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

---

6. `MultiVectorRetriever`

1) `다중 벡터저장소 검색기`

- `MultiVectorRetriever`
  - `LangChain`에서 문서를 다양한 상황에서 효율적으로 쿼리할 수 있는 특별한 기능
  - 문서를 `여러 벡터로 저장하고 관리`할 수 있음 → 정보 검색의 **`정확도, 효율성`** **대폭 향상 가능**

- **`MultiVectorRetriever`: 문서당 여러 벡터를 생성하는 방법**
  - **`① 작은 청크 생성`**: 문서를 `더 작은 단위`로 `나눔` → 각 청크에 대해 `별도의 임베딩 생성`
    - 문서의 특정 부분에 좀 더 세심한 주의를 기울일 수 있음
    - **`ParentDocunmentRetriever`** 통해 구현 → 세부 정보에 대한 탐색 용이해짐
  - **`② 요약 임베딩`**: 각 문서의 `요약`을 `생성` → 이 요약으로부터 `임베딩 생성`
    - 문서의 핵심 내용을 신속하게 파악하는 데 큰 도움
    - 문서 전체를 분석하는 대신 **`핵심적인 요약 부분만을 활용`** → **`효율성 극대화`** **가능**
  - **`③ 가설 질문 활용`**: 각 문서에 대해 `적합한 가설 질문` 생성 → 이 질문에 기반한 `임베딩 생성`
    - `특정 주제` or `특정 내용`에 대해 깊이 있는 탐색을 원할 때 유용
    - 가설 질문 = 문서의 내용을 `다양한 관점에서 접근 가능` = `더 광범위한 이해를 가능하게 함`
  - **`④ 수동 추가 방식`**: 사용자가 문서 검색시 고려해야 할 **`특정 질문`** **or** **`특정 쿼리`** **직접 추가 가능**
    - 사용자: 검색 과정에서 보다 `세밀한 제어 가능`
    - 사용지: `자신의 요구 사항에 맞춘 맞춤형 검색 가능`

---

2) `설정`

- 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

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchai
    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: '{langc
    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 AI 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

```
# 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",              # 768
        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)        # 부모
```

- 더 큰 `Chunk` 인 `Parent` 문서 생성하기

```
parent_docs = []

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

    for _doc in parent_doc:
        _doc.metadata[id_key] = _id                                     # metadata에 문서 I
    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 가져
    child_doc = child_text_splitter.split_documents([doc])          # 현재 문서를 하위 문.

    for _doc in child_doc:
        _doc.metadata[id_key] = _id                                 # metadata에 문서 I
    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 24.3
```

- 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

```
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


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


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

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

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

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

•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

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

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

```python
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 해소하기 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 Direct
Forum and over $10 million for a new AI Safety Fund, 2023.10.25.
```

```
huggingface/tokenizers: The current process just got forked, after parallelism
To disable this warning, you can either:
    - Avoid using `tokenizers` before the fork if possible
    - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | fa
```

- **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 해소하기 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 Direct
Forum and over $10 million for a new AI Safety Fund, 2023.10.25.
```

```python
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 해소하기 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 Direct
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!
```

```
    print(type(doc))                                    # <class 'langchain_core.document
```

```
    # 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()
```

```
    # 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 poin
        ])
        | RunnableLambda(groq_function)                        # 함수를 Runnable로!
        | StrOutputParser()
    )

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

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

```
# 배치 처리
summaries = []

for doc in split_docs:
    try:
        summary = summary_chain.invoke({"doc": 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"\n 🎉 총 {len(summaries)}개 요약 완료!")
```

- 셀 출력

```
❌ 오류: 'dict' object has no attribute 'page_content'
❌ 오류: 'dict' object has no attribute 'page_content'
❌ 오류: 'dict' object has no attribute 'page_content'

    (중략...)

