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

---

5. `05. LLM-as-Judge`

2) `Question-Answer Evaluator`

- 가장 기본 기능을 가진 `Evaluator` = `Query` - `Answer` 평가하기
- 사용자 입력 = `input` → `LLM`이 생성한 답변 → `prediction`으로 정답 답변은 `reference`로 정의됨
  - `Prompt`변수
    - `query` : 질문
    - `result` : `LLM` 답변
    - `answer` : 정답 답변

- **새로운 가상환경 생성 - `lc_eval_env`**
  - `Python-3.12`
  - `Pydantic` : `ver 1.10.18`

```
import pydantic
print(f"Pydantic 버전: {pydantic.__version__}")
```

숨겨진 출력 표시

- **`Pydantic 버전: 1.10.18`** - (`0.1s`)

- **`환경 설정`**

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

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

숨겨진 출력 표시

```
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("❌ 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.")
```

숨겨진 출력 표시

- 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.
```

```
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

GOOGLE_API_KEY=os.getenv("GOOGLE_API_KEY2")

if not os.getenv("GOOGLE_API_KEY2"):
    os.environ["GOOGLE_API_KEY2"] = input("Enter your GOOGLE_API_KEY2: ")

if "GOOGLE_API_KEY2" not in os.environ:
    print("❌ 경고: GOOGLE_API_KEY2 환경 변수가 설정되지 않았습니다. 반드시 설정해야 gemini LLM이 작동합니다.")
```

```
from langchain_google_genai import ChatGoogleGenerativeAI

llm2 = ChatGoogleGenerativeAI(model="gemini-2.5-flash-lite", temperature=0, api_key=os.getenv("GOOGLE_API_KEY2"))

print("✅ gemini-2.5-flash-lite 성공!")
```

```
✅ gemini-2.5-flash-lite 성공!
```

- **✅ 성공!**
  - *API 할당량 부족으로 두번째 계정으로 다시 시도*

```
from langchain_google_genai import ChatGoogleGenerativeAI

llm2 = ChatGoogleGenerativeAI(model="gemini-2.5-flash-lite", temperature=0, api_key=os.getenv("GOOGLE_API_KEY2"))

rag = PDFRAG(
    "../15_Evaluations/data/SPRI_AI_Brief_2023년12월호_F.pdf",
    llm2,
)

print("✅ PDFRAG 생성 성공!")
```

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

- **PDFRAG 생성** - (`4.9s`)

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

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

숨겨진 출력 표시

- **✅ 검색기 생성 완료 (k=4)**

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

숨겨진 출력 표시

- ✅ **RAG 체인 생성 완료**

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

숨겨진 출력 표시

- **삼성전자가 자체 개발한 생성형 AI의 이름은 '삼성 가우스'입니다.** - (2.0s)

```
# 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
```

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

- **ask_question()** - (1.2s)

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

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

```
from langsmith.evaluation import evaluate, LangChainStringEvaluator
from langchain_google_genai import ChatGoogleGenerativeAI

# qa 평가자 생성
qa_evalulator = LangChainStringEvaluator(
    "qa",
    config={
        "llm":ChatGoogleGenerativeAI(
            model="gemini-2.5-flash-lite",
            temperature=0,
            google_api_key=os.getenv("GOOGLE_API_KEY2"))
        }
)

# 프롬프트 출력
print_evaluator_prompt(qa_evalulator)
```

숨겨진 출력 표시

- **qa_evaluator** - (0.1s)

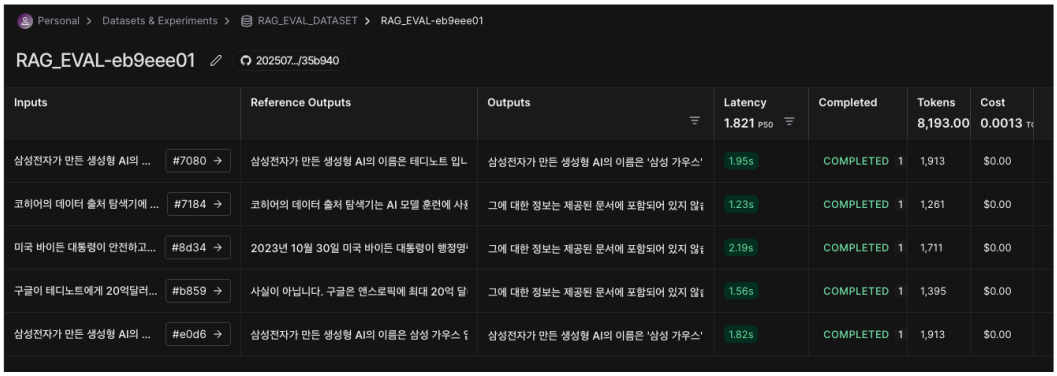
```
You are a teacher grading a quiz.
You are given a question, the student's answer, and the true answer, and are asked to score the student answer as

