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

---

6. `MultiVectorRetriever`

- 실습에 활용할 문서: *소프트웨어정책연구소(`SPRi`) - 2023년 12월호*
  - 저자: 유재홍 (AI정책연구실 책임연구원), 이지수(AI정책연구실 위촉연구원)
  - 링크 - 링크: https://spri.kr/posts/view/23669
  - 파일명: `SPRI_AI_Brief_2023년12월호_F.pdf`

```
# 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("❌ 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.")
```

- **`전처리 과정`**
  - 텍스트 파일에서 데이터 로드 → 로드된 문서들을 지정된 크기로 분할
  - 분할된 문서들 = 추구 벡터화 및 검색 등의 작업에 사용 가능

```
from langchain_community.document_loaders import PyMuPDFLoader

loader = PyMuPDFLoader("../10_Retriever/data/SPRI_AI_Brief_2023년12월호_F.pdf")
docs = loader.load()
```

- **`docs`** 변수 = 데이터로부터 로드한 원본 도큐먼트

```
print(len(docs))                                    # 23
```

```
print(docs[5].page_content[:500])
```

- 6번째 페이지 일부 출력해보기

```
1. 정책/법제
2. 기업/산업
3. 기술/연구
4. 인력/교육
영국 AI 안전성 정상회의에 참가한 28개국, AI 위험에 공동 대응 선언
n 영국 블레츨리 파크에서 개최된 AI 안전성 정상회의에 참가한 28개국들이 ✦ 안전 보장을
```

```
위한 협력 방안을 담은 블레츨리 선언을 발표
n 첨단 AI를 개발하는 국가와 기업들은 AI 시스템에 대한 안전 테스트 계획에 합의했으며,
영국의 AI 안전 연구소가 전 세계 국가와 협력해 테스트를 주도할 예정
KEY Contents
£ AI 안전성 정상회의 참가국들, 블레츨리 선언 통해 AI 안전 보장을 위한 협력에 합의
n 2023년 11월 1~2일 영국 블레츨리 파크에서 열린 AI 안전성 정상회의(AI Safety Summit)에
참가한 28개국 대표들이 AI 위험 관리를 위한 '블레츨리 선언'을 발표
•선언은 AI 안전 보장을 위해 국가, 국제기구, 기업, 시민사회, 학계를 포함한 모든 이해관계자의 협력이
중요하다고 강조했으며,
```

---

3) `Chunk + 원본 문서 검색`

- 대용량 정보 검색 시: `더 작은 단위`로 정보를 `임베딩`하는 것이 유용할 수 있음
  - **`MultiVectorRetriever`** → 문서를 `여러 벡터`로 `저장`하고 `관리` 가능
  - **`docstore`** = 원본 문서 저장
  - **`vectoresotre`** = *`임베딩된 문서 저장`*
    - 문서룰 더 작은 단위로 나눠 더 정확한 검색이 가능
    - 원본 문서의 내용도 조회 가능

```python
# 자식 청크를 인덱싱하는 데 사용할 벡터 저장소
import uuid
from langchain.storage import InMemoryStore
from langchain_chroma import Chroma
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain.retrievers.multi_vector import MultiVectorRetriever

# 허깅페이스의 임베딩 모델 생성하기
embeddings = HuggingFaceEmbeddings(
        model_name="sentence-transformers/all-mpnet-base-v2",              # 768차원
        model_kwargs={'device': 'cpu'},
        encode_kwargs={'normalize_embeddings': True}
    )

vectorstore = Chroma(
    collection_name="small_bigger_chunks",
    embedding_function=embeddings,
)

# 부모 문서의 저장소 계층
store = InMemoryStore()

id_key = "doc_id"

# 검색기 (시작 시 비어 있음)
retriever = MultiVectorRetriever(
    vectorstore=vectorstore,
    byte_store=store,
    id_key=id_key,
)

# 문서 ID 생성하기
doc_ids = [str(uuid.uuid4()) for _ in docs]

# 두개의 생성된 id 확인하기
doc_ids
```

- 셀 출력 (9.3s)

```
['1805bc44-13a9-41d8-a14d-b1bf44539678',
'529ddd7a-9a21-4565-918e-5bff97b56446',
'0ff14000-5de3-4ef2-bc18-0c77243dde1f',
'e49c1163-768f-44bb-b0c7-dd057712f453',
'f4e4e965-80e1-49c6-a8ab-2493c4fde4ce',
'87dde3ca-f58e-411e-bca9-3485e052aecc',
'fd567373-971c-42d7-bebf-572dbd165780',
'cc3b7e74-9869-4b50-963b-133c0f84744a',
'6816e3ad-99df-457e-b17b-bf8fec279a65',
'919e761e-9d75-4ce9-b985-5db3c41cf7d8',
'3cd2d962-c039-429c-a68c-f697dd5dd96c',
'6136ed3f-922b-4425-9059-137b0244fa3e',
```

```
'22a231fe-cee6-401f-8876-680178accb37',
'43b07501-66ed-4cab-a1ce-98d07a600ff3',
'9e07384d-3602-469f-9349-fa21c6e2cc1a',
'07697bba-4cc8-42f1-8778-1ec6ef46e5bc',
'54d333d0-2e46-474e-acfd-1f2eab0113c2',
'39d06f11-ecec-47ca-b085-c35302c89632',
'4aab318d-b7d3-4fdf-afd6-4514e13c1e88',
'b7166404-aa6a-4c05-ac52-f0121cafa6c2',
'4c051212-901a-4306-92c5-812b4e962241',
'85f3eab5-d161-4f7b-8ac1-10e4ae6f7821',
'180511ed-2a51-44ef-b73e-081e3fc8b2e5']
```

- **문서 계층 분할기 생성**
  - **parent_text_splitter**: 큰 청크로 분할하기 위한 객체
  - **chile_text_splitter**: 더 작은 청크로 분할하기 위한 객체

```
# RecursiveCharacterTextSplitter 객체 생성하기
parent_text_splitter = RecursiveCharacterTextSplitter(chunk_size=600)

# 더 작은 청크를 생성하는 데 사용할 분할기
child_text_splitter = RecursiveCharacterTextSplitter(chunk_size=200)        # 부모보다 작게 설정
```

