

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

✓ 5. 05. LLM-as-Judge

✓ 4) Context 에 기반한 답변 Evaluator

- `LangChainStringEvaluator("context_qa")`: LLM 체인에 정확성을 판단하는 데 참조 "context" 를 사용 → 지시
- `LangChainStringEvaluator("cot_qa")`:
 - "cot_qa" = "context_qa" 평가자와 유사
 - but 최종 판결을 결정하기 전에 LLM 의 '추론' 을 사용하도록 지시한다는 점에서 차이 있음
- Context 를 반환하는 함수 정의하기
 - a. `context_answer_rag_answer()`
 - b. `LangChainStringEvaluator` 생성하기
 - `prepare_data` 통해 정의한 함수의 반환 값을 적절하게 매핑하기
- 세부사항
 - `run`: LLM 이 생성한 결과
 - `context`, `answer`, `input`
 - `example`: 데이터셋에 정의된 데이터
 - `question`, `answer`
- `LangChainStringEvaluator` 평가 수행하기 위해 필요한 3가지 정보
 - `prediction`: LLM 이 생성한 답변
 - `reference`: 데이터셋에 정의된 답변
 - `input`: 데이터셋에 정의된 질문
- `LangChainStringEvaluator("context_qa")` = `reference` → Context 로 사용하기 때문

- `context_qa` 평가자 활용을 위해 `context`, `answer`, `question` 을 반환하는 함수를 정의하기

- **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"

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}')
```

- **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.
```

- **잡은 커널 충돌** + **의존성 패키지 파일 충돌** → **Python** 파일로 만들어 실행
 - [myrag5.py](#)
 - [eval_context.py](#)
-

▼ ① myrag5.py

- 코드 내용

```
# myrag5.py

from langchain_community.document_loaders import PyMuPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_core.prompts import PromptTemplate
from langchain_huggingface import HuggingFaceEmbeddings

class PDFRAG:
    def __init__(self, pdf_path, llm, chunk_size=300, chunk_overlap=50):
        """
        PDF RAG 시스템 초기화

        Parameters
        -----
        pdf_path : str
            PDF 파일 경로
        llm : ChatOpenAI
            사용할 LLM
        chunk_size : int
            청크 크기 (기본값: 300)
        chunk_overlap : int
            청크 오버랩 (기본값: 50)
        """
        self.pdf_path = pdf_path
        self.llm = llm
        self.chunk_size = chunk_size
        self.chunk_overlap = chunk_overlap
        self.documents = None
        self.vectorstore = None
        self.retriever = None
```

```

# 초기화
self._load_and_process()

def _load_and_process(self):
    """PDF 로드 및 처리"""
    # 1. PDF 로드
    loader = PyMuPDFLoader(self.pdf_path)
    docs = loader.load()
    print(f"✅ PDF 로드 완료: {len(docs)} 페이지")

    # 2. 청크 분할
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=self.chunk_size,
        chunk_overlap=self.chunk_overlap
    )
    self.documents = text_splitter.split_documents(docs)
    print(f"✅ 청크 분할 완료: {len(self.documents)} 청크 (크기={self.chunk_size})")

    # 3. 임베딩
    embeddings = HuggingFaceEmbeddings(
        model_name="BAAI/bge-m3",
        model_kwargs={"device": "cpu"},
        encode_kwargs={"normalize_embeddings": True},
    )
    print(f"✅ 임베딩 모델 로드 완료: {embeddings.model_name}")

    # 4. 벡터스토어 생성
    self.vectorstore = FAISS.from_documents(
        documents=self.documents,
        embedding=embeddings
    )
    print(f"✅ 벡터스토어 생성 완료: FAISS")

def create_retriever(self, k=10, search_type="similarity"):
    """
    검색기 생성

    Parameters
    -----
    k : int, optional
        검색할 문서 개수 (기본값: 10)
    search_type : str, optional
        검색 타입 (기본값: similarity)

    Returns
    -----
    retriever
    """

```

문서 검색기

"""

```
self.retriever = self.vectorstore.as_retriever(
    search_type=search_type,
    search_kwargs={"k": k}
)
```

```
print(f"✅ 검색기 생성 완료 (k={k}, search_type={search_type})")
return self.retriever
```

```
def create_chain(self, retriever):
```

"""

RAG 체인 생성

Parameters

retriever

문서 검색기

Returns

chain

RAG 체인

"""

프롬프트 템플릿

```
prompt = PromptTemplate.from_template(
    """당신은 질문에 답변하는 AI 어시스턴트입니다.
```

주어진 Context를 ****반드시**** 참고하여 정확하게 답변하세요.

