랭체인 week5

요약

랭체인 업데이트

구버전으로 진행해야 프로젝트 코드 동작함

- 구버전

모듈 구조 직접 변경 불가능

개발자들이 직접 완제품 삭제하고 커스텀 시작

- 신버전

runnable로 직접 커스터마이지 가능하게 수정

메모리는 체인 외부에서 내부로 주입하는 구조로 변경 (원래는 체인 내부에서 이루어짐)

ConservationBufferMemory

대화 모든 메모리 바로 저장

기타 기능 없이 그대로 보관, 저장

크기 커지면 프롬프트 그대로 길어짐

ConversationBufferWindowMemory

최근 n개의 대화만 저장

토큰 절약할 수 있지만 장기기억에 문제 있음

llm에 들어가는 형태는 딕셔너리 불가

리스트 형태로 변환해야만 적용 가능

pomrpt | llm 이 리스트로 변환하는 역할

ConversationTokenBufferMemory

대화 수가 아닌, 토큰 수 기준으로 최근 대화내용 저장

openai API 비용 최적화시 유용

대화가 언제 잘릴지 사람이 예측하기 힘듦

ConversationEntityMemory

엔티티에 대한 정보를 추출하여 기억

사람, 사물 등에 대한 추상정보를 키워드로 기억

필요한 정보만 기억시킬 수 있음

llm이 잘못된 정보 추출할 위험

ConversationKGMemory

대화 내용을 지식 그래프 형태로 저장 (주어, 동사, 목적어등 구조적 관계 추출)

지식 구조적으로 작성 가능

처음 구조 설계가 어려움

최신버전에서 KG 메모리 단점 → 구버전에서는 고정이었으나, 신버전에서 사용자가 직접 일일히 커스텀 해줘야 함

ConversationSummaryMemory

llm이 직접 요약해서 내용 관리

전체적인 맥락 유지하면서 토큰 관리 가능

요약 과정에서 세부 정보 손실 가능성 있음

ConversationSummaryBufferMemory

최근 k개 문장을 요약하여 관리 (너무 길어질것을 대비함)

토큰이 설정한거 이상으로 들어가면 요약에 포함안함

VectorStoreRetrieverMemory

대화 내용을 DB에 저장하고 필요할 때 검색해옴

검색 기반, 오래된 내용도 필요하면 불러오기 가능

대화 내용 길어저도 기존 맥락 유지 가능

벡터함수 정확도에 의존함

LCEL 메모리 사용

랭체인에서 체인 내부에 메모리를 끼워넣는 방식

pompt | llm 사이에 history라는 메모리 넣어서 사용

디버깅 쉬우나 러닝커브 높음

sql 메모리

sqlite에 대화 내용 저장

영속성 가능, 테이블 스키마 필요

Q&A

1. rag와 sqlite에 저장해서 가져오는 것의 차이점

- rag는 vecotor 저장된것중에서 연결된걸 가져오는 기술이고, 벡터 메모리는 대화 내용을 백터 형태로 저장하는 기술임. 기술적으로는 구분되는게 맞으나 상호 의존적으로 보는게 맞을거 같음

2. summary 생성 시점

- 새로운거 대화 내용 들어올때마다 요약 새로 생성, 그럼 한번 대화 보낼때마다 토큰을 작업을 2번씩 하는건데 효율적이라고 볼 수 있을지 의문이지만 새로운 기술에서 이런거 최적화 할 방법이 있을거라고 생각됨

3. TOkenbuffer에서 토큰 길이를 정하게 되면 저장하는 방식이 old한걸 삭제하는 방식인지 요약해서 저장하는 방식인가

- old 한거 삭제하는 방식

4. k개 저장시 대화 내용 길어지면 프롬프트 길어질텐데 괜찮은걸까?

- 별 문제 없을거 같음