- 더 큰 `Chunk` 인 `Parent` 문서 생성하기

```
parent_docs = []

for i, doc in enumerate(docs):
    _id = doc_ids[i]                                                # 현재 문서의 ID 가져오기
    parent_doc = parent_text_splitter.split_documents([doc])    # 현재 문서를 하위 문서로 분할

    for _doc in parent_doc:
        _doc.metadata[id_key] = _id                                 # metadata에 문서 ID 를 저장
    parent_docs.extend(parent_doc)
```

- **parent_docs** 에 기입된 **doc_id** 확인하기

```
# 생성된 Parent 문서의 메타데이터 확인하기

parent_docs[0].metadata
```

- `parent_docs`의 `doc_id` 확인하기

```
{'producer': 'Hancom PDF 1.3.0.542',
'creator': 'Hwp 2018 10.0.0.13462',
'creationdate': '2023-12-08T13:28:38+09:00',
'source': '../10_Retriever/data/SPRI_AI_Brief_2023년12월호_F.pdf',
'file_path': '../10_Retriever/data/SPRI_AI_Brief_2023년12월호_F.pdf',
'total_pages': 23,
'format': 'PDF 1.4',
'title': '',
'author': 'dj',
'subject': '',
'keywords': '',
'moddate': '2023-12-08T13:28:38+09:00',
'trapped': '',
'modDate': "D:20231208132838+09'00'",
'creationDate': "D:20231208132838+09'00'",
'page': 0,
'doc_id': '1805bc44-13a9-41d8-a14d-b1bf44539678'}           # ✓ doc_id 확인하기
```

- 상대적으로 더 작은 **Chunk** 인 **Child** 문서 생성하기

```
child_docs = []

for i, doc in enumerate(docs):
    _id = doc_ids[i]                                                # 현재 문서의 ID 가져오기
    child_doc = child_text_splitter.split_documents([doc])      # 현재 문서를 하위 문서로 분할

    for _doc in child_doc:
```

```
        _doc.metadata[id_key] = _id                                    # metadata에 문서 ID 를 저장
    child_docs.extend(child_doc)
```

- **`child_docs`** 에 기입된 **`doc_id`** 확인하기

```
# 생성된 Child 문서의 메타데이터 확인하기

child_docs[0].metadata
```

- `child_docs`의 `doc_id` 확인하기

```
{'producer': 'Hancom PDF 1.3.0.542',
'creator': 'Hwp 2018 10.0.0.13462',
'creationdate': '2023-12-08T13:28:38+09:00',
'source': '../10 Retriever/data/SPRI AI Brief 2023년12월호_F.pdf',
'file_path': '../10 Retriever/data/SPRI AI Brief 2023년12월호_F.pdf',
'total_pages': 23,
'format': 'PDF 1.4',
'title': '',
'author': 'dj',
'subject': '',
'keywords': '',
'moddate': '2023-12-08T13:28:38+09:00',
'trapped': '',
'modDate': "D:20231208132838+09'00'",
'creationDate': "D:20231208132838+09'00'",
'page': 0,
'doc_id': '1805bc44-13a9-41d8-a14d-b1bf44539678'}           # ✓ doc_id 확인하기
```

- 각각 분할된 청크의 수 확인하기

```
print(f"분할된 parent_docs의 개수: {len(parent_docs)}")
print(f"분할된 child_docs의 개수: {len(child_docs)}")
```

- 각 분할된 청크의 수 확인하기

```
분할된 parent_docs의 개수: 73
분할된 child_docs의 개수: 440
```

- 벡터저장소에 새롭게 생성한 작게 쪼개진 하위문서 집합 추가하기
- 다음: 상위 문서는 생성한 `UUID`와 맵핑하여 **`docstore`** 에 추가하기
  - **`mset()`** 메서드 → 문서 `ID`, 문서 내용 = **`key-value`** 쌍으로 문서 저장소에 저장

```
# 벡터 저장소에 parent + child 문서를 추가
retriever.vectorstore.add_documents(parent_docs)
retriever.vectorstore.add_documents(child_docs)

# docstore 에 원본 문서를 저장
retriever.docstore.mset(list(zip(doc_ids, docs)))                      # 1m 24.3s
```

- 유사도 검색 수행하기
  - 가장 유사도가 높은 첫 번째 문서 조각 출력하기
  - **`retriever.vectorstore.similarity_search`** 메서드 → `child` + `parent` 문서 `chunk` 내에서 검색을 수행

```
# vectorstore의 유사도 검색 수행하기
relevant_chunks = retriever.vectorstore.similarity_search(
    "삼성전자가 만든 생성형 AI 의 이름은?"
)

# 출력하기
print(f"검색된 문서의 개수: {len(relevant_chunks)}")
```

- 검색된 문서의 개수: 4 (0.2s)

```
for chunk in relevant_chunks:
    print(chunk.page_content, end="\n\n")
    print(">" * 100, end="\n\n")
```

- 셀 출력

```
중점 지원할 예정
•포럼에 따르면 AI 레드팀에 대한 자금 지원은 AI 모델의 안전과 보안 기준의 개선과 함께 AI 시스템
위험 대응 방안에 관한 산업계와 정부, 시민사회의 통찰력 확보에 도움이 될 전망으로, 포럼은 향후 몇
달 안에 기금 지원을 위한 제안 요청을 받을 계획

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

같이 AI 도구를 사용해 사실적으로 변경되거나 합성된 콘텐츠에는 AI 라벨을 표시 필요
•유튜브는 이러한 규칙이 선거나 분쟁 상황, 공중 보건, 공직자 관련 문제와 같이 민감한 주제를 다루는
콘텐츠에서 특히 중요하다고 강조했으며, 크리에이터가 AI로 제작한 콘텐츠에 AI 라벨을 표시하지 않으면

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

•첨단 AI 시스템의 개발 과정에서 AI 수명주기 전반에 걸쳐 위험을 평가 및 완화하는 조치를 채택하고,
첨단 AI 시스템의 출시와 배포 이후 취약점과 오용 사고, 오용 유형을 파악해 완화
•첨단 AI 시스템의 성능과 한계를 공개하고 적절하거나 부적절한 사용영역을 알리는 방법으로 투명성을
보장하고 책임성을 강화

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

AI 거버넌스와 위험 관리 정책을 마련
•AI 수명주기 전반에 걸쳐 물리보안, 사이버보안, 내부자 위협 보안을 포함한 강력한 보안 통제 구현
•사용자가 AI 생성 콘텐츠를 식별할 수 있도록 워터마크를 비롯하여 기술적으로 가능한 기법으로
신뢰할 수 있는 콘텐츠 인증과 출처 확인 메커니즘을 개발 및 구축

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

- `retriever.invoke()` 메서드 ⟶ 쿼리 실행 (원본 문서의 전체 내용 검색함)