❌ 오류: 'dict' object has no attribute 'page_content'
❌ 오류: 'dict' object has no attribute 'page_content'
❌ 오류: 'dict' object has no attribute 'page_content'

🎉 총 61개 요약 완료!
```

```
# split_docs 데이터 구조 확인하기

print(type(split_docs[0]))                          # <class 'langchain_core.document
```

```
# 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 els
            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
    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 poin
    ])
    | 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!
```

- 요약된 내용 출력 → 결과 확인하기

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


# 요약된 내용 출력해보기
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 "Samsu
* Samsung Gauss was first introduced at the "Samsung AI Forum 2023" event on No
* Samsung plans to gradually integrate Samsung Gauss into various products, leve
```

- **`Chroma`** 벡터 저장소 초기화 → `Child chunks` 인덱싱 (이 때 **`임베딩 함수`** 사용)
  - **문서 `ID` = `doc_id`**

```
import uuid

# 요약 정보를 저장할 벡터 저장소를 생성하기
```

```
summary_vectorstore = Chroma(
    collection_name="summaries",
    embedding_function=embeddings,
)

# 부모 문서를 저장할 저장소 생성하기
store = InMemoryStore()

# 문서 ID를 저장할 키 이름 지정하기
id_key = "doc_id"

# 검색기 초기화하기
# 시작 시 비어 있음
retriever = MultiVectorRetriever(
    vectorstore=summary_vectorstore,                # 벡터 저장소
    byte_store=store,                               # 바이트 저장소
    id_key=id_key,                                  # 문서 ID 키
)
# 문서 ID 생성하기
doc_ids = [str(uuid.uuid4()) for _ in split_docs]
```

- 요약된 문서와 메타데이터 (생성한 요약본에 대한 `Document ID`) 저장하기

```
summary_docs = [
    # Document 객체 생성하기
    Document(page_content=s, metadata={id_key: doc_ids[i]})         # 요약된 내용=페
    for i, s in enumerate(summaries)
]
```

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

- **`요약본의 문서의 개수 = 원본 문서의 개수`**

```
# 요약본의 문서의 개수
len(summary_docs)                                   # 61
```

- **`retriever.vectorstor.add_documents(summary_docs)`** → **`summary_docs`** 를 벡터 저장소에 추가하기
- **`retriever.vectorstor.mset(list(zip(doc_ids, docs)))`** → **`doc_ids`**, **`docs`** 매핑 → 문서 저장소에 저장하기

```
# 요약된 문서를 벡터 저장소에 추가하기
retriever.vectorstore.add_documents(
    summary_docs
)

# 문서 ID와 문서를 매핑하여 문서 저장소에 저장하기
retriever.docstore.mset(list(zip(doc_ids, split_docs)))                 # 14.6s
```

- `vectorstore` 객체의 `similarity_search` 메서드 → 유사도 검색 수행하기

```
# 유사도 검색 수행하기

result_docs = summary_vectorstore.similarity_search(
    "삼성전자가 만든 생성형 AI 의 이름은?"
)
```

```
# 1개의 결과 문서 출력해보기

print(result_docs[0].page_content)
```

- 1개 결과 문서를 출력해보기

```
To address the request, I will provide a summary of the document "인공지능 산업 동향

* The AI industry is rapidly evolving, with advancements in machine learning, na
* Key trends in the AI industry include the increasing adoption of cloud-based A
* As the AI industry continues to expand, it is expected to have a significant
```

- `retriever` 객체의 `invoke()` → 질문과 관련된 문서 검색하기

```
# 관련된 문서를 검색하여 가져오기
retrieved_docs = retriever.invoke("삼성전자가 만든 생성형 AI 의 이름은?")
print(retrieved_docs[0].page_content)
```

- 관련된 문서 검색해서 가져오기 (1번째 페이지)
  - Ⅰ. 인공지능 산업 동향 브리프

```
# 관련된 문서를 검색하여 가져오기
retrieved_docs = retriever.invoke("삼성전자가 만든 생성형 AI 의 이름은?")
print(retrieved_docs[1].page_content)
```

- 관련된 문서 검색해서 가져오기 (2번째 페이지)

```
평가 개발과 시행 △AI 안전 연구 촉진 △정보 교류 활성화를 핵심 기능으로 함
n (첨단 AI 시스템 평가 개발과 시행) 시스템의 안전 관련 속성을 중심으로 안전과 보안 기능을 이해
하고 사회적 영향을 평가
•평가 우선순위는 △사이버범죄 조장, 허위 정보 유포 등 악의적으로 활용될 수 있는 기능 △사회에 미치는
영향 △시스템 안전과 보안 △인간의 통제력 상실 가능성 순
•연구소는 외부 기관과 협력해 자체 시스템 평가를 개발 및 수행하고, 평가와 관련된 의견 공유 및 지침
마련을 위해 전문가 커뮤니티를 소집할 계획
n (AI 안전 연구 촉진) 외부 연구자를 소집하고 다양한 예비 연구 프로젝트를 통해 AI 안전 기초연구를 수행
•AI 시스템의 효과적 거버넌스를 위한 도구 개발* 및 안전한 AI 시스템 개발을 위한 새로운 접근 방식 연구를
```