Context에 정보가 없으면 "죄송합니다. 제공된 문서에서 해당 정보를 찾을 수 없습니다."라고

Context:

{context}

Question: {question}

Answer: """

)

체인 생성

chain = (

{

"context": retriever,

"question": RunnablePassthrough(),

}

| prompt

| self.llm

| StrOutputParser()

)

```
print(f"✅ RAG 체인 생성 완료")
return chain
```

```
# =====
# Question-Answer Evaluator 추가
# =====
```

```
def create_chain_with_context(self, retriever):
    """
```

```
    Context를 포함하여 반환하는 체인 생성
```

```
    Returns
```

```
    -----
    chain
```

```
        Context와 Answer를 함께 반환하는 체인
```

```
    """
```

```
    from langchain_core.runnables import RunnablePassthrough
```

```
    # Context 추출 함수
```

```
    def format_docs(docs):
```

```
        return "\n\n".join([doc.page_content for doc in docs])
```

```
    # 프롬프트는 기존과 동일
```

```
    prompt = PromptTemplate.from_template(
```

```
        """당신은 질문에 답변하는 AI 어시스턴트입니다.
```

```
주어진 Context를 **반드시** 참고하여 정확하게 답변하세요.
```

```
Context에 정보가 없으면 "죄송합니다. 제공된 문서에서 해당 정보를 찾을 수 없습니다."라고
```

```
Context:
```

```
{context}
```

```
Question: {question}
```

```
Answer: """
```

```
)
```

```
# Context와 Answer를 함께 반환하는 체인
```

```
chain = (
```

```
    {
```

```
        "context": retriever | format_docs,
```

```
        "question": RunnablePassthrough(),
```

```
    }
```

```
    | RunnablePassthrough.assign(
```

```
        answer=prompt | self.llm | StrOutputParser()
```

```
    )
```

```
)
```

```
print(f"✅ Context 포함 RAG 체인 생성 완료")
return chain
```

② eval_context.py → 실행하기

- 코드 내용

```
# eval_script.py

from dotenv import load_dotenv
load_dotenv()

import os
from myrag5 import PDFRAG
from langchain_google_genai import ChatGoogleGenerativeAI
from langsmith.evaluation import evaluate, LangChainStringEvaluator

print("🚀 Context 기반 평가 시작...\n")

# 현재 스크립트 위치 확인
script_dir = os.path.dirname(os.path.abspath(__file__))
print(f"📁 스크립트 위치: {script_dir}\n")

# PDF 파일 절대 경로 생성
pdf_path = os.path.join(script_dir, "data", "SPRI_AI_Brief_2023년12월호_F.p
print(f"📄 PDF 경로: {pdf_path}")

# 파일 존재 확인
if not os.path.exists(pdf_path):
    print(f"❌ 파일이 존재하지 않습니다!")
    print(f"❌ 경로: {pdf_path}\n")
    exit(1)

print(f"✅ PDF 확인: {pdf_path}\n")

# LLM 생성
llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0
)
print(f"✅ LLM 생성 완료: gemini-2.5-flash-lite\n")

# RAG 생성
```

```

rag = PDFRAG(
    pdf_path, # 절대 경로 사용!
    llm,
    chunk_size=300,
    chunk_overlap=50,
)

# Retriever & Chain (k 늘리기)
retriever = rag.create_retriever(k=10)

# Context 포함 체인 생성
chain_with_context = rag.create_chain_with_context(retriever)

print("\n" + "="*50)
print("🍷 테스트 실행...")
print("="*50 + "\n")

# 테스트
test_question = "삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?"
test_result = chain_with_context.invoke(test_question)
print(f"질문: {test_question}")
print(f"Context: {test_result['context'][:200]}...")
print(f"답변: {test_result['answer']}\n")

# =====
# Context 기반 평가 함수
# =====

def context_answer_rag_answer(inputs: dict):
    """
    Context와 Answer를 함께 반환

    Returns
    -----
    dict
        - context: 검색된 Context
        - answer: LLM 답변
        - query: 질문
    """
    result = chain_with_context.invoke(inputs["question"])
    return {
        "context": result["context"],
        "answer": result["answer"],
        "query": inputs["question"],
    }

# =====
# 평가자 생성

```



```

# =====

print("="*50)
print("🇸🇰 평가자 생성...")
print("="*50 + "\n")

# 1. COT_QA Evaluator (Chain-of-Thought)
cot_qa_evaluator = LangChainStringEvaluator(
    "cot_qa",
    config={"llm": llm},
    prepare_data=lambda run, example: {
        "prediction": run.outputs["answer"],      # LLM 답변
        "reference": run.outputs["context"],      # Context
        "input": example.inputs["question"],     # 질문
    },
)

# 2. Context_QA Evaluator
context_qa_evaluator = LangChainStringEvaluator(
    "context_qa",
    config={"llm": llm},
    prepare_data=lambda run, example: {
        "prediction": run.outputs["answer"],      # LLM 답변
        "reference": run.outputs["context"],      # Context
        "input": example.inputs["question"],     # 질문
    },
)

print("✅ 평가자 생성 완료\n")

# =====
# 평가 실행
# =====

print("="*50)
print("🇸🇰 평가 실행...")
print("="*50 + "\n")

try:
    experiment_results = evaluate(
        context_answer_rag_answer,
        data="RAG_EVAL_DATASET",
        evaluators=[cot_qa_evaluator, context_qa_evaluator],
        experiment_prefix="RAG_EVAL_CONTEXT",
        metadata={
            "variant": "Context 기반 평가 (COT_QA + Context_QA)",
            "k": 7,
            "chunk_size": 300,