```
relevant_docs = retriever.invoke("삼성전자가 만든 생성형 AI 의 이름은?")

print(f"검색된 문서의 개수: {len(relevant_docs)}", end="\n\n")
print("=" * 100, end="\n\n")
print(relevant_docs[0].page_content)
```

- 셀 출력

```
검색된 문서의 개수: 3

====================================================================================================

1. 정책/법제
2. 기업/산업
3. 기술/연구
4. 인력/교육
미국 프런티어 모델 포럼, 1,000만 달러 규모의 AI 안전 기금 조성
n 구글, 앤스로픽, 마이크로소프트, 오픈AI가 참여하는 프런티어 모델 포럼이 자선단체와 함께 AI
안전 연구를 위한 1,000만 달러 규모의 AI 안전 기금을 조성
n 프런티어 모델 포럼은 AI 모델의 취약점을 발견하고 검증하는 레드팀 활동을 지원하기 위한
모델 평가 기법 개발에 자금을 중점 지원할 계획
KEY Contents
£ 프런티어 모델 포럼, 자선단체와 함께 AI 안전 연구를 위한 기금 조성
n 구글, 앤스로픽, 마이크로소프트, 오픈AI가 출범한 프런티어 모델 포럼이 2023년 10월 25일 AI 안전
연구를 위한 기금을 조성한다고 발표
•참여사들은 맥거번 재단(Patrick J. McGovern Foundation), 데이비드 앤 루실 패커드 재단(The
David and Lucile Packard Foundation) 등의 자선단체와 함께 AI 안전 연구를 위한 기금에
1,000만 달러 이상을 기부
•또한 신기술의 거버넌스와 안전 분야에서 전문성을 갖춘 브루킹스 연구소 출신의 크리스 메서롤(Chris
Meserole)을 포럼의 상무이사로 임명
n 최근 AI 기술이 급속히 발전하면서 AI 안전에 관한 연구가 부족한 시점에, 포럼은 이러한 격차를 해소
하기 위해 AI 안전 기금을 조성
•참여사들은 지난 7월 백악관 주재의 AI 안전 서약에서 외부자의 AI 시스템 취약점 발견과 신고를
촉진하기로 약속했으며, 약속을 이행하기 위해 기금을 활용해 외부 연구집단의 AI 시스템 평가에
자금을 지원할 계획
£ AI 안전 기금으로 AI 레드팀을 위한 모델 평가 기법 개발을 중점 지원할 계획
n 프런티어 모델 포럼은 AI 안전 기금을 통해 AI 레드팀 활동을 위한 새로운 모델 평가 기법의 개발을
중점 지원할 예정
•포럼에 따르면 AI 레드팀에 대한 자금 지원은 AI 모델의 안전과 보안 기준의 개선과 함께 AI 시스템
```

```
위험 대응 방안에 관한 산업계와 정부, 시민사회의 통찰력 확보에 도움이 될 전망으로, 포럼은 향후 몇
달 안에 기금 지원을 위한 제안 요청을 받을 계획
n 프런티어 모델 포럼은 출범 이후 업계 전반에 걸쳐 AI 레드팀 구성에 관한 모범사례 공유를 추진하는
한편, 첨단 AI 모델의 취약점이나 잠재적으로 위험한 기능 및 위험 완화 관련 정보를 공유할 수
있는 공개 절차도 개발 중
☞ 출처: Google, Anthropic, Google, Microsoft and OpenAI announce Executive Director of the Frontier Model
Forum and over $10 million for a new AI Safety Fund, 2023.10.25.
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling p
To disable this warning, you can either:
    - Avoid using `tokenizers` before the fork if possible
    - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
```

- `MMR` 지원
  - `retriever`가 벡터 데이터베이스에서 기본적으로 수행하는 검색 유형: *`유사도 검색`*
  - `LangChain Vector Store`: **`MMR`** (`Max Marginal Relevance`) 검색도 지원 → **`search_type`** 속성 설정
    - `retriever` 객체의 `search_type` 속성 = **`SearchType.mmr`** 로 설정하기
    - 검색 시 `MMR` (`Maximal Marginal Relevance`) 알고리즘 사용하도록 지정하는 것

