 다운그레이드해도 불가)

☞ 현재 로컬 리소스(GTX1080)에서는 진행 불가. (GPU 업그레이드 필요...?)

```python
print(f"✅ DB 문서 로드 성공. 총 {len(docs)}건.")

print("4. 문서 chunking 시작...")
splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=100)
split_docs = splitter.split_documents(docs)
print(f"✅ 문서 chunking 완료. 총 {len(split_docs)}개 청크 생성.")

print("5. Ollama Embeddings 모델에 연결 중...")
# model="nomic-embed-text"이 너무 부정확해서 llama3로 변경
embeddings = OllamaEmbeddings(
    model="llama3",
    base_url="http://127.0.0.1:11434"
)

print("6. FAISS 벡터 스토어 생성 중...")
vectorstore = FAISS.from_documents(split_docs, embeddings)

os.makedirs("faiss_index", exist_ok=True)
print("7. 생성된 벡터 스토어를 'faiss_index' 디렉터리에 저장 중...")
vectorstore.save_local("faiss_index")

print("🎉 FAISS 벡터 스토어를 'faiss_index' 디렉터리에 성공적으로 저장했습니다.")
```

<create_vector_store.py (중략)>

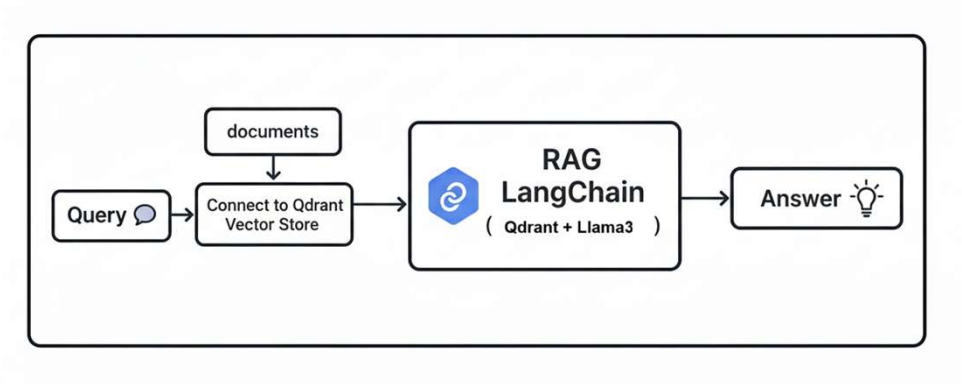
```python
print(f"✅ DB 문서 로드 성공. 총 {len(docs)}건.")

print("4. 문서 chunking 시작...")
splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=100)
split_docs = splitter.split_documents(docs)
print(f"✅ 문서 chunking 완료. 총 {len(split_docs)}개 청크 생성.")

print("5. KURE-v1 임베딩 모델에 연결 중 (GPU 사용)...")
embeddings = HuggingFaceEmbeddings(
    model_name="nlpai-lab/KURE-v1",
    model_kwargs={"device": "cuda"}  # GPU 사용
)

print("6. FAISS 벡터 스토어 생성 중...")
vectorstore = FAISS.from_documents(split_docs, embeddings)

os.makedirs("faiss_index", exist_ok=True)
print("7. 생성된 벡터 스토어를 'faiss_index' 디렉터리에 저장 중...")
vectorstore.save_local("faiss_index")

print("🎉 FAISS 벡터 스토어를 'faiss_index' 디렉터리에 성공적으로 저장했습니다.")
```

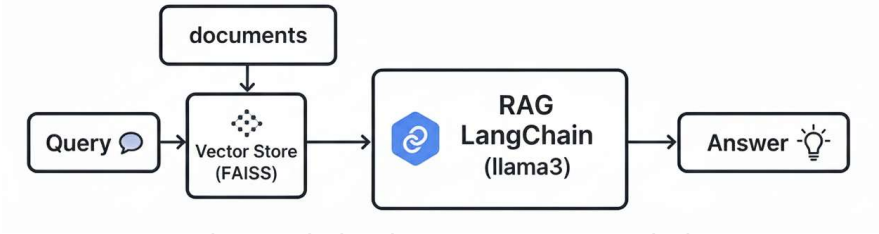
<create_vector_store_hf.py (중략)>

LangChain 예제6 (1/2)

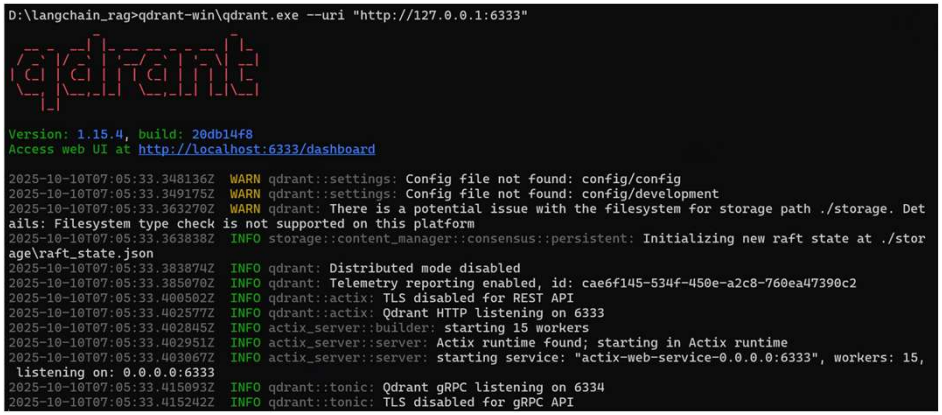
■ Qdrant로 답변얻기

▼ Ollama 외에 Qdrant 서버 추가 실행





<비고 : 예제1의 Ollama+FAISS 방식>



<qdrant 서버 실행 예시>

LangChain 예제6 (2/2)