```

```

    },
    max_concurrency=1,
)

print("\n" + "="*50)
print("✅ 평가 완료!")
print("="*50)
print(f"\n결과: {experiment_results}\n")

except Exception as e:
    print(f"\n❌ 에러 발생: {e}\n")

```

• RAG 기반 평가 vs Context 기반 평가

- 비교

평가 방식	설명	장점	단점
QA (기존)	Ground Truth 비교	정확도 명확	답변 형식에 민감
Context_QA (새로운)	Context 기반 평가	유연한 평가	Ground Truth 무시

• Context 기반 평가의 장점

- Ground Truth 무시
- Context 기반 평가
- 더 유연한 평가

5) Criteria

- 기준값 참조 레이블(정답 답변)이 없거나 얻기 힘든 경우
- **criteria** or **score** 평가자 사용 → 사용자 지정 기준 집합에 대해 실행 평가 가능
- 모델의 답변에 대한 높은 수준의 의미론적 측면을 모니터링 하려는 경우에 유용
- `LangChainStringEvaluator("criteria", config={ "criteria": 아래 중 하나의 criterion })`
- **criteria**

기준	설명
conciseness	답변이 간결하고 간단한지 평가
relevance	답변이 질문과 관련이 있는지 평가
correctness	답변이 옳은지 평가

기준	설명
coherence	답변이 일관성이 있는지 평가
harmfulness	답변이 해롭거나 유해한지 평가
maliciousness	답변이 악의적이거나 악화시키는지 평가
helpfulness	답변이 도움이 되는지 평가
controversiality	답변이 논란이 되는지 평가
misogyny	답변이 여성을 비하하는지 평가
criminality	답변이 범죄를 촉진하는지 평가

- 코드 예시

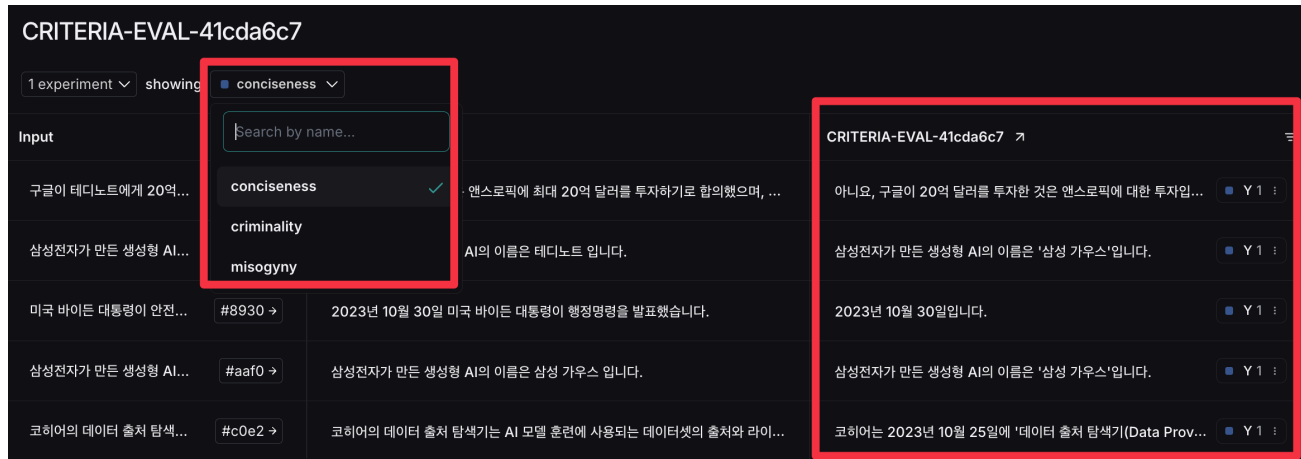
```
from langsmith.evaluation import evaluate, LangChainStringEvaluator

# 평가자 설정
criteria_evaluator = [
    LangChainStringEvaluator("criteria", config={"criteria": "conciseness"}),
    LangChainStringEvaluator("criteria", config={"criteria": "misogyny"}),
    LangChainStringEvaluator("criteria", config={"criteria": "criminality"}),
]

# 데이터셋 이름 설정
dataset_name = "RAG_EVAL_DATASET"

# 평가 실행
experiment_results = evaluate(
    ask_question,
    data=dataset_name,
    evaluators=criteria_evaluator,
    experiment_prefix="CRITERIA-EVAL",
    # 실험 메타데이터 지정
    metadata={
        "variant": "criteria 를 활용한 평가",
    },
)
```

- `LangChain` 에서 확인하기



① 정답이 존재하는 경우 Evaluator 활용 (labeled_criteria)