* 편향된 훈련 데이터에 대한 분석기술, 민감한 정보를 포함하는 AI 시스템에 대한 미세 조정 방법
n (정보 교류 활성화) 현행 개인정보보호와 데이터 규제 하에서 연구소와 정책입안자, 국제 파트너, 학계, 시민사회 및 일반 대중과 정보 공유 채널을 구축

---

5) `가설 쿼리를 활용한 문서 내용 탐색`

- `LLM` = 특정 문서에 대해 가정할 수 있는 질문 목록을 생성하는 데에도 사용될 수 있음
  - 이렇게 생성된 질문 → `임베딩`될 수 있음 → 문서의 내용을 더욱 깊이 있게 탐색, 이해할 수 있게 함
  - **`가정 질문 생성`**
    - 문서의 주요 주제, 개념 파악에 도움
    - 독자들이 문서 내용에 더 많은 궁금증을 갖도록 유도할 수 있음

- **`Function Calling`** 활용 → 가설 질문 생성

```
functions = [
    {
        "name": "hypothetical_questions",                                    # 함수
        "description": "Generate hypothetical questions",                    # 함수
        "parameters": {                                                      # 함수
            "type": "object",                                                # 매개
            "properties": {                                                  # 객체
                "questions": {                                               # 'qu
                    "type": "array",                                         # 'qu
                    "items": {                                               # 배열
                        "type": "string"
                    },
                },
            },
            "required": ["questions"],                                       # 필수
        },
    }
]
```

- **`ChatPromptTemplate`** → 주어진 문서를 기반으로 3개의 가상 질문을 생성하는 프롬프트 템플릿 정의하기
  - **`functions`**, **`function_call`** 설정 → 가상 질문 생성 함수 호출하기
  - **`JsonKeyOutputFunctionsParser`** → 생성된 가상 질문 파싱
  - **`questions`** → `key`에 해당하는 값 추출하기

```
import os
from dotenv import load_dotenv
from groq import Groq
```

```
# 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}],
            messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": f"Generate a list of exactly 3 hypothetic
            ],
            temperature=0.1,
            functions=functions,                                    # 함수 설정하기
            function_call={"name": "hypothetical_questions"}        # 함수 호출 설정
        )
        return response.choices[0].message.content

    return groq_invoke

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

```
from langchain_core.prompts import ChatPromptTemplate
from langchain.output_parsers.openai_functions import JsonKeyOutputFunctionsParse
from langchain_core.runnables import RunnableLambda
from groq import Groq

# 함수형으로 처리해보기
hypothetical_query_chain = (
    {"doc": lambda x: x.page_content}

    # 아래 문서를 사용하여 답변할 수 있는 가상의 질문을 정확히 3개 생성하도록 요청하기
    | ChatPromptTemplate.from_template(
        "Generate a list of exactly 3 hypothetical questions that the below docum
        "Potential users are those interested in the AI industry. Create question
        "Output should be written in Korean:\n\n{doc}"
    )

    | RunnableLambda(groq_function)

    # 출력에서 "questions" 키에 해당하는 값을 추출하기
    | JsonKeyOutputFunctionsParser(key_name="questions")
)
```

```
print("🎯 Groq으로 가상 질문 생성하기 준비 완료 !")                    # 🎯 Groq으로 가상 질문
```

↓

여기에서부터 실패 및 실습 불가

- 문서에 대한 답변 출력하기
  - 출력 = 생성한 `3개의 가설 쿼리`