```
from langchain.retrievers.multi_vector import SearchType

# 검색 유형을 MMR(Maximal Marginal Relevance)로 설정하기
retriever.search_type = SearchType.mmr

# 관련 문서 전체 검색하기
print(retriever.invoke("삼성전자가 만든 생성형 AI 의 이름은?")[0].page_content)
```

- `MMR`로 검색해보기 (0.1s)

```
1. 정책/법제
2. 기업/산업
3. 기술/연구
4. 인력/교육
미국 프런티어 모델 포럼, 1,000만 달러 규모의 AI 안전 기금 조성
n 구글, 앤스로픽, 마이크로소프트, 오픈AI가 참여하는 프런티어 모델 포럼이 자선단체와 함께 AI
안전 연구를 위한 1,000만 달러 규모의 AI 안전 기금을 조성
n 프런티어 모델 포럼은 AI 모델의 취약점을 발견하고 검증하는 레드팀 활동을 지원하기 위한
모델 평가 기법 개발에 자금을 중점 지원할 계획
KEY Contents
£ 프런티어 모델 포럼, 자선단체와 함께 AI 안전 연구를 위한 기금 조성
n 구글, 앤스로픽, 마이크로소프트, 오픈AI가 출범한 프런티어 모델 포럼이 2023년 10월 25일 AI 안전
연구를 위한 기금을 조성한다고 발표
•참여사들은 맥거번 재단(Patrick J. McGovern Foundation), 데이비드 앤 루실 패커드 재단(The
David and Lucile Packard Foundation) 등의 자선단체와 함께 AI 안전 연구를 위한 기금에
1,000만 달러 이상을 기부
•또한 신기술의 거버넌스와 안전 분야에서 전문성을 갖춘 브루킹스 연구소 출신의 크리스 메서롤(Chris
Meserole)을 포럼의 상무이사로 임명
n 최근 AI 기술이 급속히 발전하면서 AI 안전에 관한 연구가 부족한 시점에, 포럼은 이러한 격차를 해소
하기 위해 AI 안전 기금을 조성
•참여사들은 지난 7월 백악관 주재의 AI 안전 서약에서 외부자의 AI 시스템 취약점 발견과 신고를
촉진하기로 약속했으며, 약속을 이행하기 위해 기금을 활용해 외부 연구집단의 AI 시스템 평가에
자금을 지원할 계획
£ AI 안전 기금으로 AI 레드팀을 위한 모델 평가 기법 개발을 중점 지원할 계획
n 프런티어 모델 포럼은 AI 안전 기금을 통해 AI 레드팀 활동을 위한 새로운 모델 평가 기법의 개발을
중점 지원할 예정
•포럼에 따르면 AI 레드팀에 대한 자금 지원은 AI 모델의 안전과 보안 기준의 개선과 함께 AI 시스템
위험 대응 방안에 관한 산업계와 정부, 시민사회의 통찰력 확보에 도움이 될 전망으로, 포럼은 향후 몇
달 안에 기금 지원을 위한 제안 요청을 받을 계획
n 프런티어 모델 포럼은 출범 이후 업계 전반에 걸쳐 AI 레드팀 구성에 관한 모범사례 공유를 추진하는
한편, 첨단 AI 모델의 취약점이나 잠재적으로 위험한 기능 및 위험 완화 관련 정보를 공유할 수
있는 공개 절차도 개발 중
☞ 출처: Google, Anthropic, Google, Microsoft and OpenAI announce Executive Director of the Frontier Model
Forum and over $10 million for a new AI Safety Fund, 2023.10.25.
```

```
from langchain.retrievers.multi_vector import SearchType

# 검색 유형을 similarity_score_threshold로 설정
retriever.search_type = SearchType.similarity_score_threshold
```

```
retriever.search_kwargs = {"score_threshold": 0.3}

# 관련 문서 전체를 검색
print(retriever.invoke("삼성전자가 만든 생성형 AI 의 이름은?")[0].page_content)
```

- `similarity_score_threshold = 0.3`으로 설정

```
1. 정책/법제
2. 기업/산업
3. 기술/연구
4. 인력/교육
미국 프런티어 모델 포럼, 1,000만 달러 규모의 AI 안전 기금 조성
n 구글, 앤스로픽, 마이크로소프트, 오픈AI가 참여하는 프런티어 모델 포럼이 자선단체와 함께 AI
안전 연구를 위한 1,000만 달러 규모의 AI 안전 기금을 조성
n 프런티어 모델 포럼은 AI 모델의 취약점을 발견하고 검증하는 레드팀 활동을 지원하기 위한
모델 평가 기법 개발에 자금을 중점 지원할 계획
KEY Contents
£ 프런티어 모델 포럼, 자선단체와 함께 AI 안전 연구를 위한 기금 조성
n 구글, 앤스로픽, 마이크로소프트, 오픈AI가 출범한 프런티어 모델 포럼이 2023년 10월 25일 AI 안전
연구를 위한 기금을 조성한다고 발표
•참여사들은 맥거번 재단(Patrick J. McGovern Foundation), 데이비드 앤 루실 패커드 재단(The
David and Lucile Packard Foundation) 등의 자선단체와 함께 AI 안전 연구를 위한 기금에
1,000만 달러 이상을 기부
•또한 신기술의 거버넌스와 안전 분야에서 전문성을 갖춘 브루킹스 연구소 출신의 크리스 메서롤(Chris
Meserole)을 포럼의 상무이사로 임명
n 최근 AI 기술이 급속히 발전하면서 AI 안전에 관한 연구가 부족한 시점에, 포럼은 이러한 격차를 해소
하기 위해 AI 안전 기금을 조성
•참여사들은 지난 7월 백악관 주재의 AI 안전 서약에서 외부자의 AI 시스템 취약점 발견과 신고를
촉진하기로 약속했으며, 약속을 이행하기 위해 기금을 활용해 외부 연구집단의 AI 시스템 평가에
자금을 지원할 계획
£ AI 안전 기금으로 AI 레드팀을 위한 모델 평가 기법 개발을 중점 지원할 계획
n 프런티어 모델 포럼은 AI 안전 기금을 통해 AI 레드팀 활동을 위한 새로운 모델 평가 기법의 개발을
중점 지원할 예정
•포럼에 따르면 AI 레드팀에 대한 자금 지원은 AI 모델의 안전과 보안 기준의 개선과 함께 AI 시스템
위험 대응 방안에 관한 산업계와 정부, 시민사회의 통찰력 확보에 도움이 될 전망으로, 포럼은 향후 몇
달 안에 기금 지원을 위한 제안 요청을 받을 계획
n 프런티어 모델 포럼은 출범 이후 업계 전반에 걸쳐 AI 레드팀 구성에 관한 모범사례 공유를 추진하는
한편, 첨단 AI 모델의 취약점이나 잠재적으로 위험한 기능 및 위험 완화 관련 정보를 공유할 수
있는 공개 절차도 개발 중
☞ 출처: Google, Anthropic, Google, Microsoft and OpenAI announce Executive Director of the Frontier Model
Forum and over $10 million for a new AI Safety Fund, 2023.10.25.
```