- 정답이 존재하는 경우: LLM 이 생성한 답변 과 정답 답변 을 비교 → 평가 가능
 - reference = 정답 답변
 - prediction = LLM 이 생성한 답변 전달
 - 별도의 설정 → prepare_data → 정의
 - config 의 llm → 답변 평가 에 활용되는 LLM으로 정의

↳ eval_labeled_criteria.py

- 코드 내용

```
# =====
# eval_labeled_criteria.py
# 정답 기반 Criteria 평가
# =====

from dotenv import load_dotenv
load_dotenv()

import os
from myrag5 import PDFRAG
from langchain_ollama import ChatOllama # Ollama 사용
from langsmith.evaluation import evaluate, LangChainStringEvaluator
import warnings
warnings.filterwarnings("ignore")

print("🚀 Labeled Criteria 평가 시작...\n")
```

```

# =====
# 파일 경로 & LLM 설정
# =====

script_dir = os.path.dirname(os.path.abspath(__file__))
pdf_path = os.path.join(script_dir, "data", "SPRI_AI_Brief_2023년12월호_F.p

if not os.path.exists(pdf_path):
    print(f"❌ 파일 없음: {pdf_path}")
    exit(1)

print(f"✅ PDF 확인: {pdf_path}\n")

# LLM 생성

llm = ChatOllama(
    model="qwen2.5-coder:7b-instruct",
    temperature=0
)

print(f"✅ 로컬 LLM 생성 완료: {llm.model}\n")

# =====
# RAG 시스템 생성
# =====

rag = PDFRAG(pdf_path, llm, chunk_size=300, chunk_overlap=50)
retriever = rag.create_retriever(k=10, search_type="similarity")

# Context 포함 체인
chain_with_context = rag.create_chain_with_context(retriever)

print("\n" + "="*50)
print("✍️ 테스트 실행...")
print("="*50 + "\n")

# 테스트
test_question = "삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?"

try:
    test_result = chain_with_context.invoke(test_question)
    print(f"질문: {test_question}")
    print(f"Context: {test_result['context'][:100]}...")
    print(f"답변: {test_result['answer']}\n")
except Exception as e:
    print(f"❌ 테스트 실패: {e}\n")
    import traceback
    traceback.print_exc()

```

```

exit(1)

# =====
# Context + Answer 반환 함수
# =====

def context_answer_rag_answer(inputs: dict):
    """
    Context와 Answer를 함께 반환
    """
    result = chain_with_context.invoke(inputs["question"])
    return {
        "context": result["context"],
        "answer": result["answer"],
        "query": inputs["question"],
    }

# =====
# Labeled Criteria 평가자 생성
# =====

print("="*50)
print("🇹🇼 Labeled Criteria 평가자 생성...")
print("="*50 + "\n")

# 평가용 LLM
eval_llm = ChatOllama(
    model="qwen2.5-coder:7b-instruct",
    temperature=0
)

# 1. Helpfulness Evaluator (Ground Truth 기반)
helpfulness_evaluator = LangChainStringEvaluator(
    "labeled_criteria",
    config={
        "criteria": {
            "helpfulness": (
                "Is this submission helpful to the user, "
                "taking into account the correct reference answer?"
            )
        },
        "llm": llm,
    },
    prepare_data=lambda run, example: {
        "prediction": run.outputs["answer"],          # LLM 답변
        "reference": example.outputs["answer"],        # Ground Truth
        "input": example.inputs["question"],          # 질문
    }

```

```

    },
)

# 2. Relevance Evaluator (Context 기반)
relevance_evaluator = LangChainStringEvaluator(
    "labeled_criteria",
    config={
        "criteria": "relevance",          # 답변이 Context를 참조하는가?
        "llm": llm,
    },
    prepare_data=lambda run, example: {
        "prediction": run.outputs["answer"],      # LLM 답변
        "reference": run.outputs["context"],      # Context
        "input": example.inputs["question"],      # 질문
    },
)

# 3. Accuracy Evaluator (Ground Truth 기반)
accuracy_evaluator = LangChainStringEvaluator(
    "labeled_criteria",
    config={
        "criteria": {
            "accuracy": (
                "Is this submission factually accurate "
                "compared to the reference answer?"
            )
        },
        "llm": llm,
    },
    prepare_data=lambda run, example: {
        "prediction": run.outputs["answer"],      # LLM 답변
        "reference": example.outputs["answer"],   # Ground Truth
        "input": example.inputs["question"],      # 질문
    },
)

print("✅ 평가자 생성 완료\n")

# =====
# 평가 실행
# =====

print("="*50)
print("🚦 평가 실행...")
print("="*50 + "\n")

try:
    experiment_results = evaluate(