```
# 주어진 문서에 대해 체인 실행해보기
hypothetical_query_chain.invoke(split_docs[33])
```

- 오류 메시지

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[47], line 2
    1 # 주어진 문서에 대해 체인 실행해보기
----> 2 hypothetical_query_chain.invoke(split_docs[33])

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
3243                 input_ = context.run(step.invoke, input_, config, **kwargs
3244             else:
-> 3245                 input_ = context.run(step.invoke, input_, config)
3246 # finish the root run
3247 except BaseException as e:

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/outpu
    199 if isinstance(input, BaseMessage):
    200     return self._call_with_config(
    201         lambda inner_input: self.parse_result(
    202             [ChatGeneration(message=inner_input)]
(...)    206         run_type="parser",
    207     )
--> 208 return self._call_with_config(
    209     lambda inner_input: self.parse_result([Generation(text=inner_input)
    210     input,
    211     config,
    212     run_type="parser",
    213 )
```

```
File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
2085     child_config = patch_config(config, callbacks=run_manager.get_child())
2086     with set_config_context(child_config) as context:
2087         output = cast(
2088             "Output",
-> 2089             context.run(
2090                 call_func_with_variable_args,  # type: ignore[arg-type]
2091                 func,
2092                 input_,
2093                 config,
2094                 run_manager,
2095                 **kwargs,
2096             ),
2097         )
2098 except BaseException as e:
2099     run_manager.on_chain_error(e)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
    428 if run_manager is not None and accepts_run_manager(func):
    429     kwargs["run_manager"] = run_manager
--> 430 return func(input, **kwargs)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/outpu
    199 if isinstance(input, BaseMessage):
    200     return self._call_with_config(
    201         lambda inner_input: self.parse_result(
    202             [ChatGeneration(message=inner_input)]
(...)    206         run_type="parser",
    207     )
    208 return self._call_with_config(
--> 209     lambda inner_input: self.parse_result([Generation(text=inner_input)
    210     input,
    211     config,
    212     run_type="parser",
    213 )

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/load/
    113 def __init__(self, *args: Any, **kwargs: Any) -> None:
    114     """"""  # noqa: D419
--> 115     super().__init__(*args, **kwargs)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/pydantic/main.py:253
    251 # `__tracebackhide__` tells pytest and some other tools to omit this fu
    252 __tracebackhide__ = True
--> 253 validated_self = self.__pydantic_validator__.validate_python(data, self_
    254 if self is not validated_self:
    255     warnings.warn(
    256         'A custom validator is returning a value other than `self`.\n'
```

```
    257         "Returning anything other than `self` from a top level model va
    258         'See the `model_validator` docs (https://docs.pydantic.dev/late
    259         stacklevel=2,
    260     )

ValidationError: 1 validation error for Generation
text
Input should be a valid string [type=string_type, input_value=None, input_type=N
    For further information visit https://errors.pydantic.dev/2.11/v/string_typ
```

- **`chain.batch`** 메서드 → **`split_docs`** 데이터에 대해 동시에 여러 개의 요청을 처리

```
# 문서 목록에 대해 가설 질문을 배치 생성

hypothetical_questions = hypothetical_query_chain.batch(
    split_docs, {"max_concurrency": 10}
)                                                          # m s
```