■ Qdrant로 답변얻기

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python rag_qdrant.py

※ 속도는 Ollama와 비슷하거나 약간 더 빠름.

```
[2025.10.08]「 세 번째 신입 오너님 서포트 패널 미션」 추가 안내
[2025.10.08]「 홈 화면」 즐겨찾기 기능 안내

⏰ DB 조회 및 목록 출력 소요 시간: 0.17초
----------------------------

--- AI가 요약한 DB 조회 결과 ---
Here are the summaries of the main updates for each document:

**Document 1**: The game "DEAD OR ALIVE Xtreme Venus Vacation" will have a new campaign event starting on October 1st, f
eaturing three new items and two new characters.

**Document 2**: A notice to players that the contents and schedules of the campaign event may change at any time without
 prior notice.

⏰ AI 답변 생성 소요 시간: 15.47초

(langchain_venv) D:\langchain_rag>
llama_context: freq_base     = 500000.0
llama_context: freq_scale    = 1
llama_context: n_ctx_per_seq (4096) < n_ctx_train (8192) -- the full capacity of the model will not be utilized
llama_context:  CUDA_Host  output buffer size =     0.50 MiB
llama_kv_cache_unified:      CUDA0 KV buffer size =   512.00 MiB
llama_kv_cache_unified: size =  512.00 MiB (  4096 cells,  32 layers,  1/1 seqs), K (f16):  256.00 MiB, V (f16):  256.00
 MiB
llama_context:      CUDA0 compute buffer size =   300.01 MiB
llama_context:  CUDA_Host compute buffer size =    20.01 MiB
llama_context: graph nodes  = 1126
llama_context: graph splits = 2
time=2025-10-10T16:08:56.036+09:00 level=INFO source=server.go:1289 msg="llama runner started in 4.22 seconds"
time=2025-10-10T16:08:56.037+09:00 level=INFO source=sched.go:470 msg="loaded runners" count=2
time=2025-10-10T16:08:56.037+09:00 level=INFO source=server.go:1251 msg="waiting for llama runner to start responding"
time=2025-10-10T16:08:56.037+09:00 level=INFO source=server.go:1289 msg="llama runner started in 4.22 seconds"
time=2025-10-10T16:08:56.066+09:00 level=WARN source=runner.go:127 msg="truncating input prompt" limit=4096 prompt=23057
 keep=25 new=4096
[GIN] 2025/10/10 - 16:09:06 | 200 |   15.4305049s |       127.0.0.1 | POST     "/api/generate"
```

<ollama + qdrant + rag_qdrant.py 실행 예시>

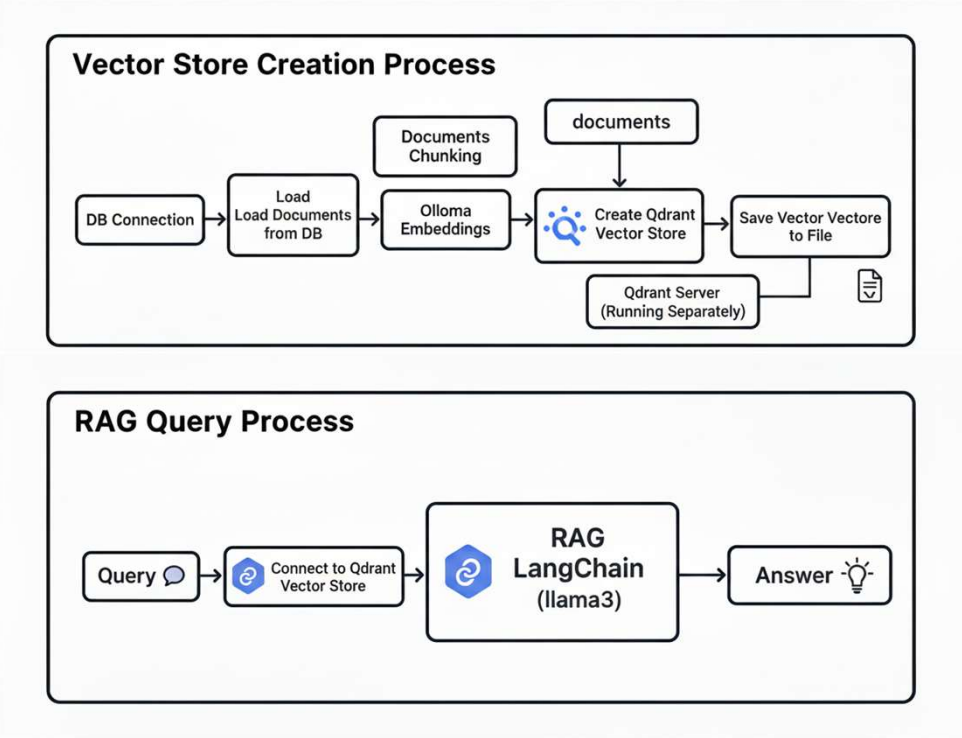
```python
print(f"✅ 데이터베이스 연결 성공! 로드된 문서 수: {len(docs)}건")

elapsed_time_db = time.time() - start_time_db
print(f"\n⏰ DB 조회 및 목록 출력 소요 시간: {elapsed_time_db:.2f}초")
print("-" * 30)

# --- Ollama LLM 및 임베딩 설정 (GPU 사용) ---
embeddings = OllamaEmbeddings(model="nomic-embed-text:latest", base_url="http://127.0.0.1:11434")
llm = OllamaLLM(model="llama3:latest", base_url="http://127.0.0.1:11434")

# 프롬프트 로드
try:
    with open("prompts/doaxvv_notice.llama3.txt", "r", encoding="utf-8") as f:
        prompt_template = f.read()
except FileNotFoundError:
    print("❌ 'prompts/doaxvv_notice.llama3.txt' 파일을 찾을 수 없습니다. 경로를 확인하세요.")
    return

PROMPT = PromptTemplate(
    template=prompt_template, input_variables=["context", "question"]
)

# --- Qdrant 메모리 모드 벡터스토어 ---
vectorstore = Qdrant.from_documents(
    docs,
    embedding=embeddings,
    location=":memory:",
    collection_name="game_updates"
)

# --- RAG 체인 ---
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=vectorstore.as_retriever(),
    chain_type_kwargs={"prompt": PROMPT}
)

start_time_ai = time.time()
query_to_ai = f"총 {len(docs)}건의 문서에 대해, 주요 업데이트 내용을 각각 1~2문장으로 요약해서 알려줘"
response = qa_chain.invoke({"query": query_to_ai})

print("\n--- AI가 요약한 DB 조회 결과 ---")
print(response['result'])

elapsed_time_ai = time.time() - start_time_ai
print(f"\n⏰ AI 답변 생성 소요 시간: {elapsed_time_ai:.2f}초")
```

