

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

CH15. 평가 (Evaluations)

- LLM** (Large Language Model) 평가: 인공지능 언어 모델의 성능, 정확성, 일관성 및 기타 중요한 측면을 측정하고 분석하는 과정
 - 모델의 개선, 비교, 선택 및 응용 프로그램에 적합한 모델 결정에 필수적인 단계
-

- 평가 방법**
 - 자동화된 메트릭**: BLEU, ROUGE, METEOR, SemScore 등의 지표 사용
 - 인간 평가**: 전문가 or 크라우드 소싱 통한 직접적 평가 → 수행
 - 작업 기반 평가**: 특정 작업에서의 성능을 측정함
 - LLM-as-judge**: 다른 LLM을 평가자로 사용하는 방법
-

- LangChain에서의 Evaluation**
 - LangChain은 LLM의 애플리케이션의 평가를 위한 다양한 도구와 프레임워크를 제공함
 - 모듈화된 평가 컴포넌트**: 다양한 평가방법 쉽게 구현 및 조합 가능
 - Chain 평가**: 전체 LLM 애플리케이션 파이프라인 평가
 - 데이터셋 기반 평가**: 사용자 정의 데이터셋 사용 → 모델 평가
 - 평가 지표**: 정확성, 일관성, 관련성 등 다양한 지표 제공
-

- LLM-as-judge**
 - LLM-as-judge: 다른 LLM의 출력을 평가하기 위해 LLM을 사용하는 혁신적인 접근 방식
 - 자동화**: 인간의 개입 없이 대규모 평가 수행 가능
 - 일관성**: 평가 기준 → 일관되게 적용할 수 있음
 - 유연성**: 다양한 평가 기준, 상황에 적응 가능
 - 비용 효율성**: 인간 평가자에 비해 비용이 적게 들 수 있음

*

- LLM-as-judge의 작동 방식**
 - 입력 제공**: 평가할 LLM의 출력 • 평가 기준 제공
 - 분석**: 평가자 LLM이 제공된 출력을 분석
 - 평가**: 정의된 기준에 따라 점수 or 피드백 생성
 - 결과 집계**: 여러 평가 결과 종합 → 최종 평가 도출
-


- 장단점**
 - 장점**
 - 대규모 평가 기능
 - 빠른 피드백 루프
 - 다양한 평가 기준 적용 가능

- **단점**
 - 평가자 LLM의 편향 가능성
 - 복잡 or 미묘한 평가에 한계가 있을 수 있음
 - 평가자 LLM의 성능에 의존적

- **평가의 중요성**

- **모델 개선**: 약점 식별 → 개선 방향 제시함
- **신뢰성 확보**: 모델의 성능 • 한계 이해 → 도움을 줌
- **적합한 모델 선택**: 특정 작업 or 도메인에 가장 적합한 모델을 선택 가능
- **윤리적 고려사항**: 편향, 공정성 등의 윤리적 측면 평가 가능

1. 합성 테스트 데이터셋 생성 (RAGAS)

- 잦은 에러 발생
 - → Jupyter Notebook의 tqdm 진행바
 - → ipywidgets + ContextVar 충돌
 - → 멀티스레딩 환경 문제
-  **Python 스크립트**로 실행하기

① 01_Test-Dataset-Generator-RAGAS_1.py

- 코드

```
# 01_Test-Dataset-Generator-RAGAS_1.py

"""
llama3.2:3b 교체 및 간단 테스트용
소요 시간: 5~8분
"""

import os
import warnings
warnings.filterwarnings("ignore")
os.environ["TOKENIZERS_PARALLELISM"] = "false"

from dotenv import load_dotenv
load_dotenv()

print("="*60)
print("llama3.2:3b 모델 RAGAS 간단 테스트")
print("="*60)
print()

# 패키지 임포트
from ragas.testset.generator import TestsetGenerator
from ragas.testset.evaluations import simple
from ragas.llms import LangchainLLMWrapper
from ragas.embeddings import LangchainEmbeddingsWrapper
from ragas.testset.extractor import KeyphraseExtractor
from ragas.testset.docstore import InMemoryDocumentStore

from langchain_ollama import ChatOllama
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.document_loaders import PDFPlumberLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

# =====
# 1. LLM 설정
# =====
print("1. LLM 설정 중...")
```

```

generator_llm = ChatOllama(
    model="llama3.2:3b",          # 자연어 모델로 교체
    temperature=0.7,
    num_predict=512,
)

# 간단 테스트
try:
    test = generator_llm.invoke("Say hello")
    print(f"    ✅ LLM 작동: {test.content[:30]}...")
except Exception as e:
    print(f"    ❌ 에러: {e}")
    print("    💡 확인: ollama list")
    print("    💡 다운로드: ollama pull llama3.2:3b")
    exit(1)

print()

# =====
# 2. Embeddings
# =====
print("2. Embeddings 설정 중...")
embeddings = HuggingFaceEmbeddings(
    model_name="BAAI/bge-small-en-v1.5",
    model_kwargs={'device': 'cpu'},
)
print("    ✅ 완료")
print()

# =====
# 3. 문서 로드 (3페이지지만!)
# =====
print("3. 문서 로드 중...")
loader = PDFPlumberLoader("../data/SPRI AI Brief 2023년12월호_F.pdf")
docs = loader.load()[3:6]          # 3페이지만

for doc in docs:
    doc.metadata["filename"] = doc.metadata["source"]

print(f"    ✅ {len(docs)}페이지 로드")
print()

# =====
# 4. DocumentStore
# =====
print("4. DocumentStore 초기화 중...")

splitter = RecursiveCharacterTextSplitter(
    chunk_size=300,          # 사이즈 감소
    chunk_overlap=30        # 오버랩 감소
)

langchain_llm = LangchainLLMWrapper(generator_llm)
keyphrase_extractor = KeyphraseExtractor(llm=langchain_llm)
ragas_embeddings = LangchainEmbeddingsWrapper(embeddings)

docstore = InMemoryDocumentStore(
    splitter=splitter,
    embeddings=ragas_embeddings,
    extractor=keyphrase_extractor,
)

print("    ✅ 완료")
print()

# =====
# 5. Generator
# =====
print("5. Generator 생성 중...")

generator = TestsetGenerator.from_langchain(