```
from langchain.retrievers.multi_vector import SearchType

# 검색 유형을 similarity로 설정, k값을 1로 설정
retriever.search_type = SearchType.similarity
retriever.search_kwargs = {"k": 1}

# 관련 문서 전체 검색하기
print(len(retriever.invoke("삼성전자가 만든 생성형 AI 의 이름은?")))               # 1
```

---

4) 요약본 (summary)을 벡터저장소에 저장

- 요약의 이점
  - 종종 `chunk`의 내용을 `보다 정확하게 추출` 가능 → 더 나은 검색 결과를 얻을 수 있음
  - 요약을 생성하는 방법, 임베딩하는 방법 알아보기

```
# PDF 파일을 로드하고 텍스트를 분할하기 위한 라이브러리 임포트
from langchain_community.document_loaders import PyMuPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter

# PDF 파일 로더 초기화
loader = PyMuPDFLoader("../10_Retriever/data/SPRI_AI_Brief_2023년12월호_F.pdf")

# 텍스트 분할
text_splitter = RecursiveCharacterTextSplitter(chunk_size=600, chunk_overlap=50)

# PDF 파일 로드 및 텍스트 분할 실행
split_docs = loader.load_and_split(text_splitter)
```

```
# 분할된 문서의 개수 출력
print(f"분할된 문서의 개수: {len(split_docs)}")
```

- 분할된 문서의 개수: 61

```
import os
from dotenv import load_dotenv
from groq import Groq
from langchain_core.documents import Document
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate

load_dotenv()

# Groq LLM 래퍼 클래스
class GroqLLMWrapper:
    def __init__(self, api_key=None, model="llama-3.3-70b-versatile"):
        self.client = Groq(api_key=api_key or os.getenv("GROQ_API_KEY"))
        self.model = model

    def invoke(self, prompt):
        """LangChain 호환 invoke 메서드"""
        # 프롬프트가 딕셔너리 형태로 올 때 처리
        if isinstance(prompt, dict):
            prompt_text = str(prompt)
        else:
            prompt_text = str(prompt)

        response = self.client.chat.completions.create(
            model=self.model,
            messages=[{"role": "user", "content": prompt_text}],
            temperature=0.1,
            max_tokens=500
        )
        return response.choices[0].message.content

    print("✅ Groq 불러오기 성공!")

# 사용하기
groq_llm = GroqLLMWrapper()                                    # ✅ Groq 불러오기 성공! (0.2s)
```

```
# 환경변수 설정 확인
if not os.getenv("GROQ_API_KEY"):
    print("⚠️ GROQ_API_KEY 환경변수를 설정해주세요!")

# 🎯 간단한 함수형 접근
def create_groq_llm(api_key=None, model="llama-3.3-70b-versatile"):
    """Groq LLM 함수 생성"""
    client = Groq(api_key=api_key or os.getenv("GROQ_API_KEY"))

    def groq_invoke(prompt):
        # 프롬프트 처리
        if isinstance(prompt, dict):
            prompt_text = str(prompt)
        else:
            prompt_text = str(prompt)

        response = client.chat.completions.create(
            model=model,
            messages=[{"role": "user", "content": prompt_text}],
            temperature=0.1,
            max_tokens=500
        )
        return response.choices[0].message.content

    return groq_invoke

# Groq LLM 생성
groq_function = create_groq_llm()
```

```
from langchain_core.runnables import RunnableLambda

# 두 번째 체인 (완전 호환!)
summary_chain = (
    {"doc": lambda x: x.page_content}
    | ChatPromptTemplate.from_messages([
        ("system", "You are an expert in summarizing documents in Korean."),
        ("user", "Summarize the following documents in 3 sentences in bullet points format.\n\n{doc}"),
    ])
    | RunnableLambda(groq_function)                    # 함수를 Runnable로!
    | StrOutputParser()
)
```

```
print("🎯 Groq 초고속 무료 LLM 준비 완료!")                # 🎯 Groq 초고속 무료 LLM 준비 완료!
```

- `chain.batch` 메서드 사용 → `docs` 리스트의 문서들을 일괄 요약
  - `max_concurrency` = `10` → 최대 10개의 문서를 동시에 처리할 수 있도록 설정

```
# split_docs 데이터 구조 확인하기
print("🔍 split_docs 데이터 구조 진단:")
print(f"📊 총 개수: {len(split_docs)}")
print(f"🔬 첫 번째 항목 타입: {type(split_docs[0])}")

if split_docs:
    first_item = split_docs[0]
    print(f"📋 첫 번째 항목 내용:")

    if isinstance(first_item, dict):
        print("  🔑 딕셔너리 키들:", list(first_item.keys()))
        for key, value in first_item.items():
            if isinstance(value, str):
                print(f"    {key}: {value[:100]}..." if len(str(value)) > 100 else f"    {key}: {value}")
            else:
                print(f"    {key}: {type(value)} - {str(value)[:50]}...")
    else:
        print(f"  📄 Document 객체: {first_item}")
        if hasattr(first_item, 'page_content'):
            print(f"    page_content: {first_item.page_content[:100]}...")
```

- 데이터 구조 확인하기

```
🔍 split_docs 데이터 구조 진단:
📊 총 개수: 61
🔬 첫 번째 항목 타입: <class 'langchain_core.documents.base.Document'>
📋 첫 번째 항목 내용:
  📄 Document 객체: page_content='2023년 12월호' metadata={'producer': 'Hancom PDF 1.3.0.542', 'creator': 'Hwp 2018 10
    page_content: 2023년 12월호...
```

```
# 람다 함수만 수정
summary_chain_simple = (
    {"doc": lambda x: str(x.get('page_content', '') if isinstance(x, dict)
                          else getattr(x, 'page_content', str(x)))}
    | ChatPromptTemplate.from_messages([
        ("system", "You are an expert in summarizing documents in Korean."),
        ("user", "Summarize the following documents in 3 sentences in bullet points format.\n\n{doc}"),
    ])
    | RunnableLambda(groq_function)
    | StrOutputParser()
)

# 실행
summaries = []

for doc in split_docs:
    try:
        summary = summary_chain_simple.invoke(doc)
        summaries.append(summary)
        print(f"✅ 문서 처리 완료: {len(summaries)}/{len(split_docs)}")
    except Exception as e:
        print(f"❌ 오류: {e}")
        summaries.append("요약 실패")

print(f"🎉 총 {len(summaries)}개 요약 완료!")
```

- 배치 처리 및 요약 (1m 54.2s)

```
✅ 문서 처리 완료: 1/61
✅ 문서 처리 완료: 2/61
✅ 문서 처리 완료: 3/61

    (중략...)

