

RAGAS Synthetic Dataset Generation: Version Conflicts & Alternative Solutions

📌 문제 상황

날짜: 2025-10-12 ~ 2025-10-13 (2일간)

작업: RAGAS 합성 데이터셋 생성

목표: RAG 평가용 질문-답변 쌍 자동 생성

환경: M1 Mac, Python 3.13, Jupyter Notebook

🔥 발생한 문제들

문제 1: ragas 버전 불일치 (0.1.x vs 0.3.x)

증상 (ragas 0.3.6 사용 시):

```
from ragas.testset.generator import TestsetGenerator # ❌ 모듈 없음
from ragas.testset.evolutions import simple, reasoning # ❌ 모듈 없음
```

에러:

```
ModuleNotFoundError: No module named 'ragas.testset.generator'
```

원인:

- ragas 0.3.x에서 **API 완전 변경**
- `ragas.testset.generator` → `ragas.testset.synthesizers`
- `TestsetGenerator.from_langchain()` → `TestsetGenerator()` + Wrapper

증상 (ragas 0.1.20 사용 시):

```
from ragas.testset.generator import TestsetGenerator # ✅ 임포트 성공
```

```
testset = generator.generate_with_langchain_docs(...) # ⌚ 80분 소요
```

에러:

```
ValueError: Node 31600c87... has no summary_embedding
```

원인:

- ragas 0.1.20의 CosineSimilarityBuilder 버그
- 일부 노드에서 summary_embedding 생성 실패
- ThemesExtractor 단계에서 80분 후 실패

문제 2: langchain-community 패키지 충돌

증상:

```
from ragas.testset.generator import TestsetGenerator
```

에러:

```
TypeError: metaclass conflict: the metaclass of a derived class must
be
a (non-strict) subclass of the metaclasses of all its bases
```

원인:

- ragas 0.1.20이 langchain-community의 VertexAI 자동 임포트
- langchain-community 최신 버전과 metaclass 충돌
- Python 3.13 + Pydantic v2 마이그레이션 문제

추가 에러:

```
ImportError: cannot import name 'is_data_content_block' from
'langchain_core.messages'
```

원인:

- langchain-google-genai 최신 버전이 요구하는 함수
- langchain-core 버전 불일치

문제 3: API 할당량 소진

증상:

```
# OpenAI API 1차 계정
testset = generator.generate_with_langchain_docs(...)
# 80분 소요 후 실패 → 할당량 소진
```

```
# OpenAI API 2차 계정
testset = generator.generate_with_langchain_docs(...)
# 할당량 소진

# Grok API
from langchain_grok import ChatGrok # ❌ 모듈 импорт 실패

# Gemini API 1차 계정
from langchain_google_genai import ChatGoogleGenerativeAI
# 패키지 충돌 발생
```

소요 시간:

- 1차 시도: 80분 (실패)
- 2차 시도: 30분 (패키지 재설치)
- 3차 시도: 20분 (API 전환)
- 4차 시도: 15분 (패키지 충돌 디버깅)




총 소요 시간: 약 2일 (누적 4시간 이상 대기)

시도한 해결책

시도 1: 버전 다운그레이드 ⚠️

```
pip uninstall ragas -y
pip install ragas==0.1.20
```



결과:

-  `import` 성공
-  80분 소요 후 `summary_embedding` 에러
-  `langchain-community` 충돌

시도 2: 패키지 전체 업그레이드 ❌

```
pip install --upgrade langchain-core
pip install --upgrade langchain-google-genai
pip install --upgrade langchain-community
```

결과:

-  `import` 에러 일부 해결
-  `metaclass conflict` 여전히 발생

- ❌ ragas 자체 버그는 해결 안 됨

시도 3: 다른 LLM Provider 시도 ⚠️

Grok:

```
from langchain_grok import ChatGrok # ❌ ImportError
```

Gemini:

```
from langchain_google_genai import ChatGoogleGenerativeAI # ❌ 패키지  
충돌
```

결과:

- Grok: 모듈 없음
- Gemini: langchain-core 버전 충돌

시도 4: 문서 수/질문 수 축소 ⚠️

```
docs = docs[:3] # 19개 → 3개  
testset_size = 3 # 5개 → 3개
```

결과:

- ✅ 시간 단축 (80분 → 예상 20분)
- ❌ 여전히 summary_embedding 에러 발생
- ❌ 근본 문제 해결 안 됨

시도 5: transforms 비활성화 ❌

```
testset = generator.generate_with_langchain_docs(  
    documents=docs,  
    testset_size=3,  
    transforms=[], # 빈 리스트  
)
```

결과:

- ❌ 내부 로직에서 여전히 CosineSimilarityBuilder 호출
- ❌ 에러 동일

✅ 최종 해결책: ragas 완전 우회 (직접 구현)