LangChain 예제7 (1/3)

■ Qdrant로 벡터DB 생성 후, 조회하여 답변하기

▶ documents의 내용을 DB에서 조회하여, Embedding 과정 후 Vector Store로 파일 저장

▶ 단일 스크립트 내에서 질문 → 저장된 Vector Store 파일 → RAG 체인수행 → 답변 하는 과정



LangChain 예제7 (2/3)

■ Qdrant로 벡터DB 생성 후, 조회하여 답변하기

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python create_vector_store_mxbai.py
(venv) > python rag_qdrant_vector.py

※ Vector DB는 다른 경로에 생성됨.

```python
print(f"✅ DB 문서 로드 성공. 총 {len(docs)}건.")

print("4. 문서 chunking 시작...")
splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=100)
split_docs = splitter.split_documents(docs)
print(f"✅ 문서 chunking 완료. 총 {len(split_docs)}개 청크 생성.")

print("5. Ollama Embeddings 모델에 연결 중...")
# model="nomic-embed-text"이 너무 부정확해서 llama3로 변경
embeddings = OllamaEmbeddings(
    model="llama3",
    base_url="http://127.0.0.1:11434"
)

print("6. FAISS 벡터 스토어 생성 중...")
vectorstore = FAISS.from_documents(split_docs, embeddings)

os.makedirs("faiss_index", exist_ok=True)
print("7. 생성된 벡터 스토어를 'faiss_index' 디렉터리에 저장 중...")
vectorstore.save_local("faiss_index")

print("🎉 FAISS 벡터 스토어를 'faiss_index' 디렉터리에 성공적으로 저장했습니다.")
```

<create_vector_store.py (중략)>

```python
print(f"✅ DB 문서 로드 성공. 총 {len(docs)}건.")

print("4. 문서 chunking 시작...")
splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=100)
split_docs = splitter.split_documents(docs)
print(f"✅ 문서 chunking 완료. 총 {len(split_docs)}개 청크 생성.")

print("5. Ollama Embeddings(httpx) 래퍼 생성 중...")
embeddings = HTTPXEmbeddingsWrapper(model="mxbai-embed-large:latest")

print("6. Qdrant 벡터 스토어 생성 중...")
"""vectorstore = Qdrant.from_documents(
    split_docs,
    embedding=embeddings,
    location="vector_db_mxbai",  # 벡터 DB 저장 경로
    collection_name="game_updates"
)"""
vectorstore = Qdrant.from_documents(
    split_docs,
    embedding=embeddings,
    url="http://127.0.0.1:6333",  # location 대신 url 사용
    collection_name="game_updates"
)

print("🎉 Qdrant 벡터 스토어를 'vector_db_mxbai' 디렉터리에 성공적으로 생성했습니다.")
```

<create_vector_store_mxbai.py (중략)>

LangChain 예제7 (3/3)

■ Qdrant로 벡터DB 생성 후, 조회하여 답변하기

▼ 명령 프롬프트의 venv 환경에서 실행

(venv) > python create_vector_store_mxbai.py
(venv) > python rag_qdrant_vector.py

※ 임베딩 시 사용하는 LLM에 따라 다중 row를 제대로 답변하지 못하기도 함.
※ 최신 정보와 오래된 정보를 제대로 구분하지 못하는 경우도 발생.

```
(langchain_venv) D:\langchain_rag>python rag_qdrant_vector.py

▶ AI에 전달할 질문 내용: 가장 최근 공지 1건의 주요 내용을 한글로 요약해서 공지날짜와 같이 답변해줘.

--- AI가 요약한 결과 ---
2022년 8월 18일

이벤트에서는 "첫 커버 걸"이 등장하여, 친구로 등록된 타 오너와 협력하여 인연 페스티벌을 공략하고, 얻은 스코어에 따라 랭킹을 경쟁하게 됩니다. 또한, 주목할 만한 스코어 보상과 랭킹 보상을 받을 수 있습니다.

⏰ 답변 생성 소요 시간: 6.54초
```

