〈PROJECT 최종보고서〉

Retrieval-Augmented Generation 기반의 Research Assistant 개발

김상범, 손동현, 이재영

엄현상 교수님

박성준 대표님 (Softly AI, CEO)

Table of Contents

1. Abstract

본 프로젝트는 급변하는 NLP 학계에서, entry-level의 NLP 연구자들이 빠르게 새로운 연구를 이해할 수 있도록 보조해주는 Research Assistant chatbot을 연구 및 개발하는 것을 목표로 삼았다. 이를 위해, 사전에 답변의 근거를 제공해줄 수 있는 문서를 업로드하고, 유저의 질의와 연관된 문서(passage)를 참조하여 답변을 생성는 Retrieval-Augmented Generation을 기반으로 접근하였다. RAG 파이프라인을 이용하여, 구현한 파이프라인에 유저가 업로드한 문서와 입력한 질의를 바탕으로 답변을 생성해주는 웹 애플리케이션을 구현하였다. 이와 함께 Research Assistant 도메인에서 기존의 RAG 파이프라인 및 구성요소 모델들의 성능 한계를 극복하기 위한 연구를 수행하고, 이를 파이프라인 구현에 반영하여 답변의 질을 개선한다. 이 프로젝트를 통해 프로젝트 참여자들은 ML 기반의 웹 애플리케이션 개발 및 서빙에 필요한 엔지니어링부터 LLM 및 RAG 연구까지 폭넓은 역량을 연마할 수 있으며, 최종적으로 완성된 애플리케이션은 연구자의 입장에서 새로운 분야를 빠르게 파악하는 데에 필요한 시간을 단축시킬 것이다.

2. Introduction

LLM, Diffusion Model들을 필두로 생성형 AI의 출현으로, 학계, 정보 산업, 제조업, 교육 등 모든 분야는 새로운 시대를 맞이했다고 해도 과언이 아니다. 문학, 예술과 같이 창의성이 핵심적이라고 여겨졌던 분야들에서도 큰 변화를 맞이하고 있다. 연구자, 개발자들과 같은 지식 노동자들도 마찬가지다. 그 중에서도 자연어 처리(NLP) 분야는 LLM의 성공으로 많은 재정적 지원과 기반 기술의 발달로 성과가 빠르게 발표되고 있다. 때문에 그 어느 때보다 논문들이 빠르게 있는 이 시점에, NLP 연구에 진입한 entry-level 연구자들은 기존 연구의 흐름을 이해하고, 지식을 습득하는 것에 어려움을 겪을 수 밖에 없다.

이러한 entry-level의 NLP 연구자들을 위해, 대화형 인터페이스의 연구 보조자(Research Assistant) 웹 애플리케이션을 개발하였다. 연구자들은 지식 습득에 필요한 질의를 하고, 그에 대한 답을 빠르게 제공받을 수 있다. 즉, Research Assistant는 사용자가 특정 연구를 자세히 이해하는 데에도 도움을 줄 수 있고, 여러 연구들을 빠르게 훑어보는 데에도 도움을 줄 수 있다.

Research Assistant를 대화형으로 구현하기 위해서는 LLM을 사용하는 것이 자연스러운 선택이다. 최근 Meta AI에서 공개한 Llama2 모델은 아키텍처뿐만 아니라 모델 weight가 공개되어 있기 때문에, 라이선스가 금지하는 일부 용도를 제외하고는 누구나 사용할 수 있다. 특히 Reinforcement Learning from Human Feedback(RLHF)를 통해 대화에서 사람들이 선호하는 답변을 얻는 데에 알맞게 조정된 Llama2-chat-hf 모델들이 함께 공개되어있기 때문에, 누구나 컴퓨팅 리소스와 지식이 있다면 LLM을 이용한 대화형 시스템을 구축할 수 있게 되었다.

