Solutions

## **TA's 솔루션 해설**

* <https://gist.github.com/ICE0208/8d296c5aabecc9a62fc4c9610d8dc493>

1. ChatOpenAI 모델 초기화 및 설정
   * ChatOpenAI 생성자를 이용하여 OpenAI의 GPT 모델을 초기화합니다.
   * 모델 초기화 시 몇몇 설정을 커스텀할 수 있습니다.
   * 해설 코드에서는 temperature를 0.1로 설정하였습니다.
2. 문서 로드와 쪼개기 / 임베딩 생성 및 캐시
   * TextLoader를 이용하여 텍스트 문서를 로드합니다.
   * CharacterTextSplitter를 사용하여 텍스트를 일정한 크기의 조각으로 나눕니다.
   * OpenAIEmbeddings를 사용하여 텍스트 조각들에 대한 임베딩을 생성합니다.
   * CacheBackedEmbeddings를 이용하여 임베딩을 캐시하여 성능을 향상시킵니다.
   * FAISS를 이용하여 임베딩 벡터들을 저장하고, 효율적으로 검색할 수 있도록 합니다.
3. 메모리 초기화
   * ConversationBufferMemory를 사용하여 대화 기록을 저장하고 관리합니다.
   * return\_messages=True로 설정하여 메모리를 채팅 형식으로 로드할 수 있도록 합니다.
4. 메모리 로드 함수 정의
   * 메모리를 로드하여 대화 기록을 반환하는 함수를 정의합니다.
   * 이 함수는 체인에서 대화 기록을 로드하는 데 사용됩니다.
5. 검색기 설정
   * vectorstore.as\_retriever()를 사용하여 벡터 저장소를 검색기로 설정합니다.
6. 프롬프트 생성
   * ChatPromptTemplate.from\_messages를 이용하여 대화 메시지 템플릿을 생성합니다.
   * 프롬프트는 시스템 메시지, 대화 기록 플레이스홀더, 사용자 질문으로 구성됩니다.
   * 시스템 메시지는 모델에게 주어진 컨텍스트 내에서만 답변하도록 지시합니다.
7. 체인 설정 및 체인 호출 함수 정의
   * RunnablePassthrough와 메모리 로드 함수, 검색기(retriever)를 사용하여 체인을 구성합니다.
   * 체인은 컨텍스트를 검색하고, 사용자 질문과 대화 기록을 결합하여 최종 프롬프트를 생성한 후, LLM을 호출합니다.
   * invoke\_chain 함수를 정의하여 이 함수를 실행하였을 때, 체인을 실행하고 그 결과를 바로 메모리에 저장하도록 합니다.
8. 체인 테스트
   * invoke\_chain 함수를 통해 예시 질문을 입력받아 체인을 실행합니다.
   * 예시 질문에 대한 답변이 정상적으로 반환되는지 확인합니다.
9. 메모리 확인
   * load\_memory 함수를 호출하여 메모리에 저장된 대화 기록이 체인에 적용되었는지 확인합니다.
10. 결론

* 이번 챌린지는 Stuff Documents 체인을 이용하여 완전한 RAG 파이프라인을 수동으로 구현하는 연습을 해보는 것이었습니다. 또한, 체인에 ConversationBufferMemory를 추가하여 대화 기록을 관리하는 방법을 복습했습니다. 이를 통해, 이전에 배운 메모리 관리 기법과 새롭게 배운 RAG 파이프라인의 수동 구현을 결합함으로써 학습 내용을 더욱 잘 이해하고 응용할 수 있었습니다.