<실행 예시 : 프롬프트 규칙에 따라 한글로 답변 중>

```python
# --- Ollama LLM 및 Embeddings 설정 ---
embeddings = OllamaEmbeddings(
    model="mxbai-embed-large:latest",
    base_url="http://127.0.0.1:11434"
)
llm = OllamaLLM(
    model="llama3:latest",
    base_url="http://127.0.0.1:11434"
)

# --- 프롬프트 로드 ---
try:
    with open("prompts/doaxvv_notice.llama3.txt", "r", encoding="utf-8") as f:
        prompt_template = f.read()
except FileNotFoundError:
    print("❌ 'prompts/doaxvv_notice.llama3.txt' 파일을 찾을 수 없습니다. 경로를 확인하세요.")
    return

PROMPT = PromptTemplate(
    template=prompt_template,
    input_variables=["context", "question"]
)

# --- 기존 Qdrant 서버에 연결 ---
qdrant_client = QdrantClient(url="http://127.0.0.1:6333")
vectorstore = QdrantVectorStore(
    client=qdrant_client,
    collection_name="game_updates",
    embedding=embeddings
)

# --- RAG 체인 ---
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=vectorstore.as_retriever(),
    chain_type_kwargs={"prompt": PROMPT}
)

# --- 질문 정의 ---
query_to_ai = "가장 최근 공지 1건의 주요 내용을 한글로 요약해서 공지날짜와 같이 답변해줘."
print(f"\n▶ AI에 전달할 질문 내용: {query_to_ai}")

start_time = time.time()
response = qa_chain.invoke({"query": query_to_ai})
elapsed_time = time.time() - start_time

print("\n--- AI가 요약한 결과 ---")
print(response['result'])
print(f"\n⏰ 답변 생성 소요 시간: {elapsed_time:.2f}초")
```

LangChain 예제8

■ Openai LLM 사용하기

▶ gpt-4o, gpt-3.5-turbo 등의 LLM 사용 가능

▶ 클라우드 기반의 외부 LLM을 사용하므로, GPU의 중요성이 낮아짐

```
# --- 2단계: OpenAI LLM 및 Embeddings 설정 ---
#embeddings = OpenAIEmbeddings(model="text-embedding-3-large")
#llm = ChatOpenAI(model="gpt-4o", temperature=0)
# 비용 절감 모델 사용 예시
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
```

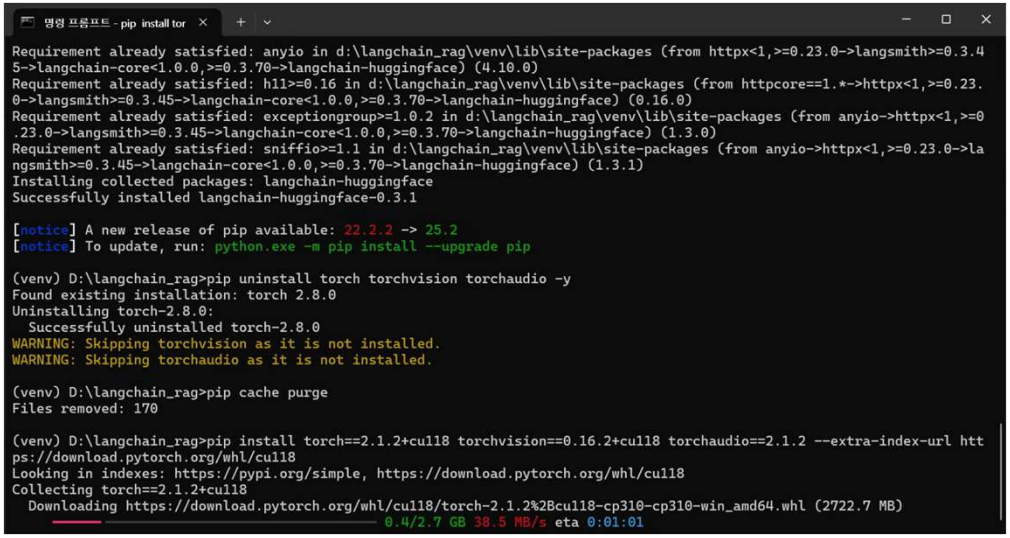
```
(langchain_venv) D:\langchain_rag>python rag_db_openai.py
데이터베이스 연결을 시도하고 데이터를 로드합니다...
✅ DB 연결 성공! 로드된 문서 수: 5건

--- DB에서 조회된 공지 목록 ---
[2025.10.07]「『レスレリアーナのアトリエ』コラボ記念キャンペーン第2弾」開催のお知らせ
[2025.10.07]「トレンドコーデガチャ」「なつかしコーデガチャ」開催のお知らせ
[2025.10.07]「レスレリアーナのアトリエ・デコブロマイドガチャ」開催のお知らせ
[2025.10.07]「月明かりのハロウィンフェス」開催のお知らせ
[2025.10.07]「ビーチフラッグ」の遊び方

⏰ DB 조회 및 목록 출력 소요 시간: 0.19초
------------------------------

▶ AI에 전달할 질문 내용: 가장 최근 공지 1건의 주요 내용을 [공지 날짜], [공지 URL], [이벤트 진행기간] 형태로 답변할 것.
❌ 예기치 않은 오류가 발생했습니다: Error code: 429 - {'error': {'message': 'You exceeded your current quota, please che
ck your plan and billing details. For more information on this error, read the docs: https://platform.openai.com/docs/gu
ides/error-codes/api-errors.', 'type': 'insufficient_quota', 'param': None, 'code': 'insufficient_quota'}}
```