하지만 LLM 기반의 도구를 있는 그대로 사용하는 경우, LLM이 학습한 시점 이후의 지식에 대해서는 알 수 없는 지식 단절(knowledge cutoff) 및 LLM이 잘못된 정보를 그럴듯하게 지어내서 생성하는 환각(hallucination) 문제로 인해 원하는 답을 얻기 어렵다. 시중에서 접할 수 있는 최고 성능의 LLM 중 하나인 OpenAI의 ChatGPT에게 질의하는 경우에도 같은 문제가 있다. ChatGPT의 knowledge

cutoff 이후에 출판된 최신 논문에 대해서 질의하면 알 수가 없다는 문제는 아무리 모델 성능이 높아져도 해결할 수 없는 근본적인 문제다. 환각의 경우 실제로 프로젝트 참여자들 또한 논문을 읽다가 ChatGPT에게 질의한 경우에 중요한 디테일을 마음대로 지어낸 답변을 얻은 경험이 있다.

그래서 RAG(Retrieval-Augmented Generation)을 이용해서 연구자가 참고하고자 하는 논문들을 사전에 업로드받고, 여기에 담긴 지식을 기반으로 LLM을 레버리지하는 방법을 선택하였다. RAG는 주어진 질의를 답변하는 데에 도움이 되는 텍스트들을 데이터베이스로부터 불러와서 답변 생성에 활용하는 방법론이다. 도움이 되는 텍스트를 판별하기 위해서는 임베딩 모델을 이용해서 질의와 텍스트의 임베딩을 추출하고 그 거리(L2, cosine distance 등)가 가까운 텍스트를 선택한다. 이를 새로운 질의에 대해 빠르게 수행하기 위해서는 사전에 업로드된 논문에 들어있는 텍스트의 임베딩을 추출해서 Vector DB에 삽입하고 인덱스를 구축해두어야 한다.

이 보고서에서는 우선 문제 해결을 위해 사용되는 관련 기술 및 문제 정의, 그리고 우리의 접근 방법에 대해 서술하고, 아키텍처 및 우리가 구현한 웹 애플리케이션의 디테일과 겪었던 기술적인 이슈들을 서술한다. 특히 본 프로젝트는 웹 애플리케이션 구현뿐만 아니라 LLM 분야의 최신 연구 성과를 반영하고 추가로 탐구하는 것을 목적으로 했기 때문에, 연구 측면에서 수행한 부분과 겪었던 이슈들에 대해서도 별도로 서술할 것이다. 끝으로, 부록 A의 유저 매뉴얼에서는 프로젝트 결과물을 사용하는 유저 입장에서의 사용 방법을, 부록 B에는 최종적으로 사용했던 프롬프트들을 수록하였다.

3. Background Study

A. 관련 접근방법/기술 장단점 분석

본 프로젝트의 핵심인 RAG 관련 접근방법에 대해서는 좀 더 깊이 있는 논의를 위해 Research Issues & Details 절 (8.C절)에서 별도로 다룰것이다. 여기에서는 Research Assistant 웹 애플리케이션을 구현하고 서빙하기 위해 필요한 기술을 중심으로 서술한다.

LLM을 서빙하기 위해서는 Huggingface에서 제공하는 Text Generation Inference(TGI)를 선택했다. TGI에는 효과적인 LLM 추론을 위한 최신 연구성과들이 반영되어있다. 운영체제에서 메모리 페이지를 관리하는 방식으로부터 아이디어를 차용해서 Attention에 필요한 대규모의 GPU 메모리를 효율적으로 관리하는 PagedAttention, batch 중에서 일부가 먼저 생성이 끝난 경우에 새로운 추론 요청을 기존에 추론중이던 batch에 끼워서 처리함으로써 throughput을 높이는 Continuous Batching 등의 기법이 적용되어있다. 또한 TGI는 Huggingface에서 제공하는 프로덕션 수준의 서비스에서도 사용되는 코드베이스로, 안정성 또한 검증되었다고 판단하였다. 그 외의 방법으로는 Huggingface transformers를 이용해서 직접 추론 서버를 구현할 수 있지만, 성능과 안정성 모두 TGI를 넘기 어려울 것이며, 특히 기존의 모델들과 달리 LLM 분야에서는 국소적인 모델 아키텍처 변경보다는 목적에 맞는 학습 데이터를 이용해서 foundation 모델을 튜닝하고 프롬프트를 잘 설계하는 것이 더 높은 성과로 이어지는 경향이 있기 때문에, 커스텀 모델 서빙이 가능하다는 이점 커스텀 서버의 이점 또한 필요가 없을 것이라고 판단했다.

