섹션 5: 고급 검색기법

개선된 검색 기법의 중요성

- 정확성과 관련성 향상
- 검색 결과의 다양성 확보
- 계산 비용 최적화

실제 적용 시 고려사항

- 데이터의 특성과 규모
- 검색 속도와 정확도의 균형
- 사용 가능한 컴퓨팅 리소스

각 기법의 장단점 요약

기법	장점	단점
벡터저장소 검색	빠른 검색 속도	정확도에 한계
Query Translation	다양한 관점 확보 가능	계산 비용 증가
Re-rank	정확도 향상	추가 처리 시간 필요
Contextual compression	관련 정보 집중	정보 손실 가능성

1. 쿼리(Query) 확장

- 사용자의 원래 쿼리를 확장하여 더 관련성 있는 결과를 얻는 기술

Multi Query

기본 개념

다양한 관점에서 여러 개의 쿼리 생성

장점: 다양한 관점의 쿼리로 검색 범위 확장

구현 방법

MultiQueryRetriever 활용

```
multi_query_retriever = MultiQueryRetriever.from_llm(retriever=chroma_k, llm=llm)
```

Custom Prompt 활용

QUERY_PROMPT 템플릿 사용, LineListOutputParser를 통한 결과 파싱

Decomposition

기본 개념

입력된 질문을 여러 개의 하위 질문으로 분해

장점: 복잡한 질문을 단순화하여 정확한 정보 검색 가능

구현 방법

LEAST-TO-MOST PROMPTING 기법 활용

```
decomposition_chain = QUERY_PROMPT | llm | LineListOutputParser()
```

공통점

- 복잡한 질문 처리
- LLM 활용
- LineListOutputParser 사용

1.1 Multi Query

- 하나의 원본 쿼리로부터 여러 개의 관련 쿼리를 생성
- 검색의 범위와 정확도를 높이는 방법
- 작동 원리:
  - Retriever에 쿼리를 생성할 LLM(Large Language Model)을 지정
  - 지정된 LLM이 원본 쿼리를 분석하고, 다양한 관점에서 여러 개의 관련 쿼리를 생성
  - 생성된 모든 쿼리를 사용하여 검색을 수행
- 장점:
  - 검색의 다양성 증가: 다양한 측면에서 정보를 검색
  - 누락된 정보 감소: 하나의 쿼리로는 찾기 어려운 정보도 포착
  - 컨텍스트 확장: 원본 질문의 맥락을 더 넓게 고려

1.2 Decomposition (LEAST-TO-MOST PROMPTING)

- 복잡한 질문을 여러 개의 간단한 하위 질문으로 분해하여 처리

- 작동 원리:
    - 입력된 복잡한 질문을 LLM을 사용하여 여러 개의 하위 질문으로 분해
    - 각 하위 질문에 대해 순차적으로 답변을 생성
    - 생성된 답변들을 종합하여 최종 답변을 생성
- 장점:
    - 복잡한 문제 해결: 단계별로 접근하여 복잡한 질문도 효과적으로 다룰 수 있음
    - 정확성 향상: 각 하위 문제를 개별적으로 처리하여 오류 가능성을 줄임
    - 설명 가능성: 각 단계의 결과를 볼 수 있어 답변 과정을 추적 가능

2. Re-rank (재순위화)

- 초기 검색 결과를 다시 순위를 매기는 과정
- 더 정확하고 관련성 높은 결과를 상위로 올리는 것이 목적

Cross Encoder Reranker

기본 개념

Hugging Face의 Cross-Encoder 모델 사용, 검색 쿼리와 검색된 문서 간의 유사성 계산

장점: 정밀한 유사도 계산으로 검색 결과의 품질 향상

구현 방법

Cross-Encoder 모델 사용

```
cross_encoder_reranker 함수 구현 (model='BAAI/bge-reranker-v2-m3')
```

LLM Reranker

기본 개념

LLM을 사용한 검색 결과 재정렬, LLMListwiseRerank 등의 기법 활용

장점: LLM의 이해력을 활용한 고도화된 재정렬 가능

구현 방법

LLMListwiseRerank 활용

```
re_ranker = LLMListwiseRerank.from_llm(llm, top_n=3)
```

2.1 Cross Encoder Reranker 사용

- Cross Encoder는 쿼리와 문서 쌍의 관련성을 직접 평가하는 강력한 모델
- 쿼리와 문서 간의 직접적인 관련성을 평가하여 높은 정확도를 제공

- 작동 원리:
    - 쿼리와 각 문서를 쌍으로 입력
    - 모델이 이 쌍을 동시에 처리하여 관련성 점수를 출력
    - 점수에 따라 문서들을 재정렬
- 장점:
    - 높은 정확도: 쿼리와 문서를 함께 고려하여 더 정확한 관련성 평가
    - 컨텍스트 이해: 쿼리와 문서 간의 복잡한 관계를 포착

Bi-Encoder

각 문장을 별도로 인코딩, 빠르지만 정확도 낮음

- 독립적 문장 인코딩
- 빠른 처리 속도
- 상대적으로 낮은 정확도

장단점

- 대규모 데이터셋 처리에 효율적
- 문맥 이해와 미묘한 차이 포착에 한계

Cross-Encoder

두 문장을 함께 인코딩, 느리지만 정확도 높음

- 문장 쌍 동시 인코딩
- 상대적으로 느린 처리 속도
- 높은 정확도

장단점

- 문맥 이해와 미묘한 의미 차이 포착 우수
- 대규모 데이터셋 처리에 시간 소요

실제 응용

Bi-Encoder로 대략적인 후보군을 빠르게 선별한 후, Cross-Encoder로 최종 순위를 정확하게 결정하는 하이브리드 접근 방식을 통해 각 방식의 장점을 극대화할 수 있습니다.