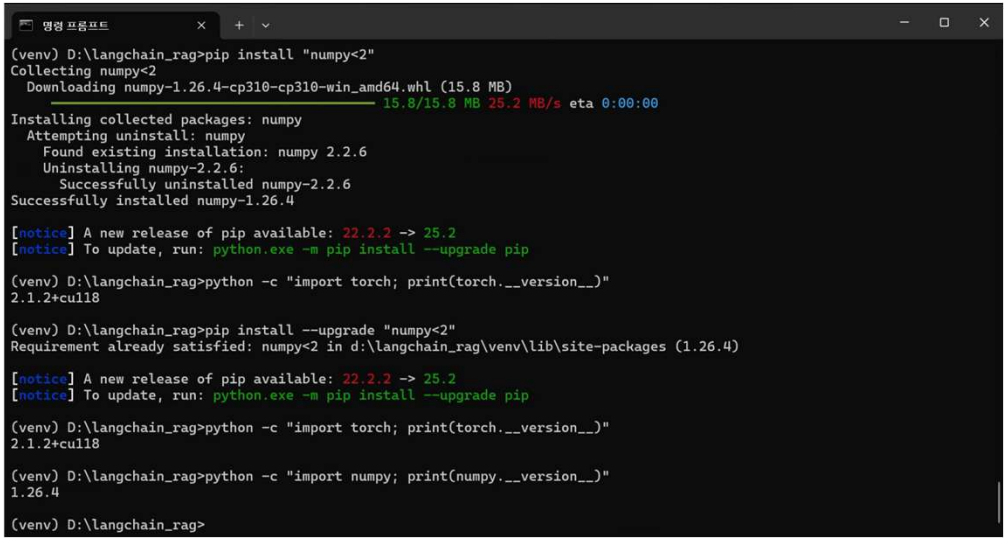
<실행 예시 : openai의 LLM 연결 (429 error)>

LangChain 후기

■ 로컬 환경
▶ GTX 1080 (sm_61) 으로 실행하기에는 약간 무리. (GPU 문제는 WSL2, Docker로도 해결 안됨)
▶ Vector Store 및 LLM에 따라 답변 퀄리티가 크게 변함.
▶ Win11 환경에서의 설치 및 의존성 문제를 해결해야 함. (PyTorch, CUDA 버전 등)

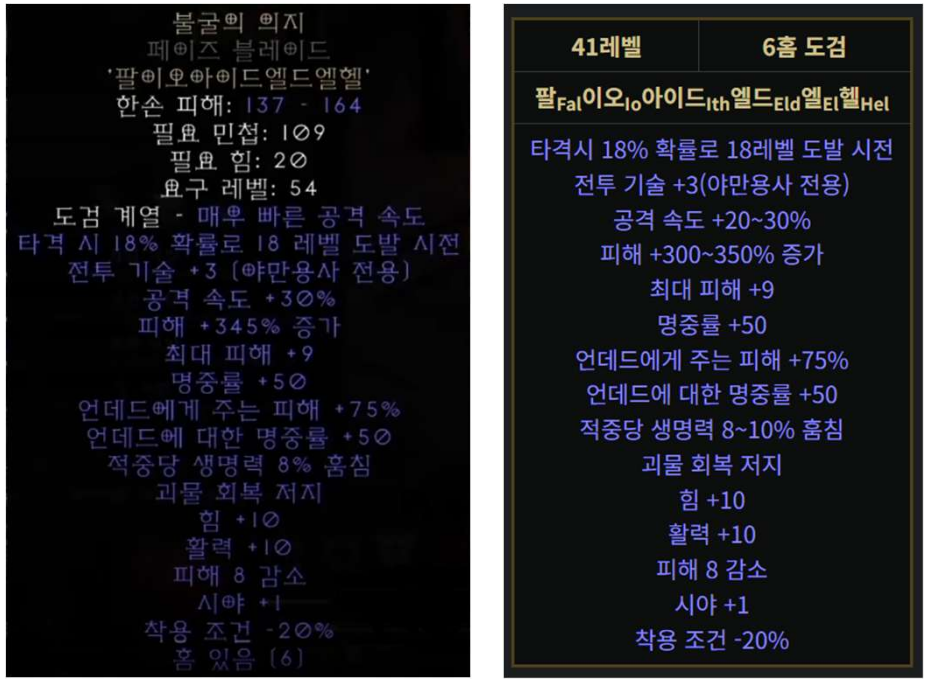
■ 클라우드 환경
▶ 답변 정확도는 높음. 대신 비용이 발생.
▶ 토큰 수의 영향을 많이 받음





< faiss-gpu 사용을 위해 torch 버전을 낮추려는 시도. (결국 실패) >

마무리



<D2R Runeword 2.4>
불굴의 의지

Unbending Will

- 캐릭터 직업/활용 여부에 따라서는 졸업템으로 사용할 수 있음
- 무기 베이스(재료)가 되는 무기가 중요
- 활용이 자유로움 (직접 착용 or 용병이 착용)
- Fal Io Ith Eld El Hel을 순서대로 정확하게 조합해야 제작됨.
- 제작 시 변동 옵션이 커서, 원하는 옵션을 얻기 이해 여러번 제작해야 함.
- 아이템을 얻기 위한 의지 (시간, 반복 투자)

LangChain

- 잘 만들어두면 코어 프레임워크로서 사용할 수 있음
- 하드웨어(CPU, GPU)가 매우 중요
- 활용이 자유로움 (NoCode 방식보다 더 정교하고 맞춤화된 솔루션)
- Loader, Splitter, Embeddings, Retriever, LLM 등이 정확하게 연결되어야 동작.
- 응답 품질을 위해 Chunk Size, LLM, 프롬프트 등을 계속 테스트해야 함.
- 지식과 기술을 얻으려는 의지 (코딩 능력, 구매 비용 등)

LangChain 참고 (1/2)

테디노트의 RAG 비법노트 : GPT·로컬 모델부터 LangGraph·Agent까지

평생소장 | 약 66시간 | 누구나

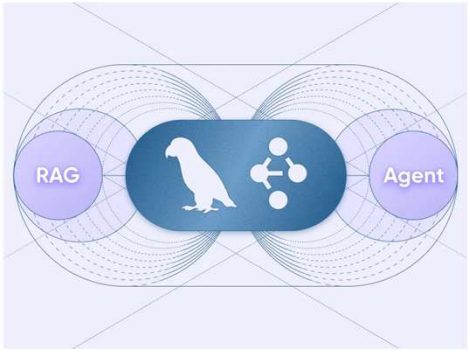
RAG 기본기부터 심화까지 제대로 끝내는 45시간 완성 로드맵, 테디노트의 '진짜' RAG 활용법!