```

```

        generator_llm,
        generator_llm,
        ragas_embeddings,
        docstore=docstore,
    )

print("    ✅ 완료")
print()

# =====
# 6. 분포
# =====
distributions = {
    simple: 1.0, # 100% 간단한 질문
}

# =====
# 7. 생성 (1개만!)
# =====
print("="*60)
print("🧠 테스트셋 생성 시작")
print("="*60)
print("예상 시간: 5-8분")
print("멈춰 보여도 정상입니다!")
print()

try:
    testset = generator.generate_with_langchain_docs(
        documents=docs,
        test_size=1, # 1개만!
        distributions=distributions,
        with_debugging_logs=False,
        raise_exceptions=True,
    )

    # =====
    # 8. 결과
    # =====
    test_df = testset.to_pandas()

    if len(test_df) > 0:
        print()
        print("="*60)
        print("✅✅✅ 성공! ✅✅✅ ")
        print("="*60)
        print()

        for idx, row in test_df.iterrows():
            print(f"질문: {row['question']}")
            print(f"답변: {row['ground_truth'][:80]}...")
            print()

        test_df.to_csv("data/.csv", index=False)
        # test_df.to_csv("data/ragas_synthetic_dataset_1.csv", index=False, encoding='utf-8-sig')
        # 혹은 데이터 경로: "../data/ragas_synthetic_dataset_1.csv"
        print("✅ 저장: data/jay_success.csv")

    else:
        print("❌ 질문 생성 실패")

except Exception as e:
    print()
    print("❌ 에러:")
    print(f"    {e}")
    print()
    print("💡 나에게 에러 메시지 보내줘!")

print()
print("="*60)
print("✅ 테스트셋 생성 완료")
print("="*60)

```

- 결과

=====
llama3.2:3b 모델 RAGAS 간단 테스트
=====

1. LLM 설정 중...

✅ LLM 작동: Hello! It's nice to meet you. ...

2. Embeddings 설정 중...

완료

3. 문서 로드 중...

3페이지 로드

4. DocumentStore 초기화 중...

완료

5. Generator 생성 중...

완료

 테스트셋 생성 시작

예상 시간: 5-8분

멈춰 보여도 정상입니다!

[illegible]

```
# Generating: 0%|
```

✓✓✓ 성공! ✓✓✓

질문: context: "n \uc8fc\uc694 7\uac1c\uad6d(G7)*\uc740 2023\ub144 10\uc6d4 30\uc77c \u2018\ud788\ub85c\uc2dc\ub9c8
답변: The answer to given question is not present in context...

✖ 에러:

```
Cannot save file into a non-existent directory: 'data'
```

💡 나에게 에러 메시지 보내줘!