Example Format:
QUESTION: question here
STUDENT ANSWER: student's answer here
TRUE ANSWER: true answer here
GRADE: CORRECT or INCORRECT here

Grade the student answers based ONLY on their factual accuracy. Ignore differences in punctuation and phrasing be

QUESTION: *{query}*
STUDENT ANSWER: *{result}*
TRUE ANSWER: *{answer}*
GRADE:
```

- `평가 진행` → 출력한 `URL` 이동 → `결과 확인하기`

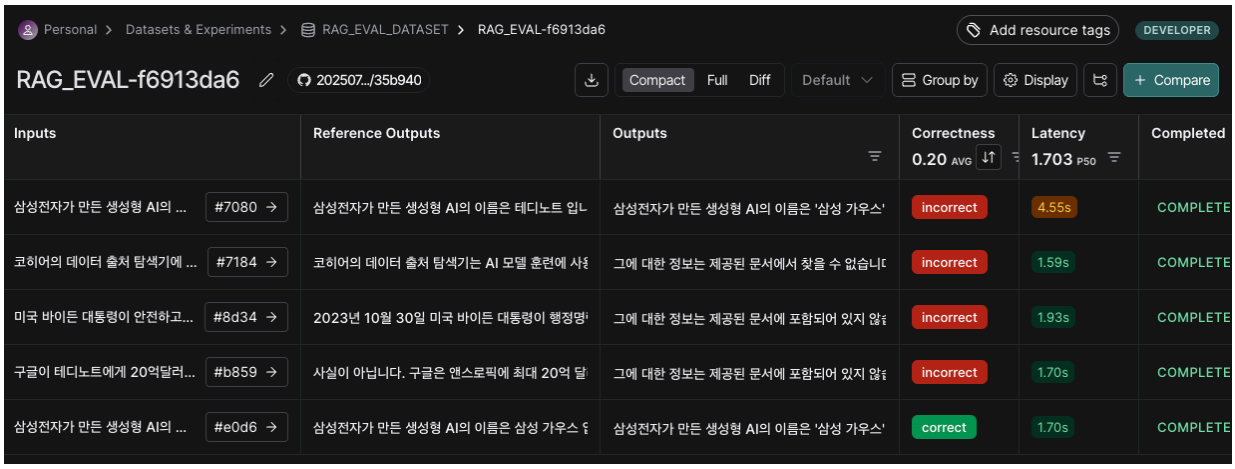
```
dataset_name = "RAG_EVAL_DATASET"

# 평가 실행
experiment_results = evaluate(
    ask_question,
    data=dataset_name,
    evaluators=[qa_evalulator],
    experiment_prefix="RAG_EVAL",
    # 실험 메타데이터 지정
    metadata={
        "variant": "QA Evaluator 를 활용한 평가",
    },
)
```

숨겨진 출력 표시

- 결과 확인하기
  - `RAG_EVAL` / `COMPILED` - (`45s`)



  - `RAG_EVAL` / `CORRECT=1`, `INCORRECT=4` - (`7.8s`)



- `RAG` 시스템 개선하기
  - a. `Retriever` 개선하기
  - b. `Chunk Size` 조정하기

```
# ======================================
# Chunk Size 조정
# ======================================
from myrag2 import PDFRAG
from langchain_google_genai import ChatGoogleGenerativeAI
import os
from dotenv import load_dotenv

load_dotenv()

# Chunk Size 조정
rag2 = PDFRAG(
    "../15_Evaluations/data/SPRI_AI_Brief_2023년12월호_F.pdf",
    ChatGoogleGenerativeAI(
```

```
        model="gemini-2.0-flash-lite",
        temperature=0,
        google_api_key=os.getenv("GOOGLE_API_KEY2")),
)
```

숨겨진 출력 표시

- `rag2 생성` - (`12.2s`)

```
✅ 문서 로드 완료: 23개 페이지
✅ 문서 분할 완료: 72개 청크
✅ 임베딩 모델 로드: all-MiniLM-L6-v2
✅ 벡터스토어 생성 완료
```

```
# ========================================
# Top-K 증가
# ========================================

# 개선
retriever2 = rag2.create_retriever()        # myrag2.py 불러오기

# 체인 재생성
chain2 = rag2.create_chain(retriever2)
```

숨겨진 출력 표시

- `✅ 검색기 생성 완료`
- `✅ RAG 체인 생성 완료`

```
# 질문에 대한 답변 생성
chain2.invoke("삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?")
```

숨겨진 출력 표시

- `'삼성전자가 자체 개발한 생성형 AI의 이름은 제공된 문서에서 찾을 수 없습니다.'` - (`1.4s`)