#Agent #AI #LangChain #LangGraph #테디노트

강의 정보 *선택한 옵션에 따라 상이할 수 있습니다.

온라인 | 12파트 · 298클립 | 24.12.23 전체 공개 | 학습자료 제공
자막 제공 | 미리보기 제공 | AI 챗봇 지원 | 커뮤니티 운영

<FastCampus 강의 예시>
https://fastcampus.co.kr/data_online_teddy



CURRICULUM

01.

LangChain 입문

파트별 수강시간 04:28:12

CH01. LangChain 시작하기
01. 강의를 보는 방법
02. RAG(Retrieval-Augmented Generation) 기술이 주목받는 이유
03. 위키독스 전자책 활용방법
04. 실습코드(GitHub)

CH02. 환경 설정
01. 환경설치(Windows)
02. 환경설치(MacOS)
03. OpenAI API 키 발급 및 설정
04. LangSmith 키 발급 및 설정
05. Visual Syudio Code - User Settings(JSON) 설정

CH03. LangChain 시작하기
01. 랭체인(LangChain) 개요
02. Jupyter Notebook 사용법(단축키, 유용한팁)
03. LangChain 시작 (OpenAI 모델 알아보기)
04. 토큰(Token), 토큰계산기, 모델별 토큰 비용
05. 모델의 입출력, Context Window 개념 이해

CH04. LangChain 문법 맛보기
01. ChatOpenAI 주요 파라미터, invoke(), stream() 스트리밍 출력
02. LangSmith 로 GPT 추론내용 추적
03. GPT-4o (멀티모달) 모델로 이미지 인식하여 답변 출력
04. 프롬프트 템플릿(PromptTemplate)
05. LCEL 로 Chain 생성
06. 출력파서(StrOutputParser) 를 체인에 연결
07. Chain 추론과정을 LangSmith 에서 확인
08. LCEL 인터페이스 - 일괄처리 batch()
09. 비동기(asynchronous) 호출 방법
10. 병렬체인 구성(RunnableParallel)
11. 값을 전달해주는 RunnablePassthrough
12. 병렬로 Runnable 을 실행하는 RunnableParallel
13. 함수를 실행하는 RunnableLambda, itemgetter

CURRICULUM

02.

체인
파이프라인의
기본 요소

파트별 수강시간 09:26:50

CH01. 프롬프트
01. PromptTemplate, 부분변수(partial_variables)
02. yaml 파일로부터 프롬프트 템플릿 로드(load_prompt)
03. ChatPromptTemplate
04. MessagesPlaceHolder
05. 퓨샷프롬프트(FewShotPromptTemplate)
06. 예제 선택기(Example Selector)
07. MaxMarginalRelevance(MMR) 알고리즘
08. FewShotChatMessagePromptTemplate
09. 목적에 맞는 예제 선택기(CustomExampleSelector)
10. 프롬프트 허브(LangChain Hub)

CH02. 미니프로젝트
01. [프로젝트] Streamlit 으로 나만의 ChatGPT 웹앱 제작 (1)
02. [프로젝트] Streamlit 으로 나만의 ChatGPT 웹앱 제작 (2)
03. [프로젝트] Streamlit 으로 나만의 ChatGPT 웹앱 제작 (3)

CH03. RAG 시작하기
01. RAG 프로세스 이해하기 (사전단계)
02. RAG 프로세스 이해하기 (실행단계)
03. PDF 문서 기반 QA RAG
04. [프로젝트] 프롬프트를 개선해주는 프롬프트 메이커
05. [프로젝트] Page 분할, 파일 업로더 기능 추가
06. [프로젝트] PDF 기반 QA 챗봇
07. [프로젝트] LangSmith 추적, 다양한 LLM 을 RAG 에 적용
08. [프로젝트] 더 나은 답변을 위한 프롬프트 조정(출처, 표)

CH04. 출력 파서(Output Parser)
01. 출력파서(Output Parser)
02. PydanticOuputParser
03. with_structured_output() 바인딩
04. LangSmith 에서 OutputParser 흐름 확인
05. 콤마로 구분된 리스트 출력파서(CommaSeparatedListOutputParser)
06. 구조화된 출력파서(StructuredOutputParser)
07. JSON 형식 출력파서(JsonOutputParser)
08. Pandas DataFrame 출력파서(PandasDataFrameOutputParser)
09. 날짜형식 출력파서(DatetimeOutputParser)
10. 열거형 출력파서(EnumOutputParser)

CURRICULUM

02.

체인
파이프라인의
기본 요소

파트별 수강시간 09:26:50

CH05. 출력 파서 활용 프로젝트
01. 이메일 내용으로부터 구조화된 정보 추출 - (1)
02. SERP API 를 활용한 정보 검색의 활용 - (2)
03. 구조화된 답변을 다음 체인의 입력으로 추가하기 - (3)
04. 수신된 이메일의 주요 정보 및 검색 정보 기반 요약 보고서 챗봇 - (4)

CH06. 모델(Model)
01. RAG 에서의 LLM(Large Language Model)
02. 다양한 LLM과 활용방법
03. LLM의 답변을 캐싱(Cache)
04. 모델의 저장 및 로드(직렬화, 역직렬화)
05. 토큰 사용량 확인
06. Google Generative AI (Gemini Pro, Gemini Flash)
07. HuggingFace Inference API 활용
08. HuggingFace 의 템플릿 이해
09. HuggingFace Dedicated Endpoint 를 활용한 로컬모델 원격 호스팅
10. HuggingFace Local모델 다운로드 받아 추론
11. Ollama 설치 및 Modelfile 설정
12. Ollama 모델 생성, pull, run, ChatOllama 의 활용
13. GPT4ALL 로 로컬 모델 실행

CH07. 모델 활용 프로젝트
01. [프로젝트] 별도의 py 파일로 기능을 분리하는 방법
02. [프로젝트] Xionic 무료 모델을 GPT 대신 사용
03. [프로젝트] Ollama 모델을 사용한 RAG
04. [프로젝트] 멀티모달 모델을 활용한 이미지 인식 기반 챗봇

LangChain 참고 (2/2)

CURRICULUM

03.