✅ 테스트셋 생성 완료

- `../15 Evaluations/data/ragas_synthetic dataset 1.csv` 으로 저장 실패

② 01_Test-Dataset-Generator-RAGAS_2.py

- 코드

```
# 01_Test-Dataset-Generator-RAGAS_2.py

"""
RAGAS 테스트셋 생성 - 빠른 테스트용
"""

import os
import warnings
warnings.filterwarnings("ignore")
os.environ["TOKENIZERS_PARALLELISM"] = "false"

from dotenv import load_dotenv
```

```

load_dotenv()

print("="*60)
print("llama3.2:3b 모델 RAGAS 간단 테스트")
print("="*60)
print()

print("\n RAGAS 초기화 중...\n")

from ragas.testset.generator import TestsetGenerator
from ragas.testset.evolutions import simple, reasoning
from ragas.llms import LangchainLLMWrapper
from ragas.embeddings import LangchainEmbeddingsWrapper
from ragas.testset.extractor import KeyphraseExtractor
from ragas.testset.docstore import InMemoryDocumentStore

from langchain_ollama import ChatOllama
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.document_loaders import PDFPlumberLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

# 1. LLM 설정
print("1. LLM 설정 중...")

generator_llm = ChatOllama(
    model="llama3.2:3b",          # 자연어 모델로 교체
    temperature=0.7,
    num_predict=512,
)

critic_llm = ChatOllama(
    model="llama3.2:3b",
    temperature=0.1,
    num_predict=512,
)

# 간단 테스트
try:
    test = generator_llm.invoke("Say hello")
    print(f"    ✅ LLM 작동: {test.content[:30]}...")
except Exception as e:
    print(f"    ❌ 에러: {e}")
    print("    💡 확인: ollama list")
    print("    💡 다운로드: ollama pull llama3.2:3b")
    exit(1)

print("    ✅ Ollama LLM 설정 완료")
print()

# 2. Embeddings 설정
print("2. Embeddings 설정 중...")

embeddings = HuggingFaceEmbeddings(
    model_name="BAAI/bge-small-en-v1.5",
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
)

print("    ✅ 완료")
print()

# 3. 문서 로드
print("3. 문서 로드 중...")
loader = PDFPlumberLoader("data/SPRI_AI_Brief_2023년12월호_F.pdf")
docs = loader.load()
docs = docs[3:8]          # 5페이지만!

# metadata 설정
for doc in docs:

```

```

doc.metadata["filename"] = doc.metadata["source"]

print(f"    ✅ {len(docs)}페이지 로드")
print()

# 4. DocumentStore 초기화
print("4. DocumentStore 초기화 중...")

splitter = RecursiveCharacterTextSplitter(
    #chunk_size=800,          # 조금 줄임
    #chunk_size=400,          # 더 작게 조절
    #chunk_overlap=50
    chunk_size=300,          # 사이즈 감소
    chunk_overlap=30         # 오버랩 감소
)

langchain_llm = LangchainLLMWrapper(generator_llm)
keyphrase_extractor = KeyphraseExtractor(llm=langchain_llm)
ragas_embeddings = LangchainEmbeddingsWrapper(embeddings)

docstore = InMemoryDocumentStore(
    splitter=splitter,
    embeddings=ragas_embeddings,
    extractor=keyphrase_extractor,
)

print("    ✅ 완료")
print()

# 5. TestsetGenerator 생성
print("5. Generator 생성 중...")

generator = TestsetGenerator.from_langchain(
    generator_llm,
    critic_llm,
    ragas_embeddings,
    docstore=docstore,
)

print("    ✅ 완료")
print()

# 6. 분포 (간단하게!)
distributions = {
    simple: 0.7,      # 70%
    reasoning: 0.3,   # 30%
}

# 7. 생성 (2개만!)
print("="*60)
print("🔄 테스트셋 생성 시작")
print("="*60)
print("🔄 테스트셋 생성 중... (시간이 걸릴 수 있습니다)")
print("    - 분포: 간단(70%), 추론(30%)\n")
print("🔄 멈춰 보여도 정상입니다! 🔄 ")
print("="*60)
print()

try:
    testset = generator.generate_with_langchain_docs(
        documents=docs,
        test_size=2,          # 2개만
        distributions=distributions,
        with_debugging_logs=False,      # 로그 비활성화
        raise_exceptions=False,         # 에러 무시
    )

# 8. 결과 저장

```

```

test_df = testset.to_pandas()
test_df.to_csv("data/ragas_synthetic_dataset_2.csv", index=False, encoding='utf-8-sig')

if len(test_df) > 0:
    print()
    print("="*60)
    print("✅✅✅ 성공! ✅✅✅")
    print("="*60)
    print()

    for idx, row in test_df.iterrows():
        print(f"질문: {row['question']}")
        print(f"답변: {row['ground_truth'][:80]}...")
        print()

    print("✅ 저장: data/ragas_synthetic_dataset_2.csv")

else:
    print("❌ 질문 생성 실패")

print()
print("="*60)
print(f"\n✅ 테스트셋 생성 완료!")
print(f" - 생성된 질문 수: {len(test_df)}")
print(f" - 저장 위치: ../15 Evaluations/data/ragas_synthetic_dataset.csv\n")
print(test_df)

# 9. 결과 미리보기
print("📄 생성된 질문 미리보기:")
print("="*80)
for idx, row in test_df.iterrows():
    print(f"\n질문 {idx+1}:")
    print(f" {row['question']}")
    print(f" Ground Truth: {row['ground_truth'][:100]}...")

except Exception as e:
    print(f"❌ 에러 발생: {e}")
    print("\n💡 해결 방법:")
    print(" 1. Ollama가 실행 중인지 확인")
    print(" 2. 모델 다운로드: ollama pull llama3.2:3b")
    print(" 3. langchain-ollama 버전 확인: pip list | grep langchain-ollama")
    print("    → 0.1.3이어야 함!")

print("\n✅ 정상적으로 테스트셋 생성 완료!")