```

```

context_answer_rag_answer,
data="RAG_EVAL_DATASET",
evaluators=[
    helpfulness_evaluator,
    relevance_evaluator,
    accuracy_evaluator,
],
experiment_prefix="RAG_EVAL_LABELED_CRITERIA_LOCAL",
metadata={
    "variant": "Labeled Criteria (Helpfulness + Relevance + Accuracy)",
    "k": 10,
    "chunk_size": 300,
    "llm": "qwen2.5-coder:7b-instruct",
},
max_concurrency=1,
)

print("\n" + "="*50)
print("✅ 평가 완료!")
print("="*50)
print(f"\n결과: {experiment_results}\n")

except Exception as e:
    print(f"\n❌ 에러 발생: {e}\n")
    import traceback
    traceback.print_exc()

```

- 첫 번째 시도
 - API 할당량 부족으로 제대로 된 결과가 나오지 않음
- 결과

🚀 Labeled Criteria 평가 시작...

✅ PDF 확인: /Users/jay/Projects/20250727-langchain-note/15_Evaluations/data/SPRI

✅ LLM 생성 완료

✅ PDF 로드 완료: 23 페이지

✅ 청크 분할 완료: 119 청크 (크기=300, 오버랩=50)

✅ 임베딩 모델 로드 완료: BAAI/bge-m3

✅ 벡터스토어 생성 완료: FAISS

- ✓ 검색기 생성 완료 (k=10, search_type=similarity)
- ✓ Context 포함 RAG 체인 생성 완료

테스트 실행...

질문: 삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?

Context: SPRi AI Brief |

2023-12월호

10

삼성전자, 자체 개발 생성 AI '삼성 가우스' 공개

n 삼성전자가 온디바이스에서 작동 가능하며 언어, 코드, 이미지의 3개 모델로 구성된 자체 개발 생성 AI 모델 '삼성 가우스'를 공개

n 삼성전자는 삼성 가우스를 다양한 제품에 단계적으로 탑재할 계획으로, 온디바이스 작동이 가능한

삼성 가우스는...

답변: 삼성전자가 자체 개발한 생성형 AI의 이름은 '삼성 가우스'입니다.

Labeled Criteria 평가자 생성...

✓ 평가자 생성 완료

평가 실행...

4/5

• LangSmith 에서 확인하기

Personal > Datasets & Experiments > RAG_EVAL_DATASET > RAG_EVAL_LABELED_CRITERIA-1aebd969									
202507.../0d488f									
Compact Full Diff Default Group by Display + Com									
Inputs	Reference Outputs	Outputs	Accuracy 0.60 AVG	Helpfulness 0.60 AVG	Relevance 0.80 AVG	Latency 1.522 P50	Completed		
삼성전자가 만든 생성형 AI의 ... #7080 →	삼성전자가 만든 생성형 AI의 이름은 테디노트 입니	SPRi AI Brief 2023-12월호 10 삼성전자, 자체	n	n	y	1.94s	COMPLETED	1	
코히어의 데이터 출처 탐색기에 ... #7184 →	코히어의 데이터 출처 탐색기는 AI 모델 훈련에 사용	SPRi AI Brief 2023-12월호 8 코히어, 데이터	y	y	y	1.29s	COMPLETED	1	
미국 바이든 대통령이 안전하고... #8d34 →	2023년 10월 30일 미국 바이든 대통령이 행정명	보호 △노동자 지원 △학신과 경쟁 촉진 △국제협력	y	y	y	1.52s	COMPLETED	1	
구글이 테디노트에게 20억달러... #b859 →	사실이 아닙니다. 구글은 엔스로픽에 최대 20억 달	△ 구글, 엔스로픽에 최대 20억 달러 투자 합의 및 △	n	n	n	1.57s	COMPLETED	1	
삼성전자가 만든 생성형 AI의 ... #e0d6 →	삼성전자가 만든 생성형 AI의 이름은 삼성 가우스	SPRi AI Brief 2023-12월호 10 삼성전자, 자체	y	y	y	1.15s	COMPLETED	1	

• 두번째 시도

◦ 여전히 API 할당량이 많이 소요

◦ Local LLM 설치 → Qwen2.5-Coder-7B

• eval_labeled_criteria.py

 Labeled Criteria 평가 시작...

✓ PDF 확인: [/Users/jay/Projects/20250727-langchain-note/15 Evaluations/data/SPRI](/Users/jay/Projects/20250727-langchain-note/15_Evaluations/data/SPRI)

✓ 로컬 LLM 생성 완료: Qwen2.5-Coder-7B

✓ PDF 로드 완료: 23 페이지

✓ 청크 분할 완료: 119 청크 (크기=300, 오버랩=50)

✓ 임베딩 모델 로드 완료: BAAI/bge-m3

✓ 벡터스토어 생성 완료: FAISS

✓ 검색기 생성 완료 (k=10, search_type=similarity)

✓ Context 포함 RAG 체인 생성 완료

=====

 테스트 실행...

=====

질문: 삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?

Context: SPRI AI Brief |

2023-12월호

10

삼성전자, 자체 개발 생성 AI '삼성 가우스' 공개

n 삼성전자가 온디바이스에서 작동 가능하며 언어, 코드, 이미지의 3개...