백엔드 개발을 위해서는 Python 언어와 FastAPI 프레임워크를 선택했다. 프로덕션 시스템의 경우 Java나 Kotlin 언어와 Spring 프레임워크가 많이 사용되지만, Spring의 강력한 의존성 분리나 플러그인 생태계를 레버리지할 만큼 프로젝트 규모가 크지 않고, 오히려 Python 생태계의 ML 관련 라이브러리들을 사용하기 어려운 점이 문제가 될 것이라고 판단했다. 특히 본 프로젝트의 백엔드는 통상적인 관계형 데이터베이스가 아닌 Vector DB를 사용하기 때문에, ORM이 제공해주는 장점도 희석된다. 같은 이유로 Python 프레임워크 중에서는 Django를 배제했다. Flask 또한 Python으로 API 서버를 개발할 때 많이 사용되는 선택지지만, asyncio를 지원하지 않기 때문에 배제하였다. 본 프로젝트의 백엔드 서버가 많은 수의 요청을 받는다면, 병목 지점은 ML 모델들에 요청을 보내는 부분이지 백엔드에서 필요한 CPU 연산이 아니다. 때문에 I/O가 병목인 서버를 고성능으로 작성할 수 있는 asyncio 기반의 비동기 프레임워크인 FastAPI를 선택했다. 마찬가지로 임베딩 모델을 서빙하기 위한 서버도 GPU 연산이 병목이기 때문에, 동일하게 FastAPI를 사용해서 구현했다.

프론트엔드 개발을 위해서는 다양한 기술 스택이 경쟁하고 있다. 특히 Javascript 진영이 높은 점유율을 보이며, 그 중에서도 React가 주로 사용된다. React는 가상 DOM을 도입하여 동적인 컨텐츠를 포함하는 웹 애플리케이션을 개발할 때 업데이트된 컨텐츠를 화면에 반영하기 위해 필요한 부분적인 변경 사항만을 업데이트하는 방식으로 고성능의 웹 애플리케이션을 개발할 수 있는 프레임워크다. 특히 Node.js 기반의 서버를 사용하는 경우, 서버와 백엔드의 기술 스택을 Javascript 기반으로 동일하게 관리할 수 있다는 장점을 더한다. 동일한 이유로, 우리는 Python을 이용해서 프론트엔드를 개발할 수 있는 프레임워크를 우선적으로 탐색했다. 화면에서 갱신되어야 하는 동적인 컨텐츠는 텍스트뿐이며, 백엔드에서 데이터를 처리할 수 있도록 데이터를 전송하고 처리된 결과를 표시하는 것이 주요 기능이기 때문에, 프론트엔드의 성능보다는 개발 편의성 및 유지보수성을 중심으로 선택하였다. Python을 사용해서 ML 데모 수준의 애플리케이션을 개발할 때는 Gradio를 사용할 수 있지만, 초보적인 수준의 UI만을 만들 수 있으며, 자유도가 떨어지기 때문에 우리의 user flow를 구현하기 위해서는 오히려 프레임워크가 발목을 잡을 것으로 판단하였다. Streamlit은 데이터 기반의 애플리케이션을 개발하는 데에 특화된 Python 프레임워크로, 다양하고 확장 가능한 UI 컴포넌트를 제공하며, 우리의 user flow에 필요한 자유도를 커버한다. 따라서 우리는 프론트엔드 개발 프레임워크로 Streamlit을 선택하였다.

B. 프로젝트 개발환경

본 프로젝트 수행 팀은 3명으로 구성된 팀으로서, 팀 내부 및 회사 담당자와의 원활한 협업 및 지식 공유를 위해 Slack과 Notion등의 커뮤니케이션 도구를 활용하여 의사소통하고 기록을 남긴다. 원활한 산출물 관리를 위해 분산 버전 관리 시스템인 git을 사용하며, GitHub을 원격 저장소로 사용하며, github-flow를 기반으로 pull request를 통해 개인의 작업 내역을 프로젝트에 통합한다. 일관적인 의존성 관리 및 협업을 위해 팀이 작성한 코드의 저장소를 기능별로 (백엔드, 프론트엔드, ML 등) 분리하지 않고, monorepo 방식으로 저장소를 관리한다.