체인의 기능 확장하기

파트별 수강시간 01:18:46

- **CH01. 메모리(Memory)**
- 01. 메모리란?
- 02. ConversationBufferMemory
- 03. ConversationBufferWindowMemory
- 04. ConversationTokenBufferMemory
- 05. ConversationEntityMemory
- 06. ConversationKGMemory
- 07. ConversationSummaryMemory
- 08. VectorStoreRetrieverMemory
- 09. LCEL Chain 에 메모리 추가
- 10. SQLite 에 대화내용 (사용자 및 대화별) 저장
- 11. 일반 변수에 대화내용 저장(휘발성 메모리)
- 12. [프로젝트] 이전 대화내용을 기억하는 멀티턴 챗봇

CURRICULUM

04.

다양한 형태의 데이터 로드

파트별 수강시간 01:34:34

- **CH01. 도큐먼트 로더(Document Loader)**
- 01. 도큐먼트 로더의 종류, 기본 구조, Document 구조
- 02. Document, Document Loader 의 구조 이해하기
- 03. PDF 로더
- 04. HWP 로더
- 05. CSV 로더 (효과적인 CSV 형식의 파일 처리 방법)
- 06. WebBase 로더(웹 정보 크롤링)
- 07. Directory 로더
- 08. Upstage LayoutAnalysis 로더
- 09. LlamaParser 로더(멀티모달을 활용한 이미지, 표 파싱 기능)

CURRICULUM

05.

Retrievel Augmented Generation: RAG

파트별 수강시간 07:00:25

- **CH01. 텍스트 분할(Text Splitter)**
- 01. 텍스트 분할의 개념과 중요성, 다양한 전략의 활용
- 02. CharacterTextSplitter
- 03. RecursiveCharacterTextSplitter
- 04. TokenTextSplitter
- 05. SemanticChunker
- 06. Code Splitter
- 07. MarkdownHeaderTextSplitter
- 08. HTMLHeaderTextSplitter
- 09. RecursiveJSONSplitter
- **CH02. 임베딩(Embedding)**
- 01. 임베딩(Embeddings) 개요
- 02. OpenAIEmbeddings
- 03. 캐시 임베딩(CacheBackedEmbeddings)
- 04. 로컬 모델 임베딩(HuggingFaceEmbeddings)
- 05. 업스테이지 임베딩(UpstageEmbeddings)
- 06. 올라마 임베딩(OllamaEmbeddings)
- **CH03. 벡터저장소(VectorStore)**
- 01. 벡터스토어(VectorStore) 개요
- 02. Chroma
- 03. FAISS
- 04. Pinecone

CURRICULUM

05.

Retrievel Augmented Generation: RAG

파트별 수강시간 07:00:25

- **CH04. 검색기(Retriever)**
- 01. 검색기(Retriever) 개요
- 02. 벡터스토어 기반 검색기(VectorStore-backed Retrieve
- 03. 문서 압축기(ContextualCompressionRetriever
- 04. 앙상블 검색기(Ensemble Retriever)
- 05. 긴 문서 재정렬(LongContext Reorder)
- 06. 부모 문서 검색기(ParentDocument Retriever)
- 07. 다중 쿼리 생성 검색기(MultiQuery Retriever)
- 08. 다중 벡터 검색기(MultiVector Retriever)
- 09. Self-Query Retriever
- 10. TimeWeightedVectorStore Retriever
- **CH05. 리랭커(Reranker)**
- 01. 리랭커(Reranker) 개요
- 02. CrossEncoderReranker(BGE-m3-reranker)
- 03. CohereReranker

CURRICULUM

06.

바로 사용할 수 있는 체인

파트별 수강시간 01:17:48

- **CH01. 사전에 정의된 체인(Chain)**
- 01. Stuff 요약
- 02. Map Reduce 요약
- 03. Map Refine 요약
- 04. Chain of Density(COD) 요약
- 05. Clustering Map Refine 요약
- 06. SQL 쿼리 생성기(create_sql_query_chain)

CURRICULUM

07.

LCEL 고급 문법

파트별 수강시간 01:14:51

- **CH01. LCEL 고급 문법**
- 01. RunnablePassthrough
- 02. Runnable 구조확인
- 03. RunnableLambda
- 04. 사용자의 질문 의도에 따라 Routing
- 05. RunnableParallel, itemgetter
- 06. 동적으로 LLM 이나 Prompt 를 변경하는 방법(Config)
- 07. 파이썬 함수에 chain 데코레이터로 runnable 설정

CURRICULUM

08.