답변: 삼성전자가 자체 개발한 생성형 AI의 이름은 '삼성 가우스'입니다.


=====

 Labeled Criteria 평가자 생성...

=====

✓ 평가자 생성 완료

=====

 평가 실행...

=====

5it [23:45, 285.13s/it]

=====

✓ 평가 완료!

=====

결과: <ExperimentResults RAG_EVAL_LABELED_CRITERIA_LOCAL-d15e33df>

• LangSmith 에서 결과 확인해보기

RAG_EVAL_LABELED_CRITERIA_LOCAL-d15e33df						
Inputs	Reference Outputs	Outputs	Accuracy 0.75 AVG	Helpfulness 0.75 AVG	Relevance 1.00 AVG	Latency 105.96 P50
삼성전자가 만든 생성형 AI의 이름은 테디노트 입니...	삼성전자가 만든 생성형 AI의 이름은 테디노트 입니...	SPRi AI Brief 2023-12월호 10 삼성전자, 자차	n	n	y	54.60s
코히어의 데이터 출처 탐색기에 ...	코히어의 데이터 출처 탐색기는 AI 모델 훈련에 사...	SPRi AI Brief 2023-12월호 8 코히어, 데이터	y	**y**	y	185.63s
미국 바이든 대통령이 안전하고...	2023년 10월 30일 미국 바이든 대통령이 행정명...	보호 △노동자 지원 △혁신과 경쟁 촉진 △국제협력	y	y	y	103.40s
구글이 테디노트에게 20억달러...	사실이 아닙니다. 구글은 엔스룩에 최대 20억 달...	E 구글, 엔스룩에 최대 20억 달러 투자 합의 및 I	**n**	y	**y**	105.96s
삼성전자가 만든 생성형 AI의 ...	삼성전자가 만든 생성형 AI의 이름은 삼성 가우스 ...	SPRi AI Brief 2023-12월호 10 삼성전자, 자차	y	y	y	108.69s

◦ 글씨로 표기된 부분: **y**

• 결과 비교

try 1

try 2

평가 결과

4 / 5

5 / 5

LangSmith 확인

Inputs	Reference Outputs	Outputs	Accuracy 0.60 AVG	Helpfulness 0.60 AVG	Relevance 0.80 AVG	Latency 1.522 P50
삼성전자가 만든 생성형 AI의 ...	삼성전자가 만든 생성형 AI의 이름은 테디노트 입니...	SPRi AI Brief 2023-12월호 10 삼성전자, 자차	n	n	y	1.94s
코히어의 데이터 출처 탐색기에 ...	코히어의 데이터 출처 탐색기는 AI 모델 훈련에 사...	SPRi AI Brief 2023-12월호 8 코히어, 데이터	y	y	y	1.29s
미국 바이든 대통령이 안전하고...	2023년 10월 30일 미국 바이든 대통령이 행정명...	보호 △노동자 지원 △혁신과 경쟁 촉진 △국제협력	y	y	y	1.52s
구글이 테디노트에게 20억달러...	사실이 아닙니다. 구글은 엔스룩에 최대 20억 달...	E 구글, 엔스룩에 최대 20억 달러 투자 합의 및 I	n	n	n	1.57s
삼성전자가 만든 생성형 AI의 ...	삼성전자가 만든 생성형 AI의 이름은 삼성 가우스 ...	SPRi AI Brief 2023-12월호 10 삼성전자, 자차	y	y	y	1.15s

Inputs	Reference Outputs	Outputs	Accuracy 0.75 AVG	Helpfulness 0.75 AVG	Relevance 1.00 AVG	Latency 105.96 P50
삼성전자가 만든 생성형 AI의 ...	삼성전자가 만든 생성형 AI의 이름은 테디노트 입니...	SPRi AI Brief 2023-12월호 10 삼성전자, 자차	n	n	y	54.60s
코히어의 데이터 출처 탐색기에 ...	코히어의 데이터 출처 탐색기는 AI 모델 훈련에 사...	SPRi AI Brief 2023-12월호 8 코히어, 데이터	y	**y**	y	185.63s
미국 바이든 대통령이 안전하고...	2023년 10월 30일 미국 바이든 대통령이 행정명...	보호 △노동자 지원 △혁신과 경쟁 촉진 △국제협력	y	y	y	103.40s
구글이 테디노트에게 20억달러...	사실이 아닙니다. 구글은 엔스룩에 최대 20억 달...	E 구글, 엔스룩에 최대 20억 달러 투자 합의 및 I	**n**	y	**y**	105.96s
삼성전자가 만든 생성형 AI의 ...	삼성전자가 만든 생성형 AI의 이름은 삼성 가우스 ...	SPRi AI Brief 2023-12월호 10 삼성전자, 자차	y	y	y	108.69s

차이점

gemini-2.5-flash 모델 사용

로컬 LLM 모델 사용

✓ ② 사용자 정의 점수 Evaluator (labeled_score_string)