✅ 문서 처리 완료: 59/61
✅ 문서 처리 완료: 60/61
✅ 문서 처리 완료: 61/61
🎉 총 61개 요약 완료!
```

```
len(summaries)                                                  # 61
```

- 요약된 내용 출력 → 결과 확인하기

```
# 원본 문서의 내용 출력하기
print(split_docs[33].page_content, end="\n\n")


# 요약 출력하기
print("[요약]")
print(summaries[33])
```

- 결과 출력해보기

```
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 외부로 사용자 정보가 유출될 위험이 없다는 장점을 보유
KEY Contents
£ 언어, 코드, 이미지의 3개 모델로 구성된 삼성 가우스, 온디바이스 작동 지원
n 삼성전자가 2023년 11월 8일 열린 ‘삼성 AI 포럼 2023’ 행사에서 자체 개발한 생성 AI 모델
‘삼성 가우스’를 최초 공개
•정규분포 이론을 정립한 천재 수학자 가우스(Gauss)의 이름을 본뜬 삼성 가우스는 다양한 상황에
최적화된 크기의 모델 선택이 가능
•삼성 가우스는 라이선스나 개인정보를 침해하지 않는 안전한 데이터를 통해 학습되었으며,
온디바이스에서 작동하도록 설계되어 외부로 사용자의 정보가 유출되지 않는 장점을 보유
•삼성전자는 삼성 가우스를 활용한 온디바이스 AI 기술도 소개했으며, 생성 AI 모델을 다양한 제품에

[요약]
Here is a summary of the document in 3 sentences in bullet points format:

* Samsung Electronics has developed and released its own AI model called "Samsung Gauss", which consists of three
* Samsung Gauss was first introduced at the "Samsung AI Forum 2023" event on November 8, 2023, and is designed to
* Samsung plans to gradually integrate Samsung Gauss into various products, leveraging its on-device AI technolog
```

```
# 🎯 API 없이 간단한 가설 쿼리 생성
def generate_simple_questions(doc_content):
    """간단한 규칙 기반 가설 쿼리 생성"""

    # 문서에서 핵심 키워드 추출
    import re

    # 회사명, 제품명, 기술명 추출
    companies = re.findall(r'[가-힣]+(?:전자|그룹|회사|기업)', doc_content)
    products = re.findall(r'[A-Za-z가-힣\s]+(?:AI|모델|시스템|플랫폼)', doc_content)
    technologies = re.findall(r'(?:인공지능|머신러닝|딥러닝|생성형\s*AI|LLM)', doc_content)

    questions = []

    # 규칙 기반 질문 생성
    if companies:
        for company in companies[:2]:  # 최대 2개
            questions.append(f"{company}의 주요 AI 기술은 무엇인가요?")
            questions.append(f"{company}이 개발한 제품의 특징은 무엇인가요?")

    if products:
        for product in products[:2]:
            questions.append(f"{product}의 핵심 기능은 무엇인가요?")

    if technologies:
        for tech in technologies[:1]:
            questions.append(f"{tech} 기술의 장점은 무엇인가요?")

    # 기본 질문들
    base_questions = [
        "이 문서의 핵심 내용은 무엇인가요?",
        "주요 기술적 특징은 무엇인가요?",
        "어떤 혁신적인 점이 있나요?",
```

```
            "시장에 미치는 영향은 무엇인가요?",
            "향후 전망은 어떻게 될까요?"
        ]

        questions.extend(base_questions)

        # 중복 제거 및 최대 3개 선택
        unique_questions = list(set(questions))[:3]

        # 3개 미만이면 기본 질문으로 채우기
        while len(unique_questions) < 3:
            unique_questions.append(f"이 내용과 관련된 주제 {len(unique_questions)+1}은 무엇인가요?")

        return unique_questions

    # 테스트
    sample_doc = split_docs[33]  # 삼성 가우스 문서
    test_questions = generate_simple_questions(sample_doc.page_content)

    print("🔍 생성된 가설 쿼리:")
    for i, q in enumerate(test_questions, 1):
        print(f"{i}. {q}")
```

- API 없이 가설 쿼리 생성

```
🔍 생성된 가설 쿼리:
1.  자체 개발 생성 AI의 핵심 기능은 무엇인가요?
2. 향후 전망은 어떻게 될까요?
3. 삼성전자의 주요 AI 기술은 무엇인가요?
```

```
# 🚀 현재: sentence-transformers/all-mpnet-base-v2 (768차원)
# 🚀 업그레이드: 더 높은 차원 + 다국어 특화 모델

from langchain_huggingface import HuggingFaceEmbeddings

# 옵션 1: 한국어 특화 고성능 모델
korean_embeddings = HuggingFaceEmbeddings(
    model_name="jhgan/ko-sroberta-multitask",  # 한국어 최적화, 768차원
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
)

# 옵션 2: 다국어 고성능 모델 (1024차원!)
multilingual_embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/paraphrase-multilingual-mpnet-base-v2",  # 1024차원
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
)

# 옵션 3: 최신 고성능 모델
high_perf_embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L12-v2",  # 384차원, 매우 빠름
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
)

print("🎯 고성능 임베딩 모델들 준비 완료!")
```

- 2. 🎯 고성능 임베딩 모델들 준비 완료! (3m 18.7s)
  - 설치 결과

```
# 3. 리트리버 성능 최적화
# 🎯 멀티 전략 리트리버
from langchain.retrievers import EnsembleRetriever
from langchain_community.retrievers import BM25Retriever

def create_optimized_retriever(split_docs, embeddings):
    """최적화된 하이브리드 리트리버 생성"""