```

- 결과

- try 1

 RAGAS 초기화 중...

- ✅ Ollama LLM 설정 완료
- ✅ Embeddings 설정 완료
- ✅ 문서 로드 완료: 5페이지

✅ TestsetGenerator 생성 완료

 시작... (10분 예상)

테스트셋 생성 중... (시간이 걸릴 수 있습니다)
- 분포: 간단(70%), 추론(30%)

중간 과정

```
# embedding nodes: 37% | 17/46 [00:33]
```

```
# embedding nodes: 87%|███████████████████████████████████████████| 40/46 [06:56]
```

[illegible]

```
# Generating: 0% | 0/
```

```
# Generating: 50% | 1/2 [06:24
```

```
Generating: 50%|███████████| 1/2 [06:24<06
```


- =====

✅ 테스트셋 생성 완료!

- 생성된 질문 수: 2

- 저장 위치: ../15 Evaluations/data/ragas_synthetic_dataset.csv

```
              question ... episode_done
0 Here is a question that can be fully answered ...    True
1 The goal is to create a rewritten question tha...    True
```

[2 rows x 6 columns]

📊 생성된 질문 미리보기:

질문 1:

Here is a question that can be fully answered from the given context:

"What does 'FTC' stand for in the provided text?"

This question can be answered by referring to the key phrase "KEY Contents\n\n \ubb8\ud6d FTC..." in the context.
Ground Truth: FTC...

질문 2:

The goal is to create a rewritten question that conveys the same meaning as "What type of AI does the Federal

Here's a revised version:

"What kind of AI system does the FTC use?"

This rewritten question achieves the same intent as the original but in a more concise and indirect manner, using the context.
Ground Truth: The Federal Trade Commission uses artificial intelligence primarily for regulating privacy, trade

✅ 정상적으로 테스트셋 생성 완료!

2. 주피터 노트북으로 시도 - try_1

1) 합성 테스트 데이터셋 생성

• 왜 합성 테스트 데이터 (Synthetic Test Dataset) 인가?

- RAG (검색 증강 생성) 증강 파이프라인의 성능을 평가하는 것은 매우 중요
- 어려움
 - 문서에서 수백 개의 QA (질문-문맥-응답) 샘플을 수동으로 생성하는 것은 시간과 노동력이 많이 소요
 - 사람이 만든 질문 = 철저한 평가에 필요한 복잡성 수준에 도달하기 어려움 = 궁극적으로 평가의 품질에 영향을 미칠 수 있음
- 합성 데이터 생성을 사용 → 데이터 집계 프로세스에서 개발자의 시간을 90% 까지 감소 가능
- 참고: [RAGAS](#)

• 사전에 VS Code 터미널에 설치할 것

```
pip install -qU ragas
```

• 구버전으로 설치할 것!

```
pip install ragas==0.1.21
```

```
# 의존성 확인
```

```
pip list | grep -E "ragas|langchain"
```

```
# 예상 출력:
```

```
# ragas                                0.1.21
```

```
# ...
```

- 오류 발생 → 커널 및 패키지 충돌
- 트러블 슈팅 문서 참고
 - [RAGAS Synthetic Dataset Generation Version Conflicts Troubleshooting](#)

2) 환경설정

① 기본설정

```
import os
import warnings
warnings.filterwarnings("ignore")
os.environ["TOKENIZERS_PARALLELISM"] = "false"
os.environ["TQDM_DISABLE"] = "1" # 진행바 비활성화

from dotenv import load_dotenv
load_dotenv()

print("✅ 환경 설정 완료")
```

- ✅ 환경 설정 완료

② LangSmith 설정

```
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.")
```

- 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.
```

③ 패키지 임포트

```
from ragas.testset.generator import TestsetGenerator
from ragas.testset.evolutions import simple, reasoning, conditional, multi_context
from ragas.llms import LangchainLLMWrapper
from ragas.embeddings import LangchainEmbeddingsWrapper
from ragas.testset.extractor import KeyphraseExtractor
from ragas.testset.docstore import InMemoryDocumentStore

from langchain_ollama import ChatOllama
from langchain_huggingface import HuggingFaceEmbeddings
```

```
from langchain_community.document_loaders import PDFPlumberLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

print("✅ 패키지 임포트 완료")
```

- ✅ 패키지 임포트 완료 - (1.8s)

3) LLM 호출하기

① 데이터셋 생성기

```
# 데이터셋 생성기
generator_llm = ChatOllama(
    model="llama3.2:3b",          # 자연어 모델로 교체
    temperature=0.7,
    num_predict=512,
)

# 테스트 호출
test_response = generator_llm.invoke("Hello")
print(f"✅ 데이터셋 생성기 LLM 작동 확인: {test_response.content[:50]}...")
```

- ✅ 데이터셋 생성기 LLM 작동 확인: How can I assist you today?... - (4.7s)

② 데이터셋 비평기

```
# 데이터셋 비평기
critic_llm = ChatOllama(
    model="llama3.2:3b",
    temperature=0.1,
    num_predict=512,
)