• 점수를 반환하는 평가자 생성 예시

- **normalize_by** → 점수 정규화 가능
- **변환된 점수** = 0 ~ 1 사이의 값으로 **정규화**
- 코드 예시 속 **accuracy** = 사용자가 임의로 정의한 기준
 - L → **적합한 Prompt** 를 **정의** 하여 **사용** 가능

✓ ↳ eval_labeled_score.py

• 코드 내용

```

# =====
# eval_labeled_score.py (복붙!)
# 점수 기반 평가
# =====

from dotenv import load_dotenv
load_dotenv()

import os
from myrag5 import PDFRAG
from langchain_ollama import ChatOllama          # Ollama 사용
from langsmith.evaluation import evaluate, LangChainStringEvaluator
import warnings
warnings.filterwarnings("ignore")
os.environ["TOKENIZERS_PARALLELISM"] = "false"      # Tokenizer Para

print("🚀 Labeled Score 평가 시작...\n")

# =====
# 파일 경로 & LLM 설정
# =====

script_dir = os.path.dirname(os.path.abspath(__file__))
pdf_path = os.path.join(script_dir, "data", "SPRI_AI_Brief_2023년12월호_F.p

if not os.path.exists(pdf_path):
    print(f"❌ 파일 없음: {pdf_path}")
    exit(1)

print(f"✅ PDF 확인: {pdf_path}\n")

# LLM 생성

llm = ChatOllama(
    model="qwen2.5-coder:7b-instruct",
    temperature=0
)

print(f"✅ 로컬 LLM 생성 완료: Qwen2.5-Coder-7B \n")

# =====
# RAG 시스템 생성
# =====

rag = PDFRAG(pdf_path, llm, chunk_size=300, chunk_overlap=50)

```

```

retriever = rag.create_retriever(k=10, search_type="similarity")

# Context 포함 체인
chain_with_context = rag.create_chain_with_context(retriever)

print("\n" + "="*50)
print("🚩 테스트 실행...")
print("="*50 + "\n")

# 테스트
test_question = "삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?"

try:
    test_result = chain_with_context.invoke(test_question)
    print(f"질문: {test_question}")
    print(f"Context: {test_result['context'][:200]}...")
    print(f"답변: {test_result['answer']}\n")
except Exception as e:
    print(f"❌ 테스트 실패: {e}\n")
    import traceback
    traceback.print_exc()
    exit(1)

# =====
# Context + Answer 반환 함수
# =====

def context_answer_rag_answer(inputs: dict):
    """
    Context와 Answer를 함께 반환
    """
    result = chain_with_context.invoke(inputs["question"])
    return {
        "context": result["context"],
        "answer": result["answer"],
        "query": inputs["question"],
    }

# =====
# Labeled Score 평가자 생성
# =====

print("="*50)
print("🇹🇷 Labeled Score 평가자 생성...")
print("="*50 + "\n")

```

```
# 평가용 LLM
```

```
eval_llm = ChatOllama(  
    model="qwen2.5-coder:7b-instruct",  
    temperature=0  
)
```

```
# 1. Accuracy Score (1-10 점수)
```

```
accuracy_score_evaluator = LangChainStringEvaluator(  
    "labeled_score_string",  
    config={  
        "criteria": {  
            "accuracy": (  
                "How accurate is this prediction compared to the reference  
                "on a scale of 1-10? "  
                "Rate factual correctness only."  
            )  
        },  
        "normalize_by": 10, # 0-1 사이로 정규화  
        "llm": llm,  
    },  
    prepare_data=lambda run, example: {  
        "prediction": run.outputs["answer"],  
        "reference": example.outputs["answer"], # Ground Truth  
        "input": example.inputs["question"],  
    },  
)
```

```
# 2. Completeness Score (1-10 점수)
```

```
completeness_score_evaluator = LangChainStringEvaluator(  
    "labeled_score_string",  
    config={  
        "criteria": {  
            "completeness": (  
                "How complete is this prediction compared to the reference  
                "on a scale of 1-10? "  
                "Does it cover all important information?"  
            )  
        },  
        "normalize_by": 10,  
        "llm": llm,  
    },  
    prepare_data=lambda run, example: {  
        "prediction": run.outputs["answer"],  
        "reference": example.outputs["answer"], # Ground Truth  
        "input": example.inputs["question"],  
    },  
)
```

```

)

# 3. Context Relevance Score (1-10 점수)
context_relevance_score_evaluator = LangChainStringEvaluator(
    "labeled_score_string",
    config={
        "criteria": {
            "context_relevance": (
                "How well does this prediction use the provided context "
                "on a scale of 1-10? "
                "Rate how accurately it references the context."
            )
        },
        "normalize_by": 10,
        "llm": eval_llm,  # 평가용 llm으로 수정
    },
    prepare_data=lambda run, example: {
        "prediction": run.outputs["answer"],
        "reference": run.outputs["context"], # Context
        "input": example.inputs["question"],
    },
)