개발 언어는 ML 생태계가 잘 갖춰져있는 Python을 사용한다. 백엔드 개발을 위해서는 asyncio를 기반으로 고성능 API를 작성할 수 있는 FastAPI를 사용한다. 프론트엔드 개발을 위해서는 Python을 기반으로 데이터 애플리케이션을 구현하는 데에 최적화된 Streamlit을 사용한다. 임베딩과 텍스트를 저장하기 위한 Vector DB의 경우 최근 많은 프로젝트가 우후죽순으로 생겨나고 있으나, Linux

Foundation의 지원을 받아 안정적으로 오픈 소스로 개발되고 있으며 기능 지원 폭이 넓으며, 무엇보다도 latency가 가장 낮은 Milvus를 사용한다.

LLM 서빙을 위해서는 Huggingface에서 제공하는 Text Generation Inference (TGI) 도구를 회사에서 제공하는 Nvidia A100 GPU 서버에 셋업해서 사용한다. RAG에 필요한 query, context encoder 추론 서버는 FastAPI를 사용하여 직접 구현하며, 마찬가지로 회사에서 제공하는 GPU 서버에서 서빙한다. 서버에 모델들을 배포할 때는 일관적인 배포 환경 구성을 위해 Docker 이미지를 사용한다. 보안상의 이유로 GPU 서버의 네트워크는 외부로부터 HTTP 등 일반적인 프로토콜로 직접 접근할 수 없도록 격리되어 있기 때문에, 백엔드 서버에서 ML 추론 서버들에 접근할 수 있도록 reverse proxy를 셋업한다.

ML 모델 및 RAG 파이프라인 실험을 위해서는 회사에서 제공하는 GPU 서버를 사용한다. 인하우스 데이터 레이블링을 위해서는 Google Spreadsheet를 사용하고, 실험 및 데이터 가공에 필요한 스크립트는 Python으로 작성한다. 실험 결과 리포팅을 위해서는 Notion을 사용한다.

4. Goal/Problem & Requirements

본 프로젝트의 목표는 entry-level의 NLP 연구자 입장에서 연구 질의에 대한 답을 빠르고 믿을 수 있게 얻을 수 있는 Research Assistant를 만드는 것이다. 구체적으로는, 연구자가 참조할 문서들을 미리 업로드해두고 관련된 질의를 하면 이를 바탕으로 답변을 생성하는 chatbot 웹 애플리케이션을 만드는 것을 목표로 하였다. 여기에 더하여 우리는 Research Assistant의 기반이 되는 대규모 언어모델과 그 활용방식에 대해 더 탐구하고, 이를 바탕으로 Research Assistant 시스템의 성능을 개선하는 것을 또 하나의 목표로 설정하였다.

이 프로젝트의 성공을 위해서는 엔지니어링부터 연구까지 폭넓은 역량을 필요로 한다. 먼저, 챗봇 웹 애플리케이션 및 모델 추론 서버를 개발하고 배포하기 위한 프론트엔드, 백엔드 및 인프라 역량이 필요했다. 또한, LLM 및 그 활용은 업계에서 통용되는 일반적인 최적해가 있는 것이 아니라 여전히 활발히 연구되고 있는 분야이기 때문에, 최신 연구 성과를 지속적으로 follow-up하면서 이해하고 우리의 도메인과 목표에 맞게 응용할 수 있어야 했다. 특히 retrieval-augmented generation(RAG) 파이프라인의 작동 원리 및 평가 방식에 대한 깊은 이해가 요구되었다.

5. Approach

본 프로젝트의 목표인, 연구자가 질의에 대한 답변을 얻기 위해서 참조할 문서들을 미리 업로드해두면 이를 바탕으로 답변을 생성하는 chatbot 웹 애플리케이션을 만들기 위해서는, 후술할 user flow에서도 잘 드러날 2가지 핵심 모듈이 필요하다. 먼저, 유저가 문서들을 업로드할 수 있는 upload 모듈이 필요하다. 그리고 업로드된 문서들을 바탕으로 답변을 얻을 수 있는 query 모듈이 필요하다. 유저 입장에서 보이는 모듈은 위와 같고, 답변 생성을 위해 필요한 ML 기능 구현을 위해서는 RAG 파이프라인을 구현하고 서빙해야 한다.