RAG 시스템에서의 활용

1. 초기 검색 — Bi-Encoder
2. 재순위화 — Cross-Encoder
3. 응답 생성 — LLM

Bi-Encoder in RAG

- 대규모 문서 컬렉션에서 빠른 초기 검색
- 임베딩 인덱스 생성 및 유지
- 실시간 쿼리 처리에 적합

Cross-Encoder in RAG

- Bi-Encoder로 검색된 상위 후보 재순위화
- 쿼리-문서 쌍의 정확한 관련성 평가
- 최종 컨텍스트 선택 정확도 향상

2.2 LLM Reranker사용

- LLM을 사용한 재순위화는 모델의 풍부한 지식과 추론 능력을 활용하여 검색 결과를 개선
- 더 넓은 맥락과 복잡한 관계를 이해할 수 있어, 특히 추론이 필요한 상황에서 유용

- 작동 원리:
    - 초기 검색 결과를 LLM에 입력으로 제공
    - LLM이 각 문서의 관련성을 평가하고 순위를 조정
    - 조정된 순위에 따라 문서를 재정렬
- 장점:
    - 맥락 이해: LLM의 깊은 언어 이해 능력을 활용
    - 유연성: 다양한 기준으로 재순위화가 가능
    - 설명 가능성: LLM이 순위 조정 이유를 설명할 수 있음

3. Contextual compression (맥락 압축)

- 검색된 문서를 쿼리의 맥락에 맞게 압축하거나 필터링하는 기법
- 긴 문서에서 핵심 정보만을 효과적으로 활용하는 효과

- LLM에 전달되는 정보의 양을 줄이고, 관련성 높은 정보만을 제공하여 성능을 개선

- 주요 구성 요소:
    1. 기본 검색기(Base Retriever): 초기 문서 검색을 수행
    2. 문서 압축기(Document Compressor): 검색된 문서를 쿼리에 맞게 압축 or 필터링

- 기대 효과:
    1. 관련성 향상: 쿼리와 직접적으로 관련된 정보만을 제공
    2. 효율성 증대: LLM에 전달되는 데이터량을 줄여 처리 속도를 높이고 비용을 절감
    3. 응답 품질 개선: 불필요한 정보를 제거하여 더 정확하고 집중된 응답을 생성

LLMChainFilter

LLM을 사용한 문서 필터링

```
context_filter = LLMChainFilter.from_llm(llm)
```

장점: 관련성 높은 문서만 선별적으로 추출

LLMChainExtractor

LLM을 사용한 관련 내용 추출 및 요약

```
compressor = LLMChainExtractor.from_llm(llm)
```

장점: 문서에서 쿼리와 관련된 핵심 정보만 추출

EmbeddingsFilter

임베딩 기반의 문서 필터링

```
embeddings_filter = EmbeddingsFilter(
  embeddings=embeddings_model,
  similarity_threshold=0.65
)
```

장점: LLM 호출 없이 빠르고 효율적인 필터링 가능

DocumentCompressorPipeline

여러 압축기를 순차적으로 결합

```
pipeline_compressor = DocumentCompressorPipeline(
  transformers=[
    redundant_filter,
    relevant_filter,
    re_ranker
  ]
)
```

장점: 다양한 압축 기법을 조합하여 최적의 결과 도출

3.1 LLMChainFilter

- LLM을 사용하여 검색된 문서 중 관련성 높은 문서만을 선택
- 문서 내용을 압축하거나 변경하지 않음 (문서 내용은 그대로 유지)

- 작동 원리:
    - 각 문서에 대해 LLM에게 관련성을 평가하도록 요청
    - LLM의 판단에 따라 문서를 유지하거나 제거

3.2 LLMChainExtractor

- LLM을 사용하여 각 문서에서 쿼리와 관련된 부분만을 추출

- 작동 원리:
    - 각 문서를 LLM에 입력으로 제공
    - LLM이 쿼리와 관련된 내용만을 추출하여 요약

3.3 EmbeddingsFilter

- 임베딩을 사용하여 문서를 필터링
- LLM을 사용하지 않아 빠르고 비용 효율적

- 작동 원리:
    - 쿼리와 문서를 벡터로 임베딩
    - 쿼리 벡터와 문서 벡터 간의 유사도를 계산
    - 유사도가 임계값을 넘는 문서만 유지

3.4 DocumentCompressorPipeline

- DocumentCompressorPipeline을 사용하면 여러 압축 기법을 순차적으로 적용 가능

- 작동 원리:
    - 여러 압축기를 순서대로 연결
    - 각 압축기의 출력이 다음 압축기의 입력으로 전