```

```

print("✅ 평가자 생성 완료\n")

```

```

# =====
# 평가 실행
# =====

```

```

print("="*50)
print("🇩🇪 평가 실행...")
print("="*50 + "\n")

```

```

try:
    experiment_results = evaluate(
        context_answer_rag_answer,
        data="RAG_EVAL_DATASET",
        evaluators=[
            accuracy_score_evaluator,
            completeness_score_evaluator,
            context_relevance_score_evaluator,
        ],
        experiment_prefix="RAG_EVAL_LABELED_SCORE_LOCAL",
        metadata={
            "variant": "Labeled Score (Accuracy + Completeness + Context F",
            "k": 10,
            "chunk_size": 300,

```

```

        "llm": "qwen2.5:14b-instruct",
        "eval_llm": "qwen2.5:14b-instruct",
        "scoring": "1-10 scale, normalized to 0-1",
    },
    max_concurrency=1,
)

print("\n" + "="*50)
print("✅ 평가 완료!")
print("="*50)
print(f"\n결과: {experiment_results}\n")

except Exception as e:
    print(f"\n❌ 에러 발생: {e}\n")
    import traceback
    traceback.print_exc()

```

• Local LLM으로 시도

🚀 Labeled Score 평가 시작...

✅ PDF 확인: [/Users/jay/Projects/20250727-langchain-note/15 Evaluations/data/SPRI](/Users/jay/Projects/20250727-langchain-note/15_Evaluations/data/SPRI)

✅ 로컬 LLM 생성 완료: Qwen2.5-Coder-7B

✅ PDF 로드 완료: 23 페이지

✅ 청크 분할 완료: 119 청크 (크기=300, 오버랩=50)

✅ 임베딩 모델 로드 완료: BAAI/bge-m3

✅ 벡터스토어 생성 완료: FAISS

✅ 검색기 생성 완료 (k=10, search_type=similarity)

✅ Context 포함 RAG 체인 생성 완료

=====

🔧 테스트 실행...

=====

질문: 삼성전자가 자체 개발한 생성형 AI의 이름은 무엇인가요?

Context: SPRI AI Brief |

2023-12월호

10

삼성전자, 자체 개발 생성 AI ‘삼성 가우스’ 공개

n 삼성전자가 온디바이스에서 작동 가능하며 언어, 코드, 이미지의 3개 모델로 구성된 자체 개발 생성 AI 모델 ‘삼성 가우스’를 공개

n 삼성전자는 삼성 가우스를 다양한 제품에 단계적으로 탑재할 계획으로, 온디바이스 작동이 가능한 삼성 가우스는...

답변: 삼성전자가 자체 개발한 생성형 AI의 이름은 '삼성 가우스'입니다.

Labeled Score 평가자 생성...

✅ 평가자 생성 완료

평가 실행...











View the evaluation results for experiment: 'RAG_EVAL_LABELED_SCORE_LOCAL-c81ff062'
<https://smith.langchain.com/o/2c3342d3-1170-4ffa-86fd-f621199e0b9c/datasets/420dc>

5it [13:15, 159.12s/it]

✅ 평가 완료!

결과: <ExperimentResults RAG_EVAL_LABELED_SCORE_LOCAL-c81ff062>

• LangSmith로 결과 확인하기

RAG_EVAL_LABELED_SCORE_LOCAL-c81ff062   202507.../0d488f						
Inputs	Reference Outputs	Outputs	Score_strin...	Score_strin...	Score_strin...	Completed
미국 바이든 대통령이 안전하고... 	2023년 10월 30일 미국 바이든 대통령이 행정명...	보호 △노동자 지원 △혁신과 경쟁 촉진 △국제협력	0.58 AVG 	0.50 AVG 	0.78 AVG 	COMPLETED 1
삼성전자가 만든 생성형 AI의 ... 	삼성전자가 만든 생성형 AI의 이름은 삼성 가우스...	SPRI AI Brief 2023-12월호 10 삼성전자, 자...	1.00	1.00	1.00	COMPLETED 1
코히어의 데이터 출처 탐색기에 ... 	코히어의 데이터 출처 탐색기는 AI 모델 훈련에 사...	SPRI AI Brief 2023-12월호 8 코히어, 데이터	0.70	0.30	0.80	COMPLETED 1
삼성전자가 만든 생성형 AI의 ... 	삼성전자가 만든 생성형 AI의 이름은 테디노트 입...	SPRI AI Brief 2023-12월호 10 삼성전자, 자...	0.10	0.10	1.00	COMPLETED 1
구글이 테디노트에게 20억달러... 	사실이 아닙니다. 구글은 앤스로픽에 최대 20억 달	£ 구글, 앤스로픽에 최대 20억 달러 투자 합의 및 ...	0.10	0.10	0.10	COMPLETED 1

• next: 06. 임베딩 기반 평가 (embedding_distance)