RAG 평가 & 개선

파트별 수강시간 03:20:51

- **CH01. RAG 평가**
- 01. LLM 과 RAG 평가 방식, 평가 지표등 소개
- 02. RAGAS 소개
- 03. 합성 테스트 데이터셋(Synthetic Test Dataset) 생성
- 04. RAGAS 를 활용한 평가(Context Precision, Recall, Relevancy, Faithfulness)
- 05. 테스트 데이터셋 번역업로드 관리
- **CH02. LangSmith API 를 활용한 프롬프트 최적화**
- 01. LangSmith API 를 활용한 프롬프트 최적화
- 02. LLM-as-judge 사용방법(QA 평가자, Context-Answer 평가자)
- 03. LLM-as-judge 사용방법(Criteria 평가자)
- 04. LLM-as-judge 사용방법(Embedding 기반 평가자)
- 05. Custom LLM-as-judge 로 평가하는 방법
- 06. 한글 형태소 분석기
- 07. 휴리스틱 평가(Rouge, BLEU, METEOR, SemScore)
- 08. 실험별 비교 분석
- 09. 전체 수준을 평가하는 Summary 평가자
- 10. 할루시네이션(Groundedness) 평가
- 11. 실험 비교 분석을 위한 Pairwise 평가
- 12. 반복 평가
- 13. 온라인 LLM 평가를 활용한 평가 자동화

CURRICULUM

09.

Agent

파트별 수강시간 02:52:10

- **CH01. Tools / Toolkits**
- 01. 랭체인에서 제공하는 다양한 도구(Tools)툴킷(Toolkits) 활용법
- 02. 사용자 정의 도구(Custom Tools) 만드는법
- **CH02. Agent 주요기능**
- 01. LLM 에 도구 바인딩(Binding Tools)
- 02. Agent, AgentExecutor 생성 방법
- 03. Agent 의 중간단계 스트리밍(stream), AgentStreamParser
- 04. Agent 에 메모리 추가(멀티턴 구현)
- 05. 다양한 LLM 을 활용한 에이전트 생성(GPT, Claude, Gemini, TogetherAI, Ollama)
- 06. iter() 함수로 단계별 출력과 Human-in-the-loop
- **CH03. Agent 활용**
- 01. Agentic RAG
- 02. CSV, EXCEL 파일을 분석하는 데이터분석 Agent
- 03. (업무자동화) FileManagementToolkits 를 활용한 파일 관리 Agent
- 04. (업무자동화) 보고서 작성 Agent (web-search, retriever, file, image-generation)
- 05. [프로젝트] CSV 파일 기반 데이터분석 Agent

CURRICULUM

10.

LangGraph

파트별 수강시간 10:36:53

- **CH01. LangGraph 개요**
- 01. LangGraph 개요
- 02. LangGraph 세부 기능(State, Node, Edge, Conditional Edge, Compile)
- **CH02. LangGraph 핵심 기능**
- 01. LangGraph 에서 자주 사용되는 파이썬 문법(TypedDict, Annotated, Reducer)
- 02. LangGraph 챗봇 만들기
- 03. LangSmith 에서 LangGraph 추적 확인
- 04. Function Calling LLM 과 도구호출 노드, Conditional Edge 구축방법
- 05. LangGraph 에 메모리 추가(Checkpointer)
- 06. LangGraph 노드의 단계별 출력하는 Stream 모드(values, updates) 와 Interrupt
- 07. Human-in-the-loop 과 이전 상태에서 되돌림하는 Replay 기능
- 08. 중간 단계의 노드에서 상태를 임의 수정 후 재개(검색쿼리 수정)
- 09. 중간 단계의 노드에서 상태를 임의 수정 후 재개(검색쿼리 수정)
- 10. 전체 실행이 완료된 후 특정 노드로 되돌림하여 수정 후 Replay
- 11. 사람의 의견을 묻는 노드 추가
- 12. RemoveMessages 로 메시지 기록 삭제
- 13. ToolNode 활용법(2개 이상의 도구 사용)
- 14. 병렬 노드의 처리(fan-out, fan-in) 및 우선순위 설정
- 15. 대화 기록의 요약을 노드로 추가
- 16. 그래프에 서브 그래프(SubGraph) 를 추가하는 방법
- 17. 2개 이상의 SubGraph 추가 및 상태 변환(State Transformation)
- 18. LangGraph 출력 스트리밍(토큰 단위 스트리밍, tag 필터링)

CURRICULUM

10.

LangGraph

파트별 수강시간 10:36:53

- **CH03. LangGraph 구조 설계**
- 01. LangGraph 로 자유롭게 그래프 로직 구성(흐름 엔지니어링)
- 02. Naive RAG(전통적인 방식의 RAG) 를 LangGraph 로 구현
- 03. 답변의 할루시네이션관련성 평가 모듈 추가
- 04. 웹 검색 노드 추가
- 05. 쿼리 재작성 모듈 추가
- 06. [프로젝트] Agentic RAG - 에이전트를 활용한 RAG
- 07. [프로젝트] Adaptive RAG
- **CH04. LangGraph 유스케이스 연구**
- 01. [프로젝트] 에이전트간 대화 시뮬레이션(고객 응대 시나리오)
- 02. [프로젝트] 사용자 요구사항 기반 메타 프롬프트 생성 에이전트
- 03. [프로젝트] Corrective RAG (CRAG)
- 04. [프로젝트] Self-RAG
- 05. [프로젝트] Plan and Execute - 전문적인 보고서 작성
- 06. [프로젝트] 멀티에이전트 협업 - 웹 검색 및 차트 생성
- 07. [프로젝트] 멀티에이전트 감독자(Supervisor) - 웹 검색 및 차트 생성
- 08. [프로젝트] 계층적인 구조를 가지는 멀티에이전트 팀 협업 - 리서치팀과 문서 작성팀 협업
- 09. [프로젝트] SQL 에이전트 - (Text2SQL) SQL 쿼리 검색 기반 RAG
- 10. [프로젝트] Research 특화 멀티에이전트 STORM - 다인격 멀티에이전트
- 11. [프로젝트] GitHub 소스코드 기반 QA 챗봇 구현 - (1)
- 12. [프로젝트] GitHub 소스코드 기반 QA 챗봇 구현 - (2)
- 13. [프로젝트] Perplexity 클론 앱 구현