```
# 질문에 대한 답변하는 함수를 생성
def ask_question(inputs: dict):
    return {"answer": chain2.invoke(inputs["question"])}

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

llm_answer
```

```
{'answer': '삼성전자가 자체 개발한 생성형 AI의 이름은 제공된 문서에서 찾을 수 없습니다.'}
```

- `ask_question()` - (`1.3s`)

```
{'answer': '삼성전자가 자체 개발한 생성형 AI의 이름은 제공된 문서에서 찾을 수 없습니다.'}
```

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

```
from langsmith.evaluation import evaluate, LangChainStringEvaluator
from langchain_google_genai import ChatGoogleGenerativeAI


# qa 평가자 생성
qa_evalulator = LangChainStringEvaluator(
    "qa",
    config={
        "llm":ChatGoogleGenerativeAI(
            model="gemini-2.5-flash-lite",
            temperature=0,
            google_api_key=os.getenv("GOOGLE_API_KEY2"))
        }
```

```
)


# 프롬프트 출력
print_evaluator_prompt(qa_evalulator)
```

```
You are a teacher grading a quiz.
You are given a question, the student's answer, and the true answer, and are asked to score the student answer as either CORRECT

Example Format:
QUESTION: question here
STUDENT ANSWER: student's answer here
TRUE ANSWER: true answer here
GRADE: CORRECT or INCORRECT here

Grade the student answers based ONLY on their factual accuracy. Ignore differences in punctuation and phrasing between the studer

QUESTION: {query}
STUDENT ANSWER: {result}
TRUE ANSWER: {answer}
GRADE:
```

```
dataset_name = "RAG_EVAL_DATASET"

# 평가 실행
experiment_results = evaluate(
    ask_question,
    data=dataset_name,
    evaluators=[qa_evalulator],
    experiment_prefix="RAG_EVAL_2",
    # 실험 메타데이터 지정
    metadata={
        "variant": "myrag2.py 반영",
    },
)
```

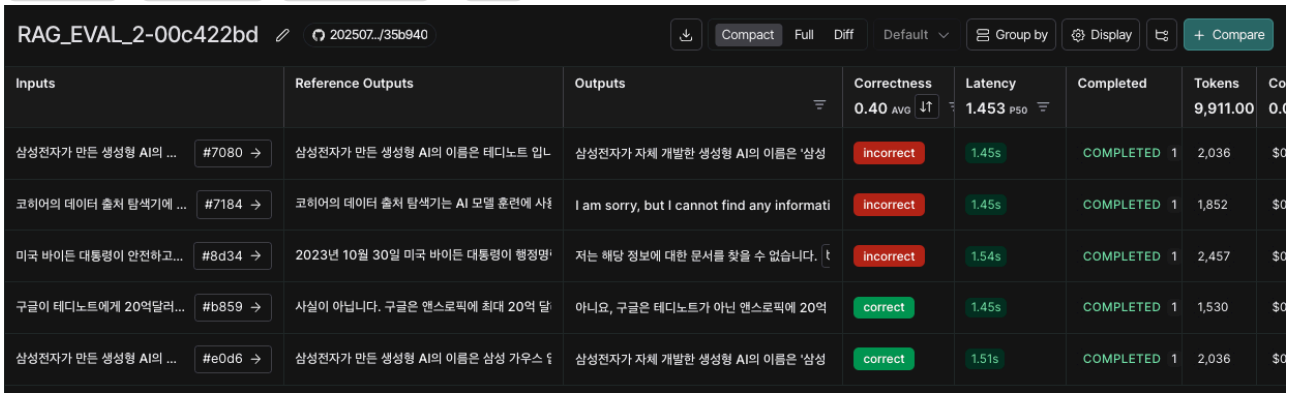
숨겨진 출력 표시

- **개선 시도:** `RAG_EVAL_2`

```
View the evaluation results for experiment: 'RAG_EVAL_2-00c422bd' at:
https://smith.langchain.com/o/2c3342d3-1170-4ffa-86fd-f621199e0b9c/datasets/420dd308-2ebd-44c9-8ce8-9aff3886dc8e/


0it [00:00, ?it/s]