- 오류 메시지 (1m 35.5s)

```
---------------------------------------------------------------------------
RateLimitError                            Traceback (most recent call last)
Cell In[56], line 3
    1 # 문서 목록에 대해 가설 질문을 배치 생성
----> 3 hypothetical_questions = hypothetical_query_chain.batch(
    4     split_docs, {"max_concurrency": 10}
    5 )

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnab
3397     else:
3398         for i, step in enumerate(self.steps):
-> 3399             inputs = step.batch(
3400                 inputs,
3401                 [
3402                     # each step a child run of the corresponding root run
3403                     patch_config(
3404                         config, callbacks=rm.get_child(f"seq:step:{i + 1}"
3405                     )
3406                     for rm, config in zip(run_managers, configs)
3407                 ],
3408                 return_exceptions=return_exceptions,
3409                 **(kwargs if i == 0 else {}),
3410             )
3412 # finish the root runs
3413 except BaseException as e:

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnab
```

```
    917     return cast("list[Output]", [invoke(inputs[0], configs[0])])
    919 with get_executor_for_config(configs[0]) as executor:
--> 920     return cast("list[Output]", list(executor.map(invoke, inputs, confi

File ~/.pyenv/versions/3.13.5/lib/python3.13/concurrent/futures/_base.py:619, i
    616 while fs:
    617     # Careful not to keep a reference to the popped future
    618     if timeout is None:
--> 619         yield _result_or_cancel(fs.pop())
    620     else:
    621         yield _result_or_cancel(fs.pop(), end_time - time.monotonic())

File ~/.pyenv/versions/3.13.5/lib/python3.13/concurrent/futures/_base.py:317, i
    315 try:
    316     try:
--> 317         return fut.result(timeout)
    318     finally:
    319         fut.cancel()

File ~/.pyenv/versions/3.13.5/lib/python3.13/concurrent/futures/_base.py:449, i
    447     raise CancelledError()
    448 elif self._state == FINISHED:
--> 449     return self.__get_result()
    451 self._condition.wait(timeout)
    453 if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

File ~/.pyenv/versions/3.13.5/lib/python3.13/concurrent/futures/_base.py:401, i
    399 if self._exception is not None:
    400     try:
--> 401         raise self._exception
    402     finally:
    403         # Break a reference cycle with the exception in self._exception
    404         self = None

File ~/.pyenv/versions/3.13.5/lib/python3.13/concurrent/futures/thread.py:59, i
    56     return
    58 try:
---> 59     result = self.fn(*self.args, **self.kwargs)
    60 except BaseException as exc:
    61     self.future.set_exception(exc)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runna
    552 def _wrapped_fn(*args: Any) -> T:
--> 553     return contexts.pop().run(fn, *args)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runna
    911         return e
    912 else:
```

```
--> 913     return self.invoke(input_, config, **kwargs)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
5009 """Invoke this ``Runnable`` synchronously.
5010
5011 Args:
(...)   5021
5022 """
5023 if hasattr(self, "func"):
-> 5024     return self._call_with_config(
5025         self._invoke,
5026         input,
5027         ensure_config(config),
5028         **kwargs,
5029     )
5030 msg = "Cannot invoke a coroutine function synchronously.Use `ainvoke` inst
5031 raise TypeError(msg)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
2085     child_config = patch_config(config, callbacks=run_manager.get_child())
2086     with set_config_context(child_config) as context:
2087         output = cast(
2088             "Output",
-> 2089             context.run(
2090                 call_func_with_variable_args,  # type: ignore[arg-type]
2091                 func,
2092                 input_,
2093                 config,
2094                 run_manager,
2095                 **kwargs,
2096             ),
2097         )
2098 except BaseException as e:
2099     run_manager.on_chain_error(e)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
    428 if run_manager is not None and accepts_run_manager(func):
    429     kwargs["run_manager"] = run_manager
--> 430 return func(input, **kwargs)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
4879                 output = chunk
4880 else:
-> 4881     output = call_func_with_variable_args(
4882         self.func, input_, config, run_manager, **kwargs
4883     )
4884 # If the output is a Runnable, invoke it
4885 if isinstance(output, Runnable):
```

```
File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/runnal
    428 if run_manager is not None and accepts_run_manager(func):
    429     kwargs["run_manager"] = run_manager
--> 430 return func(input, **kwargs)

Cell In[43], line 21, in create_groq_llm.<locals>.groq_invoke(prompt)
    18 else:
    19     prompt_text = str(prompt)
---> 21 response = client.chat.completions.create(
    22     model=model,
    23     # messages=[{"role": "user", "content": prompt_text}],
    24     messages=[
    25     {"role": "system", "content": "You are a helpful assistant."},
    26     {"role": "user", "content": f"Generate a list of exactly 3 hypotheti
    27     ],
    28     temperature=0.1,
    29     functions=functions,                                    # 함수 설정하기
    30     function_call={"name": "hypothetical_questions"}        # 함수 호출 설정
    31 )
    32 return response.choices[0].message.content

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/groq/resources/chat/
    241 def create(
    242     self,
    243     *,
(...)    299     timeout: float | httpx.Timeout | None | NotGiven = not_given,
    300 ) -> ChatCompletion | Stream[ChatCompletionChunk]:
    301     """
    302     Creates a model response for the given chat conversation.
    303
(...)    454       timeout: Override the client-level default timeout for this
    455     """
--> 456     return self._post(
    457         "/openai/v1/chat/completions",
    458         body=maybe_transform(
    459             {
    460                 "messages": messages,
    461                 "model": model,
    462                 "compound_custom": compound_custom,
    463                 "disable_tool_validation": disable_tool_validation,
    464                 "documents": documents,
    465                 "exclude_domains": exclude_domains,
    466                 "frequency_penalty": frequency_penalty,
    467                 "function_call": function_call,
    468                 "functions": functions,
    469                 "include_domains": include_domains,
    470                 "include_reasoning": include_reasoning,
```

```
    471                 "logit_bias": logit_bias,
    472                 "logprobs": logprobs,
    473                 "max_completion_tokens": max_completion_tokens,
    474                 "max_tokens": max_tokens,
    475                 "metadata": metadata,
    476                 "n": n,
    477                 "parallel_tool_calls": parallel_tool_calls,
    478                 "presence_penalty": presence_penalty,
    479                 "reasoning_effort": reasoning_effort,
    480                 "reasoning_format": reasoning_format,
    481                 "response_format": response_format,
    482                 "search_settings": search_settings,
    483                 "seed": seed,
    484                 "service_tier": service_tier,
    485                 "stop": stop,
    486                 "store": store,
    487                 "stream": stream,
    488                 "temperature": temperature,
    489                 "tool_choice": tool_choice,
    490                 "tools": tools,
    491                 "top_logprobs": top_logprobs,
    492                 "top_p": top_p,
    493                 "user": user,
    494             },
    495             completion_create_params.CompletionCreateParams,
    496         ),
    497         options=make_request_options(
    498             extra_headers=extra_headers, extra_query=extra_query, extra_
    499         ),
    500         cast_to=ChatCompletion,
    501         stream=stream or False,
    502         stream_cls=Stream[ChatCompletionChunk],
    503     )

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/groq/_base_client.py
1228 def post(
1229     self,
1230     path: str,
(...)   1237     stream_cls: type[_StreamT] | None = None,
1238 ) -> ResponseT | _StreamT:
1239     opts = FinalRequestOptions.construct(
1240         method="post", url=path, json_data=body, files=to_httpx_files(files
1241     )
-> 1242     return cast(ResponseT, self.request(cast_to, opts, stream=stream, st

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/groq/_base_client.py
1041             err.response.read()
1043         log.debug("Re-raising status error")
```

```
-> 1044         raise self._make_status_error_from_response(err.response) from N
1046     break
1048 assert response is not None, "could not resolve response (should never hap

RateLimitError: Error code: 429 - {'error': {'message': 'Rate limit reached for
```