# 테스트 호출
test_response2 = critic_llm.invoke("안녕?")
print(f"✅ 데이터셋 생성기 LLM 작동 확인: {test_response2.content[:50]}...")
```

- ✅ 데이터셋 생성기 LLM 작동 확인: 안녕하세요! (Hello!) How can I help you today?... - (2.3s)

4) 임베딩 설정하기

```
# 임베딩 생성
embeddings = HuggingFaceEmbeddings(
    model_name="BAAI/bge-small-en-v1.5",
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
)

print("✅ 임베딩 생성 완료")

# 테스트 임베딩
test_embed = embeddings.embed_query("테스트")
print(f"✅ 임베딩 작동 확인: 차원={len(test_embed)}")
```

- ✅ 임베딩 생성 완료 - (9.1s)
- ✅ 임베딩 작동 확인: 차원=384

5) 문서 로드 & 전처리

① 문서 로드하기

- 실습에 활용할 문서
 - 소프트웨어정책연구소 (SPRi) - 2023년 12월호

- 저자: 유재홍(AI정책연구실 책임연구원), 이지수(AI정책연구실 위촉연구원)
- 참고: [링크](#)
- 파일명: [SPRI_AI_Brief_2023년12월호_F.pdf](#)

```
loader = PDFPlumberLoader("../15_Evaluations/data/SPRI_AI_Brief_2023년12월호_F.pdf")

docs = loader.load()

# 목차, 끝 페이지 제외
docs = docs[3:-1]                # 교재 사이트와 똑같이 설정

print(f"✅ 문서 로드 완료: {len(docs)}페이지")
```

- ✅ 문서 로드 완료: 19페이지 - (3.3s)

```
print(type(docs))                # <class 'list'>
```

② 메타 데이터 설정하기

- **metadata** 확인하기
 - 각 문서 객체에는 **metadata** → 액세스할 수 있는 문서에 대한 추가 정보를 저장하는 데 사용할 수 있는 (메타데이터 사전)이 포함되어 있음
 - 메타데이터 사전에 **filename** 이라는 키가 포함되어 여부 확인하기
 - 이 키는 **Test datasets** 생성 프로세스에서 활용될 것
 - 메타데이터의 **filename** 속성 = 동일한 문서에 속한 청크를 식별하는 데 사용됨

```
# metadata 설정
for doc in docs:
    doc.metadata["filename"] = doc.metadata["source"]

print(f"✅ 첫 문서 미리보기: {docs[0].page_content[:50]} ...")
```

- ✅ 첫 문서 미리보기: 1. 정책/법제 2. 기업/산업 3. 기술/연구 4. 인력/교육
- 미국, 안전하고 신뢰할 수 ...

6) DocumentStore 초기화

- **docsstore** 구성요소 및 **docstore** 초기화
 - **DocumentStore** 초기화
 - 사용자 정의 LLM 과 임베딩 사용

```
# 텍스트 분할기 설정
splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=100
)

# LangchainLLMWrapper로 감싸 Ragas와 호환되도록 함
langchain_llm = LangchainLLMWrapper(generator_llm)

# 주요 구문 추출기 초기화 (위에서 정의한 LLM 사용)
keyphrase_extractor = KeyphraseExtractor(llm=langchain_llm)

# ragas_embeddings 생성
ragas_embeddings = LangchainEmbeddingsWrapper(embeddings)

# InMemoryDocumentStore 초기화 (문서를 메모리에 저장하고 관리하는 저장소)
docstore = InMemoryDocumentStore(
    splitter=splitter,
    embeddings=ragas_embeddings,
    extractor=keyphrase_extractor,
)

print("✅ DocumentStore 초기화 완료")
```

- ✅ DocumentStore 초기화 완료

7) DataSet 생성하기

① TestSet Generator 생성하기

```
generator = TestsetGenerator.from_langchain(  
    generator_llm,  
    critic_llm ,  
    ragas_embeddings,  
    docstore=docstore,  
)  
  
print("✅ Generator 생성 완료")
```

- ✅ Generator 생성 완료

② 질문 분포 설정하기

- 질문의 유형별 분포
 - simple: 간단한 질문
 - reasoning: 추론이 필요한 질문
 - multi_context: 여러 맥락을 고려해야 하는 질문
 - conditional: 조건부 질문

```
# 질문 유형별 분포 결정  
  
distributions = {  
    simple: 0.4,          # simple: 간단한 질문  
    reasoning: 0.2,       # reasoning: 추론이 필요한 질문  
    multi_context: 0.2,   # multi_context: 여러 맥락을 고려해야 하는 질문  
    conditional: 0.2      # conditional: 조건부 질문  
}  
  
print("✅ 질문 분포 설정 완료")
```


- ✅ 질문 분포 설정 완료

```
print(type(distributions))          # <class 'dict'>
```

③ 데이터셋 생성하기

- 테스트셋 생성
 - documents: 문서 데이터
 - test_size: 생성 할 질문의 수
 - distributions: 질문 유형별 분포
 - with_debugging_logs: 디버깅 로그 출력 여부