5it [00:04,  1.12it/s]
```

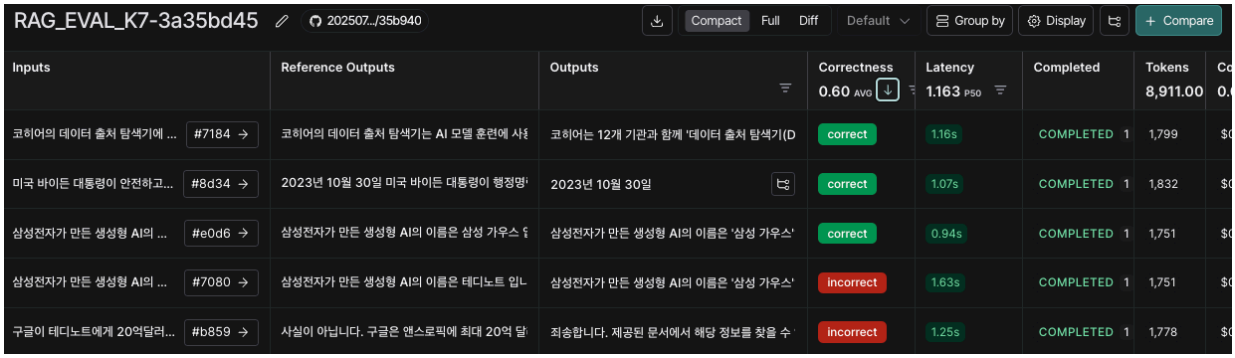
- 결과 확인하기

*

- `RAG_EVAL` / *`CORRECT=2`*, *`INCORRECT=3`* - (*`7.3s`*)
- 

- `RAG` 시스템 개선하기_2
  - **c.** `chunk_size` **조정**
  - **d. 잦은 커널 충돌 →** `python` **파일로 실행하기**

- c.
  - `chunck_size` 조절 test
    - **`chunk_size` 지정** ❌ or **`chunk_size` = `4` → 검색 결과 없음**
      - *`'죄송합니다. 제공된 문서에서 삼성전자가 자체 개발한 생성형 AI의 이름에 대한 정보를 찾을 수 없습니다.'`*
    - **`chunk_size` = `7` → 검색 결과** ⭕
      - *`"삼성전자가 자체 개발한 생성형 AI의 이름은 '삼성 가우스'입니다."`*
  - **`myrag4.py`** 로 정리

- d.
  - **`잦은 커널 충돌로 인한 문제` → `python` 파일로 실행 → 터미널에서 직접 확인하기**
  - **`eval_script.py`** 실행 → 터미널에서 결과 바로 확인
  - 


```
(가상환경)/15_Evaluations/eval_script.py

🚀 RAG 시스템 초기화 시작...

📂 스크립트 위치: ... /15_Evaluations

📄 PDF 경로: ... /data/SPRI AI Brief 2023년12월호_F.pdf

✅ 파일 확인 완료!

✅ LLM 생성 완료: gemini-2.5-flash-lite

✅ PDF 로드 완료: 23 페이지

✅ 청크 분할 완료: 119 청크 (크기=300, 오버랩=50)


✅ LLM 생성 완료: gemini-2.5-flash-lite

✅ 청크 분할 완료: 119 청크 (크기=300, 오버랩=50)
pytorch_model.bin: 100%|████████████████████████████████████████████████████████| 2.27G/2.27G [00:03
tokenizer_config.json: 100%|████████████████████████████████████████████████████| 444/444 [00:00<
sentencepiece.bpe.model: 100%|██████████████████████████████████████████████████| 5.07M/5.07M [00:00
tokenizer.json: 100%|████████████████████████████████████████████████████████████| 17.1M/17.1M [00:06<
special_tokens_map.json: 100%|███████████████████████████████████████████████████| 964/964 [00:00<
config.json: 100%|██████████████████████████████████████████████████████████████| 191/191 [00:00

✅ 임베딩 모델 로드 완료: BAAI/bge-m3                                              | 0.00/191 [00:
model.safetensors:   0%|                                                          | 2.19M/2.27G [00:24<1

✅ 검색기 생성 완료 (k=7, search_type=similarity)

✅ RAG 체인 생성 완료

================================================
🧪 테스트 실행...
================================================

View the evaluation results for experiment: 'RAG_EVAL_K7-3a35bd45' at:
https://smith.langchain.com/o/2c3342d3-1170-4ffa-86fd-f621199e0b9c/datasets/420dd308-2ebd-44c9-8ce8-9aff3886dc8e
```

```
5it [00:14,  2.90s/it]
5it [00:14,  2.18s/it]

================================================
✅ 평가 완료!
================================================

결과: <ExperimentResults RAG_EVAL_K7-3a35bd45>

model.safetensors: 100%|██████████████████████████████████████████████████████████| 2.27G/2.27G [02:14<
```

- next: ***05. LLM-as-Judge-3***