- 가설 쿼리를 벡터 저장소에 저장하기

```
# 자식 청크를 인덱싱하는 데 사용할 벡터 저장소
hypothetical_vectorstore = Chroma(
    collection_name="hypo-questions",
    embedding_function=embeddings,
)
# 부모 문서의 저장소 계층
store = InMemoryStore()

id_key = "doc_id"

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

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

- **`question_docs`** 리스트에 메타데이터 (`문서 ID`) 추가하기

```
question_docs = []

# hypothetical_questions 저장
for i, question_list in enumerate(hypothetical_questions):
    question_docs.extend(
        # 질문 리스트의 각 질문에 대해 Document 객체를 생성하고, 메타데이터에 해당 질문의 문서 ID를
        [Document(page_content=s, metadata={id_key: doc_ids[i]}) for s in questic
    )
```

- 문서 = 가설 쿼리 추가
- **`docstore`** = 원본 문서 추가

```
# hypothetical_questions 문서를 벡터 저장소에 추가하기
retriever.vectorstore.add_documents(question_docs)

# 문서 ID와 문서를 매핑하여 문서 저장소에 저장하기
retriever.docstore.mset(list(zip(doc_ids, split_docs)))
```

- **`vectorstore`** 객체의 **`similarity_search`** 메서드 → 유사도 검색 수행

```
# 유사한 문서를 벡터 저장소에서 검색하기

result_docs = hypothetical_vectorstore.similarity_search(
    "삼성전자가 만든 생성형 AI 의 이름은?"
)
```

- 유사도 검색 결과
    - 생성한 가설 쿼리만 추가해둔 상태 → **`생성 가설 쿼리 중 유사도가 가장 높은 문서 반환`**

```
# 유사도 검색 결과 출력하기

for doc in result_docs:
    print(doc.page_content)
    print(doc.metadata)
```

- **`retriever`** 객체의 **`invoke()`** → 쿼리와 관련된 문서 검색하기

```
# 관련된 문서를 검색하여 가져오기
retrieved_docs = retriever.invoke(result_docs[1].page_content)

# 검색된 문서 출력해보기
for doc in retrieved_docs:
    print(doc.page_content)
```

- *next:* ***`가설 쿼리 활용해 문서 내용 탐색하기 및 retriever 성능 검토하기`***