Upload 모듈은 유저가 자신의 기기에서 pdf로 된 문서를 선택하고 서버에 업로드하기 위한 기능을 제공하는 모듈이다. 먼저, 유저가 자신의 기기에서 문서를 선택할 수 있는 파일 선택 UI를 보여줄 수 있어야 한다. 파일을 선택하고 업로드 버튼을 누르면 선택한 문서가 서버에 업로드된다. 업로드된 문서로부터는 텍스트를 읽고, 텍스트를 LLM이 참조할 수 있는 적당한 길이의 chunk들로 나눈 뒤, 각각의 chunk에

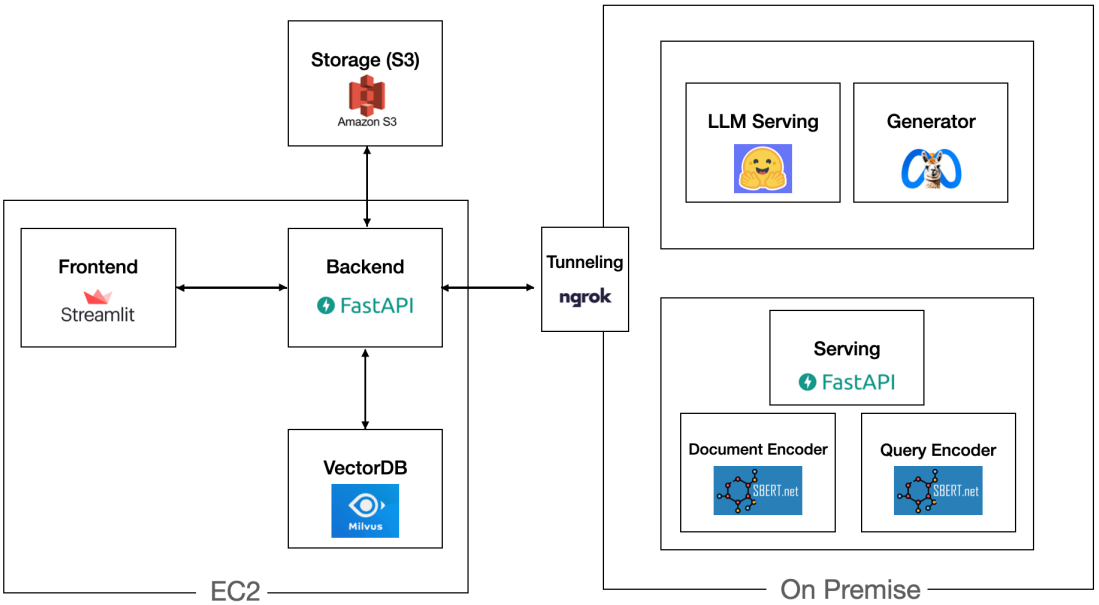
대응되는 임베딩을 추출하고, 추출된 임베딩과 chunk 텍스트의 순서쌍들을 Vector DB에 삽입한다. 모든 처리가 완료된 후에 유저는 화면에서 성공 메시지 및 업로드된 문서의 앞 부분을 확인할 수 있다. Vector DB에 삽입된 UGC는 DB에 설정된 별도의 retention 정책에 따라 관리된다. Upload 모듈에서 문서를 하나 더 업로드한다고 해서 직전에 업로드한 문서가 시스템에서 삭제되지는 않는다. 따라서, 유저는 후술할 Query 모듈에서 그동안 업로드했던 여러 문서들을 바탕으로 질의하고 답변을 얻을 수 있다.

Query 모듈은 연구자인 유저가 질의를 던지면 기존에 업로드한 문서들을 기반으로 한 답변을 제공하는 모듈이다. 먼저, 유저는 대화 창에 질의를 자유롭게 텍스트로 입력할 수 있다. 그리고 질의하기 버튼을 누르면 질의가 서버로 전송된다. 답변을 기다리는 동안 화면에는 유저의 질의가 표시되고, 답변이 생성 중이라고 표시된다. 답변 생성은 RAG 파이프라인이 담당한다. 생성된 답변이 응답으로 돌아오면 유저에게 표시되고, 이어서 다른 질의를 할 수 있다. 화면에서는 세션 안에서 앞서 나누었던 질의-답변의 리스트를 확인할 수 있다.