    # 1. 벡터 검색용 Chroma
    vectorstore = Chroma(
        collection_name="optimized_docs",
        embedding_function=embeddings,
    )

    # 2. 키워드 검색용 BM25
    bm25_retriever = BM25Retriever.from_documents(split_docs)
    bm25_retriever.k = 5
```

```
    # 3. 벡터 검색 리트리버
    vector_retriever = vectorstore.as_retriever(
        search_type="mmr",  # MMR로 다양성 확보
        search_kwargs={
            "k": 10,
            "lambda_mult": 0.7,  # 관련성 70%, 다양성 30%
        }
    )

    # 4. 하이브리드 앙상블 리트리버
    ensemble_retriever = EnsembleRetriever(
        retrievers=[vector_retriever, bm25_retriever],
        weights=[0.7, 0.3]  # 벡터 70%, 키워드 30%
    )

    # 문서 추가
    vectorstore.add_documents(split_docs)

    return ensemble_retriever, vectorstore

# 최적화된 리트리버 생성
optimized_retriever, optimized_vectorstore = create_optimized_retriever(
    split_docs,
    multilingual_embeddings  # 고성능 임베딩 사용
)

print("🚀 최적화된 하이브리드 리트리버 준비 완료!")          # 🚀 최적화된 하이브리드 리트리버 준비 완료!
```

```
# 4. 리트리버 성능 테스트

# 🎯 성능 비교 테스트
test_queries = [
    "삼성전자가 만든 생성형 AI의 이름은?",
    "AI 안전성과 관련된 내용은?",
    "프런티어 모델 포럼의 활동은?",
    "온디바이스 AI의 장점은?",
    "블레츨리 선언의 내용은?"
]

def test_retriever_performance(retriever, query, retriever_name):
    """리트리버 성능 테스트"""
    print(f"\n🔍 {retriever_name} 테스트: '{query}'")

    try:
        results = retriever.invoke(query)
        print(f"   📊 검색된 문서 수: {len(results)}")

        if results:
            # 첫 번째 결과의 관련성 미리보기
            first_result = results[0].page_content[:200]
            print(f"   📝 첫 번째 결과: {first_result}...")

            # 검색된 모든 문서에서 키워드 매칭 확인
            query_keywords = query.replace("?", "").split()
            matched_docs = 0

            for doc in results[:3]:  # 상위 3개만 확인
                content_lower = doc.page_content.lower()
                if any(keyword.lower() in content_lower for keyword in query_keywords):
                    matched_docs += 1

            accuracy = (matched_docs / min(3, len(results))) * 100
            print(f"   ✅ 키워드 매칭 정확도: {accuracy:.1f}%")

    except Exception as e:
        print(f"   ❌ 오류: {e}")

# 성능 테스트 실행
for query in test_queries[:3]:  # 첫 3개만 테스트
    test_retriever_performance(optimized_retriever, query, "최적화 리트리버")
```

- 4. 리트리버 성능 테스트

```
🔍 최적화 리트리버 테스트: '삼성전자가 만든 생성형 AI의 이름은?'
📊 검색된 문서 수: 13
📝 첫 번째 결과: 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...
✅ 키워드 매칭 정확도: 66.7%

🔍 최적화 리트리버 테스트: 'AI 안전성과 관련된 내용은?'
📊 검색된 문서 수: 13
📝 첫 번째 결과: AI 시스템의 안전을 보장할 책임이 있다고 지적
•각국은 AI 안전 보장을 위해 첨단 AI 개발기업의 투명성 향상, 적절한 평가지표와 안전 테스트 도구
개발, 공공부문 역량 구축과 과학 연구개발 등의 분야에서 협력하기로 합의
£ 영국 총리, 정부 주도의 첨단 AI 시스템 안전 테스트 계획 발표
n 리시 수낙 영국 총리는 AI 안전성 정상회의를 마무리하며 ...
✅ 키워드 매칭 정확도: 100.0%

🔍 최적화 리트리버 테스트: '프런티어 모델 포럼의 활동은?'
📊 검색된 문서 수: 12
📝 첫 번째 결과: 달 안에 기금 지원을 위한 제안 요청을 받을 계획
n 프런티어 모델 포럼은 출범 이후 업계 전반에 걸쳐 AI 레드팀 구성에 관한 모범사례 공유를 추진하는
한편, 첨단 AI 모델의 취약점이나 잠재적으로 위험한 기능 및 위험 완화 관련 정보를 공유할 수
있는 공개 절차도 개발 중
☞ 출처: Google, Anthropic, Google, Microsoft ...
✅ 키워드 매칭 정확도: 100.0%
```

```python
# 5. MultiVector Retriever 재구성하기 (without API)

# 🎯 API 없는 MultiVector 리트리버
def create_no_api_multivector_retriever(split_docs, embeddings):
    """API 없이 MultiVector 리트리버 생성"""

    import uuid
    from langchain.storage import InMemoryStore
    from langchain.retrievers.multi_vector import MultiVectorRetriever

    # 벡터 저장소
    vectorstore = Chroma(
        collection_name="multivector_no_api",
        embedding_function=embeddings,
    )

    # 문서 저장소
    store = InMemoryStore()
    id_key = "doc_id"

    # MultiVector 리트리버
    retriever = MultiVectorRetriever(
        vectorstore=vectorstore,
        byte_store=store,
        id_key=id_key,
    )

    # 문서 ID 생성
    doc_ids = [str(uuid.uuid4()) for _ in split_docs]

    # 방법 1: 작은 청크 + 원본 문서 (API 불필요)
    child_text_splitter = RecursiveCharacterTextSplitter(chunk_size=200)

    child_docs = []
    for i, doc in enumerate(split_docs):
        _id = doc_ids[i]
        child_chunks = child_text_splitter.split_documents([doc])

        for chunk in child_chunks:
            chunk.metadata[id_key] = _id
        child_docs.extend(child_chunks)

    # 벡터 저장소에 작은 청크 추가 (검색용)
    retriever.vectorstore.add_documents(child_docs)

    # 원본 문서는 docstore에 저장 (반환용)
    retriever.docstore.mset(list(zip(doc_ids, split_docs)))

    return retriever

# API 없는 MultiVector 리트리버 생성
no_api_retriever = create_no_api_multivector_retriever(split_docs, multilingual_embeddings)

print("✅ API 없는 MultiVector 리트리버 완료!")