```
print("="*60)  
print("🔄 테스트셋 생성 시작")  
print("="*60)  
print("🔄 테스트셋 생성 중... (시간이 걸릴 수 있습니다)")  
print("="*60)  
print("🔄 멈춰 보여도 정상입니다! 🔄 ")  
print("="*60)  
print("🕒 예상 시간: 20-30분")  
print("💡 멈춘 것처럼 보여도 정상입니다!")  
  
import time  
start_time = time.time()  
  
try:  
    testset = generator.generate_with_langchain_docs(  
        documents=docs,  
        test_size=3,          # 더 작게 설정  
        distributions=distributions,  
        with_debugging_logs=False, # 로그 비활성화  
        raise_exceptions=False,  # 에러 무시  
    )
```


-  저장: data/ragas_synthetic_dataset_3.csv

⑤ 결과 확인하기

```
# 미리보기

if len(test_df) > 0:

    print()
    print("="*60)
    print("✅✅✅ 성공! ✅✅✅ ")
    print("="*60)
    print()

    for idx, row in test_df.iterrows():
        print(f"\n질문 {idx+1}: ")
        print(f"        {row['question']}")
        print(f"\n답변 {idx+2}: ")
        print(f"        {row['ground_truth'][:100]}...")
        print()
        print("✅ 저장: data/ragas_synthetic_dataset_3.csv")

else:
    print("❌ 질문 생성 실패")
```

- **생성된 질문보기**

```
=====
✅✅✅ 성공! ✅✅✅
=====
```

질문 1:

Here is a question that can be fully answered from the given context:

"Who developed the model RAG \ud3ec\ud568 \uc9c8\ubb38\uacfc \ub2f5\ubcc0\uc5d0\uc11c\ub294 \ud5c8\uae45, and what is its relation to LLM?"

Note that this question can be answered by reading the context provided.

답변 2:

Who developed the model RAG 포함 질문과 답변에서는 \ud5c8\uae45, and what is its relation to LLM?...

✅ 저장: data/ragas_synthetic_dataset_3.csv

질문 2:

Here is a rewritten version of the question:

"What technology is CCnet 2024 based on?"

This version still conveys the same meaning as the original question, but in a more concise and indirect way.

답변 3:

nan...

✅ 저장: data/ragas_synthetic_dataset_3.csv

질문 3:

Here is a rewritten version of the question that conveys the same meaning in a less direct and shorter manner

"What specific aspects of UL2 training are targeted by CES 2024's AI focus, and how do they align with the OpenMoE framework?"

I made the following changes to achieve this:

- * Abbreviated "OpenMoE framework" to "OpenMoE"
- * Changed "specific aspects of UL2 training objective's configuration" to "specific aspects of UL2 training"
- * Combined "within the framework" into the question itself
- * Simplified the wording and sentence structure

답변 4:

The answer to given question is not present in context...

✅ 저장: data/ragas_synthetic_dataset_3.csv

질문 4:

Here's a rewritten version of the question that conveys the same meaning in a less direct and shorter manner:

"What triggers EU AI regulation for developers?"

Alternatively, you could also ask:

"When must EU AI regulation be applied by developers?"

답변 5:

nan...

✅ 저장: data/ragas_synthetic_dataset_3.csv

3. 주피터 노트북으로 시도 - try_2

```
# =====
# 셀 1: 환경 설정
# =====
import os
import warnings
warnings.filterwarnings("ignore")
os.environ["TOKENIZERS_PARALLELISM"] = "false"
os.environ["TQDM_DISABLE"] = "1" # 진행바 비활성화

from dotenv import load_dotenv
load_dotenv()

print("✅ 환경 설정 완료")
```

- ✅ 환경 설정 완료

```
# =====
# 셀 2: 패키지 임포트 & 확인
# =====

from ragas.testset.generator import TestsetGenerator
from ragas.testset.evolutions import simple, reasoning, conditional
from ragas.llms import LangchainLLMWrapper
from ragas.embeddings import LangchainEmbeddingsWrapper
from ragas.testset.extractor import KeyphraseExtractor
from ragas.testset.docstore import InMemoryDocumentStore

from langchain_ollama import ChatOllama
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.document_loaders import PDFPlumberLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

print("✅ 패키지 임포트 완료")
```

- ✅ 패키지 임포트 완료

```
# =====
# 셀 3: LLM 설정 (확인 가능!)
# =====

generator_llm = ChatOllama(
    model="llama3.2:3b", # 자연어 모델로 교체
    temperature=0.7,
    num_predict=512,
)

# 테스트 호출
test_response = generator_llm.invoke("Hello")
print(f"✅ 데이터셋 생성기 LLM 작동 확인: {test_response.content[:50]}...")
```

- ✅ 데이터셋 생성기 LLM 작동 확인: How can I assist you today?... - (3.9s)