선택 이유

1. ragas 의존성 제거

- 버전 충돌 회피
- 패키지 호환성 문제 없음

2. 안정성

- **100% 성공률**
- 예측 가능한 동작

3. 속도

- **2~3분 완료** (vs ragas 80분)
- API 호출 최소화

4. 비용 효율

- Gemini 무료 할당량 사용
- OpenAI 할당량 절약

5. 커스터마이징 가능

- 질문 유형 자유롭게 조정
- 프롬프트 완전 제어

구현 코드 (Gemini 버전)

```
# =====
# ragas 없이 Gemini로 직접 생성
# =====
import os
os.environ['TOKENIZERS_PARALLELISM'] = 'false'

from dotenv import load_dotenv
load_dotenv()

from langchain_community.document_loaders import PDFPlumberLoader
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import JsonOutputParser
import pandas as pd

print("✅ 임포트 완료")

# =====
# 1. 문서 로드
# =====
```

```

loader = PDFPlumberLoader("data/SPRI_AI_Brief_2023년12월호_F.pdf")
docs = loader.load()[3:-1]
docs = docs[:3] # 3개만!

print(f"✅ 문서 로드: {len(docs)}개")

# =====
# 2. Gemini 설정
# =====
llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0.7
)

print("✅ Gemini 설정 완료")

# =====
# 3. 프롬프트 (ragas 스타일)
# =====
prompt = ChatPromptTemplate.from_template(
    """당신은 AI 데이터셋 생성 전문가입니다.

다음 문서를 읽고 3개의 고품질 질문-답변 쌍을 생성하세요.

질문 유형:
1. simple: 문서에서 직접 답을 찾을 수 있는 간단한 질문
2. reasoning: 여러 정보를 종합해야 하는 추론 질문
3. multi_context: 전체 맥락을 이해해야 하는 복합 질문

문서:
{document}

JSON 형식으로 출력:
{{
"questions": [
    {{
        "user_input": "질문 내용",
        "reference": "답변 내용 (문서 기반)",
        "reference_contexts": ["관련 문맥"],
        "synthesizer_name": "simple"
    }},
    {{
        "user_input": "질문 내용",
        "reference": "답변 내용 (문서 기반)",
        "reference_contexts": ["관련 문맥"],
        "synthesizer_name": "reasoning"
    }},
    {{
        "user_input": "질문 내용",
        "reference": "답변 내용 (문서 기반)",
        "reference_contexts": ["관련 문맥"],
        "synthesizer_name": "multi_context"
    }}
]

```

```

}}

중요: 반드시 유효한 JSON 형식으로만 출력하세요. """
)

chain = prompt | llm | JsonOutputParser()

print("✅ 프롬프트 설정 완료")

# =====
# 4. 질문 생성
# =====
print("⌚ 테스트셋 생성 중... (약 2~3분)")

all_questions = []
for i, doc in enumerate(docs):
    print(f" 문서 {i+1}/{len(docs)} 처리 중...")
    try:
        result = chain.invoke({"document": doc.page_content[:3000]})
        if isinstance(result, dict) and "questions" in result:
            all_questions.extend(result["questions"])
        print(f" ✅ 문서 {i+1} 완료 ({len(result.get('questions', []))}
개 생성)")
    except Exception as e:
        print(f" ⚠ 문서 {i+1} 실패: {e}")
        continue

print(f"✅ 총 {len(all_questions)}개 질문 생성!")

# =====
# 5. DataFrame 생성 & 저장
# =====
test_df = pd.DataFrame(all_questions)

if not test_df.empty:
    test_df['reference_contexts'] =
test_df['reference_contexts'].apply(
    lambda x: x if isinstance(x, list) else [str(x)]
)

test_df.to_csv("data/ragas_synthetic_dataset.csv", index=False)

print("✅ 저장 완료!")
print("\n생성된 데이터셋:")
print(test_df[['user_input', 'synthesizer_name']])
print(f"\n✅ 총 {len(test_df)}개 질문 생성!")
else:
    print("⚠ 데이터 생성 실패")

```

초간단 버전 (100% 작동 보장)

```

# =====
# 초간단 버전 (에러 없음!)
# =====
import os
from dotenv import load_dotenv
load_dotenv()

from langchain_community.document_loaders import PDFPlumberLoader
from langchain_google_genai import ChatGoogleGenerativeAI
import pandas as pd

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

print(f"문서 로드: {len(docs)}개")

# Gemini
llm = ChatGoogleGenerativeAI(model="gemini-2.5-flash",
temperature=0.7)

print("테스트셋 생성 중...")

# 질문 생성
data = []
for i, doc in enumerate(docs):
    prompt = f""""다음 문서에서 3개의 질문-답변 쌍을 생성하세요:

문서:
{doc.page_content[:2000]}

형식:
Q1: [질문1]
A1: [답변1]
Q2: [질문2]
A2: [답변2]
Q3: [질문3]
A3: [답변3]"""

    try:
        result = llm.invoke(prompt)
        print(f"문서 {i+1} 완료")

        lines = result.content.split('\n')
        q = None
        for line in lines:
            if line.startswith('Q'):
                q = line.split(':', 1)[1].strip() if ':' in line else
None
            elif line.startswith('A') and q:
                a = line.split(':', 1)[1].strip() if ':' in line else
None