# 테스트
test_result = no_api_retriever.invoke("삼성전자가 만든 생성형 AI의 이름은?")
print(f"\n🔍 검색 결과 ({len(test_result)}개):")
```

```
if test_result:
    print(f"첫 번째 결과: {test_result[0].page_content[:300]}...")
```

- 5. API 없는 MultiVector Retriever 실행 (21.1s)

```
✅ API 없는 MultiVector 리트리버 완료!

🔍 검색 결과 (2개):
첫 번째 결과: 삼성전자, '삼성 개발자 콘퍼런스 코리아 2023' 개최, 2023.11.14.
TechRepublic, Samsung Gauss: Samsung Research Reveals Generative AI, 2023.11.08....
```

```
# 🚀 내일 Groq API 리셋 후 실행할 완전 복구 코드

def complete_recovery_setup():
    """Jay를 위한 완전 복구 설정"""

    from langchain_community.document_loaders import PyMuPDFLoader
    from langchain_text_splitters import RecursiveCharacterTextSplitter
    from langchain_huggingface import HuggingFaceEmbeddings
    from langchain_chroma import Chroma
    from langchain.storage import InMemoryStore
    from langchain.retrievers.multi_vector import MultiVectorRetriever
    import uuid

    print("🔄 Jay의 시스템 완전 복구 중...")

    # 1. 문서 로드 및 분할
    loader = PyMuPDFLoader("../10_Retriever/data/SPRI_AI_Brief_2023년12월호_F.pdf")
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=600, chunk_overlap=50)
    split_docs = loader.load_and_split(text_splitter)
    print(f"✅ 문서 분할 완료: {len(split_docs)}개")

    # 2. 고성능 임베딩 모델
    embeddings = HuggingFaceEmbeddings(
        model_name="sentence-transformers/paraphrase-multilingual-mpnet-base-v2",
        model_kwargs={'device': 'cpu'},
        encode_kwargs={'normalize_embeddings': True}
    )
    print("✅ 고성능 임베딩 모델 로드 완료")

    # 3. MultiVector 리트리버 (3가지 전략)
    retrievers = {}

    # 전략 1: 작은 청크 + 원본 (API 불필요)
    retrievers['chunk_based'] = create_no_api_multivector_retriever(split_docs, embeddings)

    # 전략 2: 요약 기반
    # (API 필요, 할당량 다 함)
    # retrievers['summary_based'] = create_summary_based_retriever(split_docs, embeddings)

    # 전략 3: 가설 쿼리 기반 (API 없는 버전)
    retrievers['question_based'] = create_question_based_retriever(split_docs, embeddings)

    print("✅ 모든 리트리버 생성 완료")

    # 4. 성능 테스트
    test_query = "삼성전자가 만든 생성형 AI의 이름은?"

    for name, retriever in retrievers.items():
        try:
            results = retriever.invoke(test_query)
            print(f"🔍 {name}: {len(results)}개 문서 검색됨")
            if results and "삼성" in results[0].page_content and "가우스" in results[0].page_content:
                print(f"   ✅ 정확한 답변 발견!")
            else:
                print(f"   ⚠️ 답변 검토 필요")
        except Exception as e:
            print(f"   ❌ {name} 오류: {e}")

    return retrievers, split_docs, embeddings

def create_question_based_retriever(split_docs, embeddings):
    """규칙 기반 가설 쿼리 MultiVector 리트리버"""

    from langchain_core.documents import Document

    # 문서별 가설 쿼리 생성 (API 없이)
    question_docs = []
    doc_ids = [str(uuid.uuid4()) for _ in split_docs]

    for i, doc in enumerate(split_docs):
        questions = generate_simple_questions(doc.page_content)
```

```
            for q in questions:
                question_docs.append(
                    Document(page_content=q, metadata={"doc_id": doc_ids[i]})
                )

        # MultiVector 리트리버 설정
        vectorstore = Chroma(
            collection_name="question_based",
            embedding_function=embeddings,
        )
        store = InMemoryStore()

        retriever = MultiVectorRetriever(
            vectorstore=vectorstore,
            byte_store=store,
            id_key="doc_id",
        )

        # 가설 쿼리 추가
        retriever.vectorstore.add_documents(question_docs)
        # 원본 문서 저장
        retriever.docstore.mset(list(zip(doc_ids, split_docs)))

        return retriever

    # 지금 실행 가능한 부분
    print("🎯 현재 실행 가능한 설정:")
    no_api_retriever = create_no_api_multivector_retriever(split_docs, multilingual_embeddings)
    test_result = no_api_retriever.invoke("삼성전자가 만든 생성형 AI의 이름은?")

    if test_result:
        for i, doc in enumerate(test_result[:2]):
            print(f"\n📄 결과 {i+1}:")
            print(doc.page_content[:300] + "...")
            if "삼성" in doc.page_content and ("가우스" in doc.page_content or "AI" in doc.page_content):
                print("   ✅ 관련 답변 발견!")
```

- 복구 코드 실행 결과 (51.0s)

```
🎯 현재 실행 가능한 설정:

📄 결과 1:
삼성전자, ‘삼성 개발자 콘퍼런스 코리아 2023’ 개최, 2023.11.14.
TechRepublic, Samsung Gauss: Samsung Research Reveals Generative AI, 2023.11.08....
✅ 관련 답변 발견!

📄 결과 2:
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 외부로 사용자 정보가 유출될 위험이 없다는 장점을 보유
KEY Contents
£ 언어, 코드, 이미지의 3개 모델로 구성된 삼성 가우스, 온디바이스 작동 지원
n 삼성전자가 2...
✅ 관련 답변 발견!
```

- *next:* ***`셀프 쿼리 검색기 (SelfQueryRetriever)`***