```
# =====
# 셀 4: 임베딩 설정 (확인 가능!)
# =====
```

```

embeddings = HuggingFaceEmbeddings(
    model_name="BAAI/bge-small-en-v1.5",
    model_kwargs={'device': 'cpu'},
)

# 테스트 임베딩
test_embed = embeddings.embed_query("테스트")
print(f"✅ 임베딩 작동 확인: 차원={len(test_embed)}")

```

- ✅ 임베딩 작동 확인: 차원=384 - (3.2s)

```

# =====
# 셀 5: 문서 로드 (확인 가능!)
# =====
loader = PDFPlumberLoader("data/SPRI_AI_Brief_2023년12월호_F.pdf")
docs = loader.load()[3:-1]

for doc in docs:
    doc.metadata["filename"] = doc.metadata["source"]

print(f"✅ 문서 로드 완료: {len(docs)}페이지")
print(f"첫 문서 미리보기: {docs[0].page_content[:100]}...")

```

- ✅ 문서 로드 완료: 19페이지 - (2.8s)

첫 문서 미리보기: 1. 정책/법제 2. 기업/산업 3. 기술/연구 4. 인력/교육
미국, 안전하고 신뢰할 수 있는 AI 개발과 사용에 관한 행정명령 발표
KEY Contents
n 미국 바이든 대통...

```

# =====
# 셀 6: DocumentStore 초기화 (확인 가능!)
# =====
splitter = RecursiveCharacterTextSplitter(
    chunk_size=600,          # 더 작게!
    chunk_overlap=50
)

langchain_llm = LangchainLLMWrapper(generator_llm)
keyphrase_extractor = KeyphraseExtractor(llm=langchain_llm)
ragas_embeddings = LangchainEmbeddingsWrapper(embeddings)

docstore = InMemoryDocumentStore(
    splitter=splitter,
    embeddings=ragas_embeddings,
    extractor=keyphrase_extractor,
)

print("✅ DocumentStore 초기화 완료")

```

- ✅ DocumentStore 초기화 완료

```

# =====
# 셀 7: Generator 생성 (확인 가능!)
# =====
generator = TestsetGenerator.from_langchain(
    generator_llm,
    generator_llm,          # critic도 동일하게
    ragas_embeddings,
    docstore=docstore,
)

print("✅ Generator 생성 완료")

```

```

# =====
# 셀 8: 질문 분포 설정 (multi_context = 불안정)
# =====
distributions = {
    simple: 0.6,             # 60%
    reasoning: 0.2,         # 20%
    conditional: 0.2,       # 20%
}

print("✅ 질문 분포 설정 완료")

```

- ```
=====
셀 9: 테스트셋 생성 (시간 소요!)
=====
print("🔄 테스트셋 생성 시작...")
print("🕒 예상 시간: 20~30분")
print("💡 멈춘 것처럼 보여도 정상입니다!")

import time
start_time = time.time()

try:
 testset = generator.generate_with_langchain_docs(
 documents=docs,
 test_size=3, # 더 작게!
 distributions=distributions,
 with_debugging_logs=False,
 raise_exceptions=False,
)

 elapsed = time.time() - start_time
 print(f"\n✅ 완료! 소요 시간: {elapsed/60:.1f}분")

except Exception as e:
 print(f"❌ 에러: {e}")
 print("💡 Ollama가 실행 중인지 확인하세요!")
```

```
=====
셀 10: 결과 확인 & 저장
=====
test_df2 = testset.to_pandas()
print(f"생성된 질문 수: {len(test_df2)}")
```