RAG 파이프라인은 본 프로젝트의 핵심으로, 주어진 유저의 질의로부터 이를 답변하는 데에 필요한 지식을 담고 있는 참고 문서를 불러오고(retrieval) 이를 이용해서 LLM의 생성 프롬프트를 증강하여(augmented) 올바른 답변을 생성하는(generation) 파이프라인이다. Retrieval을 위해서는 임베딩을 추출하는 encoder 모델 및 Vector DB를 사용해서 query 문장의 임베딩과 비슷한 임베딩을 가지는 참고 문서들을 불러온다. Augmentation을 위해서는 LLM이 활용할 수 있도록 적절한 형식의 프롬프트를 작성하고, 여기에 동적으로 불러온 문서들을 포함시킨다. 마지막으로 LLM을 이용해서 프롬프트에 이어질 답변을 생성한다. RAG 파이프라인을 구현하기 위해서는 LLM, query/context encoder, Vector DB를 비롯한 컴포넌트들을 서빙할 수 있어야 하고, 각각의 컴포넌트에 대한 요청을 오케스트레이션하는 파이프라인 구현이 필요하다. 중간발표 시점까지는 우선 기능적으로 원활히 작동하는 RAG 파이프라인을 구현했으며, 이후 최종발표 시점까지는 RAG 파이프라인의 성능 개선을 위해 Prompt Engineering, LLM fine-tuning 등의 추가 작업들을 진행하였다. 우리가 구현한 RAG 파이프라인은 대부분의 질의에 대해서 LLM만 사용했을 때와 비교하여 월등하게 도움되는 답변을 생성할 수 있으며, 특히 최신 논문에 대해서는 ChatGPT와 같이 knowledge cutoff가 있는 상용 서비스보와 달리 hallucination이 아닌 사실을 기반으로 한 답변을 생성할 수 있다.

6. Project Architecture

A. Architecture Diagram



B. Architecture Description

본 서비스의 아키텍처는 크게, 프론트엔드, 백엔드, 그리고 ML 서버로 나눌 수 있다. 또한 데이터를 저장하기 위한 목적으로 Vector DB, 파일 스토리지가 있다.

먼저 각 컴포넌트들의 배포환경에 대해서는 다음과 같이 구현되었다. 프론트엔드와 백엔드는 AWS의 EC2 위에서 클라우드 환경에 배포하여 운영한다. ML 서버의 경우, 마찬가지로 퍼블릭 클라우드 환경에서 서비스할 수 있지만, 이 경우 추론 시 막대한 비용이 발생한다. 때문에, 이미 협업사(softlyai)로부터 제공받은 GPU 를 이용하고자, GPU가 있는 서버에서 tunneling 을 하여 네트워크 망과 연결시키는 방식을 취하였다.

먼저, 프론트엔드는 Streamlit 라이브러리를 이용하여, 신속하게 필요한 UI 를 구성하였다. 백엔드 , ML 서버에도 사용된 파이썬으로 만들어진 프레임워크이기에, 동일한 모듈을 사용하는 등 개발 생산성을 높일 수 있었다. 추가적으로, Streamlit 은 LLM 기반 어플리케이션에서 사용되는 UI 를 추상화하여 제공해주기 때문에, 문제 상황에 적합한 기술이라고 판단하였다.

백엔드의 경우, Django, Flask, FastAPI 등 많은 파이썬 기반의 웹 프레임워크가 존재하였기에 기술 선택이 요구되었다. 본 서비스에서는, 별도의 RDB를 사용하지 않고, 여러 component 를 연결시켜주는 business logic 이 주된 책임이라는 점을 고려하여, 상대적으로 가장 가벼워 빠른 개발을 도모할 수 있는 FastAPI를 선택하였다.