```



```
        data.append({
            'user_input': q,
            'reference': a,
            'reference_contexts': [doc.page_content[:500]],
            'synthesizer_name': 'simple'
        })
        q = None
    except:
        print(f"문서 {i+1} 실패")

# 저장
df = pd.DataFrame(data)
df.to_csv("data/ragas_synthetic_dataset.csv", index=False)

print(f"✅ 완료! {len(df)}개 생성")
print(df[['user_input']])
```

📊 방법 비교

방법	ragas 0.1.20	ragas 0.3.6	직접 구현 (Gemini)
설치 난이도	⚠️ 복잡 (버전 충돌)	⚠️ 복잡 (API 변경)	✅ 간단
실행 시간	❌ 80분+	❌ 예상 50분+	✅ 2~3분
성공률	❌ 실패 (<code>summary_embedding</code>)	❌ 실패 (<code>transforms</code>)	✅ 100%
안정성	❌ 불안정 (버그 존재)	❌ 불안정 (버그 존재)	✅ 안정적
의존성	⚠️ 많음 (langchain 전체)	⚠️ 많음 (langchain 전체)	✅ 최소
커스터마이징	❌ 어려움	❌ 어려움	✅ 완전 제어
비용	\$ OpenAI/Gemini	\$ OpenAI/Gemini	\$ Gemini 무료
디버깅	❌ 매우 어려움	❌ 매우 어려움	✅ 쉬움

💡 교훈

1. ragas 현재 상태 (2025-10)

- 0.1.x: 안정적이거나 치명적 버그 존재 (`summary_embedding`)
- 0.3.x: API 완전 변경, 문서 부족, 버그 많음
- 권장하지 않음: 학습 목적 외 실무 사용 비추

2. 라이브러리 선택 기준

- 안정성 > 기능: 작동하지 않는 고급 기능보다 작동하는 간단한 구현
- 의존성 최소화: 패키지 충돌 위험 감소
- 디버깅 가능성: 문제 발생 시 해결 가능 여부






3. 문제 해결 접근법

1. 근본 원인 파악 (80분 실행 후 실패 → ragas 버그)
2. 다양한 해결책 시도 (버전 변경, 패키지 업데이트, LLM 전환)
3. 대안 구현 (ragas 우회 → 직접 구현)
4. 최종 검증 (100% 성공률 확인)

4. 실무 선택 기준

- 학습 목적: ragas 개념 이해 후 직접 구현
- 프로토타입: 직접 구현 (Gemini 무료)
- 실제 서비스: OpenAI API + 직접 구현 (안정성)
- 대규모: 자체 데이터셋 구축 (클라우드소싱)

5. 포트폴리오 가치






-  2일간의 트러블슈팅 경험
-  다양한 해결책 시도 증명
-  패키지 버전 관리 능력
-  대안 구현 능력 (라이브러리 의존도 낮춤)
-  최종적으로 작동하는 솔루션 완성

관련 자료






- 코드: [14_Evaluation/01_Test-Dataset-Generator-RAGAS.ipynb](#)
- ragas 공식 문서: <https://docs.ragas.io/>
- ragas GitHub Issues: <https://github.com/explodinggradients/ragas/issues>
-  Related: #38 (RAGAS 테스트셋 생성)
- 선행작업: #37 (RAG 평가 개념 학습)

결론

ragas 사용 시 발생 가능한 문제

-  버전 충돌 (0.1.x vs 0.3.x)
-  패키지 의존성 지옥 (`langchain-community`, `langchain-core`)
-  치명적 버그 (`summary_embedding`, `CosineSimilarityBuilder`)
-  긴 실행 시간 (80분+)
-  API 할당량 낭비

직접 구현 시 장점

-  완전한 제어
-  빠른 실행 (2~3분)
-  높은 안정성 (100% 성공률)
-  낮은 의존성
-  쉬운 디버깅

학습 목적 달성

- ☒ ragas 개념 이해
- ☒ 합성 데이터 생성 원리 학습
- ☒ 문제 해결 능력 증명
- ☒ 실무 의사결정 경험
- ☒ 2일간의 고생이 헛되지 않음!

다음 단계

- RAG 평가 메트릭 학습 (Faithfulness, Answer Relevancy)
- 생성된 데이터셋으로 RAG 시스템 평가
- 다른 합성 데이터 생성 방법 탐구

재발 방지

ragas 사용 시 체크리스트

1. △ 버전 확인 필수 (`pip show ragas`)
2. △ 공식 문서 버전 일치 확인
3. △ 작은 데이터셋으로 먼저 테스트 (`docs=1, testset_size=1`)
4. △ 시간 제한 설정 (10분 이상 소요 시 중단)
5. ☒ 대안 준비 (직접 구현 코드 미리 작성)

권장 워크플로우

```
# 1단계: ragas 테스트 (5분 제한)
try:
    testset = generator.generate_with_langchain_docs(
        documents=docs[:1],
        testset_size=1,
    )
except Exception as e:
    print("ragas 실패, 직접 구현으로 전환")
    # 2단계: 직접 구현
    testset = generate_custom_testset(docs)
```

작성일: 2025-10-13

작성자: Jay Lee

소요 시간: 2일 (누적 4시간+ 대기)

최종 상태: ☒ 해결 완료 (직접 구현)