

- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: <https://smith.langchain.com/hub/teddynote/summary-stuff-documents>

## 5. 05. LLM-as-Judge

- LangSmith 에서 제공되는 **Off-the-shelf Evaluators** 활용해보기
  - Off-the-shelf Evaluators = **사전에 정의된 프롬프트 기반의 LLM 평가자**
  - 이점: 쉽게 사용 가능
  - 더 확장된 기능 사용하기 위해서는 **직접 평가자를 정의해야 함**
    - **input** = 질문: 보통 데이터셋의 **Question** 이 사용됨
    - **prediction** = LLM 이 생성한 답변: 보통 **모델의 답변** 이 사용됨
    - **reference** = **정답 답변**: **Context** 등 **변칙적** 으로 **활용** 가능
  - 참고: [How to define a code evaluator](#)

### • 환경 설정

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv
```

```
# API 키 정보 로드
load_dotenv()                                # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_API_KEY') else "설정되지 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}')" )
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}")
    print(" -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.")
else:
    print(f"❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f" - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다 (현재: '{langchain_tracing_v2}').")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print(" - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print(" - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

### • 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

## 1) RAG 성능 테스트를 위한 함수 정의

- **테스트 에 활용할 RAG** 시스템 생성하기

```
import os
from myrag import PDFRAG # local 임베딩 버전으로 수정한 myrag.py 불러오기
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_google_genai import ChatGoogleGenerativeAI

# API 키 확인
from dotenv import load_dotenv
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

if "GOOGLE_API_KEY" not in os.environ:
    print("❌ 경고: GOOGLE_API_KEY 환경 변수가 설정되지 않았습니다. 반드시 설정해야 gemini LLM이 작동합니다.")

# PDFRAG 객체 생성
# 이제 myrag.py가 OpenAIEmbeddings 대신 로컬 HuggingFaceEmbeddings를 사용하므로 Pydantic 오류가 사라짐
try:
    rag = PDFRAG(
        "../15_Evaluations/data/SPRI_AI_Brief_2023년12월호_F.pdf",
        ChatGoogleGenerativeAI(model="gemini-2.5-flash-lite", temperature=0)
    )

    print("\n✅ PDFRAG 객체 초기화 성공!")

except Exception as e:
    print(f"\n❌ RAG 실행 중 오류 발생: {e}")
    print("OPENAI_API_KEY 설정 또는 PDF 파일 경로('../15_Evaluations/data/SPRI_AI_Brief_2023년12월호_F.pdf')를 확인해주세요.")
```

- **gemini** 객체 생성됨

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1760440556.370227 7916400 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untrusted
```

- ✅ 문서 로드 완료: 23개 페이지
- ✅ 문서 분할 완료: 43개 청크
- ✅ 임베딩 모델 로드: all-MiniLM-L6-v2
- ✅ 벡터스토어 생성 완료
- ✅ PDFRAG 객체 초기화 성공!

```
# PDFRAG 객체 생성
from myrag import PDFRAG
from langchain_google_genai import ChatGoogleGenerativeAI
from dotenv import load_dotenv

load_dotenv()

llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0
)

rag = PDFRAG(
    "../15_Evaluations/data/SPRI_AI_Brief_2023년12월호_F.pdf",
    llm,
)

print("\n✅ PDFRAG 객체 생성")
```


- PDFRAG 객체 생성해보기

```
E0000 00:00:1760440568.556856 7916400 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untrusted
```

- ✅ 문서 로드 완료: 23개 페이지
- ✅ 문서 분할 완료: 43개 청크
- ✅ 임베딩 모델 로드: all-MiniLM-L6-v2
- ✅ 벡터스토어 생성 완료

-  PDFRAG 객체 생성

```
# 검색기(retriever) 생성
retriever = rag.create_retriever() # 10.5s
```

-  검색기 생성 완료 ( $k=4$ ) - (10.5s)

```
# 체인(chain) 생성
chain = rag.create_chain(retriever)
```

-  RAG 체인 생성 완료

```
# 질문
answer = chain.invoke("삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?")
print(answer)
```

- 삼성전자가 자체 개발한 생성형 AI의 이름은 '삼성 가우스'입니다.

```
print(chain)
```

- `print(chain)`

```
first={
  context: VectorStoreRetriever(tags=['FAISS', 'HuggingFaceEmbeddings'], vectorstore=<langchain_community.vectors
  question: RunnablePassthrough()
} middle=[PromptTemplate(input_variables=['context', 'question'], input_types={}, partial_variables={}, template=
```

- `ask_question()` 함수 생성하기

- 입력 = `inputs` = 딕셔너리
- 출력 = `answer` = 딕셔너리 → 반환

```
# 질문에 대한 답변하는 함수를 생성
def ask_question(inputs: dict):
    return {"answer": chain.invoke(inputs["question"])}
```

```
# 사용자 질문 예시
llm_answer = ask_question(
    {"question": "삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?"})

llm_answer
```

- `llm_answer` - (0.9s)

```
{'answer': '삼성전자가 자체 개발한 생성형 AI의 이름은 '삼성 가우스'입니다.'}
```

- `evaluator prompt` 출력 함수 정의하기

```
# evaluator prompt 출력을 위한 함수
def print_evaluator_prompt(evaluator):
    return evaluator.evaluator.prompt.pretty_print()
```

- next: 05. LLM-as-Judge-2