- ```
test_df2
```

- | | question | contexts | ground_truth | evolution_type | metadata | episode_done |
|---|--|---------------------------|---|---------------------------|--|--|
| | 누락: 0 (0%)
고유: 4 (100%) | 누락: 0 (0%)
고유: 2 (50%) | 누락: 0 (0%)
고유: 2 (50%) | 누락: 0 (0%)
고유: 3 (75%) | 누락: 0 (0%)
고유: 3 (75%) | 누락: 0 (0%)
False: 2 (50%)
True: 4 (100%) |
| | Here is a question that can... | 25% [구글 딥마인드, 범용 AI 모델... | 75% nan | 50% simple | 50% [{"source": "data/SPRI_AI_Br..."}] | 75% |
| | Here's a question that can ... | 25% [위험 식별과 완화에 필요한 조... | 25% Artificial General Intelligen... | 25% reasoning | 25% [{"source": "data/SPRI_AI_Br..."}] | 25% |
| | Here is a rewritten questi... | 25% | 25% The answer to given questi... | 25% conditional | 25% | |
| | 기타 | 25% | | | | |
| 0 | Here is a question that can be fully a | [구글 딥마인드, 범용 AI 모델의 기능과 | Artificial General Intelligence (AGI) r | simple | [{"source": "data/SPRI_AI_Brief_2023년 | True |
| 1 | Here's a question that can be fully ar | [위험 식별과 완화에 필요한 조치를 포함 | The answer to given question is not | simple | [{"source": "data/SPRI_AI_Brief_2023년 | True |
| 2 | Here is a rewritten question that con | [구글 딥마인드, 범용 AI 모델의 기능과 | nan | reasoning | [{"source": "data/SPRI_AI_Brief_2023년 | True |
| 3 | Here is a rewritten version of the qu | [구글 딥마인드, 범용 AI 모델의 기능과 | nan | conditional | [{"source": "data/SPRI_AI_Brief_2023년 | True |

- `test_df2.head()`

question	contexts	ground_truth	evolution_type	metadata	episode_done
누락: 0 (0%) 고유: 4 (100%)	누락: 0 (0%) 고유: 2 (50%)	누락: 0 (0%) 고유: 2 (50%)	누락: 0 (0%) 고유: 3 (75%)	누락: 0 (0%) 고유: 3 (75%)	누락: 0 (0%) False: 2 (50%) True: 4 (100%)
Here is a question that can... 25%	['구글 딥마인드, 범용 AI 모델의 가능성과...' 25%]	nan 75%	simple 50%	[[{'source': 'data/SPRI_AI_Brief_2023년' 50%}]	75%
Here's a question that can ... 25%	['위험 식별과 완화에 필요한 조치...' 25%]	Artificial General Intelligen... 25%	reasoning 25%	[[{'source': 'data/SPRI_AI_Brief_2023년' 25%}]	25%
Here is a rewritten questio... 25%	['구글 딥마인드, 범용 AI 모델의 가능성과...' 25%]	The answer to given questi... 25%	conditional 25%	25%	
기타 25%					

0	Here is a question that can be fully a	['구글 딥마인드, 범용 AI 모델의 가능성과...' 25%]	Artificial General Intelligence (AGI) re	simple	[[{'source': 'data/SPRI_AI_Brief_2023년' 50%}]	True
1	Here's a question that can be fully ar	['위험 식별과 완화에 필요한 조치...' 25%]	The answer to given question is not	simple	[[{'source': 'data/SPRI_AI_Brief_2023년' 25%}]	True
2	Here is a rewritten question that con	['구글 딥마인드, 범용 AI 모델의 가능성과...' 25%]	nan	reasoning	[[{'source': 'data/SPRI_AI_Brief_2023년' 25%}]	True
3	Here is a rewritten version of the que	['구글 딥마인드, 범용 AI 모델의 가능성과...' 25%]	nan	conditional	[[{'source': 'data/SPRI_AI_Brief_2023년' 25%}]	True

CSV 저장

```
test_df2.to_csv("../15_Evaluations/data/ragas_synthetic_dataset_4.csv", index=False, encoding='utf-8-sig')
print("\n✅ 저장 완료: data/ragas_testset4.csv")
```

- ✅ 저장 완료: data/ragas_testset4.csv

미리보기

```
if len(test_df2) > 0:
    print()
    print("="*60)
    print("✅✅✅ 성공! ✅✅✅")
    print("="*60)
    print()

    for idx, row in test_df2.iterrows():
        print(f"\n질문 {idx+1}: ")
        print(f"    {row['question']}")
        print(f"\n답변 {idx+2}: ")
        print(f"    {row['ground_truth'][:100]}...")
        print()
        print("✅ 저장: data/ragas_synthetic_dataset_4.csv")
else:
    print("❌ 질문 생성 실패")
```

```
=====
✅✅✅ 성공! ✅✅✅
=====
```

질문 1:

Here is a question that can be fully answered from the given context:

"What is Artificial General Intelligence (AGI) and how does it differ from other types of artificial intelligence?"

This question is formed using the keyphrase "Artificial General Intelligence" which appears throughout the provided context.

답변 2:

Artificial General Intelligence (AGI) refers to a type of AI that possesses human-like intelligence ...

✅ 저장: data/ragas_synthetic_dataset_4.csv

질문 2:

Here's a question that can be fully answered from the given context:

"Who is the founder of AI?"

answer: The context does not mention the founder of AI, but it talks about AI in the context of G7 and its applications.

Note: The provided text does not contain information about the founder(s) of AI.

답변 3:

The answer to given question is not present in context...

✅ 저장: data/ragas_synthetic_dataset_4.csv

질문 3:

Here is a rewritten question that conveys the same meaning but in a less direct manner, using abbreviation and rephrasing.

"What's the term for AGI?"

This rewritten question still asks about the concept of Artificial General Intelligence (AGI), but does so in a more indirect way.

답변 4:

nan...

✅ 저장: data/ragas_synthetic_dataset_4.csv

질문 4:

Here is a rewritten version of the question that conveys the same meaning but in a less direct and shorter manner:

"What config. details in OpenMoE define AGI's UL2 objective?"

Alternatively, you could also use:

"How do OpenMoE settings impact AGI's UL2 training?"

Or even more concise:

"Which OpenMoE settings affect AGI's UL2 objective?"

답변 5:

nan...

✅ 저장: data/ragas_synthetic_dataset_4.csv

• next: **02. SQL**