ML 모델을 서빙하기 위해서는, 상술한 것과 같이 GPU 사용을 위해 할당받은 머신에 모델 추론

서버를 띄워두고, 이를 ngrok으로 터널링을 하는 방식을 취하였다. 사용하는 모델들은 크게 embedding 모델과 답변 생성을 위한 generator로 나뉜다. embedding 모델들은 가용한 모델들 중에서 Research Assistant의 retriever로 사용하기에 가장 적합하다고 평가된 모델을 사용하였다. (평가 과정 및 결과는 8.C, 9절 참조) 또한, document와 query를 별개의 모델로 처리하는 것이 성능이 더 좋다고 판단하여, document encoder와 query encoder를 따로 띄워두었다. 백엔드 컴포넌트와 통신하기 위해서는 HTTP 요청을 받을 수 있어야 하므로, 이를 위해 간단한 FastAPI 서버를 띄워두어 위 모델들을 호출하도록 하였다.

Generator의 경우, open source LLM 중 현 시점 가장 널리 쓰이는 Llama2를 사용하기로 하였다. 다만, 7B, 13B 등 모델 크기에 대한 기술 선택이 필요하였다. 모델 크기를 키움으로써 얻을 수 있는 이득인 답변의 질적 우위와, 가벼운 모델로부터 얻을 수 있는 짧은 응답의 latency 간의 트레이드 오프가 있었다. 유저가 사용하는 서비스에서는 반응 속도의 중요성이 매우 크다는 점, 7B 모델 역시 충분히 좋은 성능을 낸다는 점을 고려해 작은 모델인 7B를 사용하기로 결정하였다. 추가적으로, 작은 모델에서 추가적인 학습 (8.C. Research Issues & Details 의 Fine-tuning LLM for RAG에서 후술) 을 통해, 성능을 향상시켰다. LLM 서빙은 Huggingface의 TGI 프레임워크를 이용하여, max input token length 등 Generation 에 특화된 파라미터 값을 쉽게 처리하도록 하였다.

그 외에, embedding과 문서들을 저장하기 위한 Vector DB로는 Milvus를 사용하였다. 유저의 요청을 받아 embedding을 저장하거나 similarity search를 하기 위해서는, 무엇보다도 빠른 속도가 중요하다. 여러 Vector DB의 benchmark 를 비교한 결과, Milvus가 가장 높은 성능을 보임을 확인하였다. 다만, 아쉽게도 아직까지 AWS RDS와 같이 클라우드에서 제공하는 검증된 managed service가 출시되지 않아, Vector DB에 직접 query를 하는 컴포넌트인 백엔드 서버와 동일한 EC2 인스턴스에 직접 띄워두기로 결정하였다.

마지막으로 유저가 올린 pdf 파일을 S3에 저장하기로 하였다. 유저가 업로드한 파일을 Upload API를 처리하는 동안 메모리 상에만 유지하는 것은, 데이터가 유실될 가능성이 매우 높아, 이를 저장하는 것은 매우 필수적이다. 백엔드 서버에 직접 저장하는 방식도 가능하겠으나, 파일 저장 자체를 목적으로 신뢰성과 성능, 그리고 비용에서 우수함을 보이는 S3에 저장하는 것이 조금 더 안정적인 방식이라 판단하였다.

7. Implementation Spec

A. Input/Output Interface

Upload

서비스의 메인 화면에는 Upload PDF, Query 두 탭이 있다. 먼저, 유저는 자신이 업로드 하고자 하는 논문들을 Upload PDF 탭에서 업로드한다. Browse files 버튼을 눌러서 직접 저장되어 있는 파일을 선택하거나, drag and drop 으로 업로드 할 수 있다. 업로드가 성공하면, 자신이 올린 파일 이름의 리스트가

확인된다. 서버에서 파일을 처리하여 반환된 후에는 업로드한 PDF 의 내용 초반부를 확인할 수 있다.

Query

유저가 Upload PDF 탭을 통해 논문들을 업로드하고 나면, Query 탭에서 Research Assistant와 질의응답을 시작할 수 있다. Ask Question 이라는 제목의 입력창을 통해 논문과 관련한 질의를 입력하고 나면, History 로그에 유저가 남긴 질의와 함께 “Generating...” 이라는 로그가 표시되어 유저가 Assistant가 답변을 생성하고 있음을 인지하도록 도와준다. Assistant의 답변이 부분적으로 생성되기 시작하면 답변이 streaming되어 실시간으로 생성된 부분만큼 업데이트되어 표시된다. 최종적으로, Assistant의 답변 생성이 완료되면 유저와 Assistant의 답변 로그가 아래와 같은 형식으로 기록된다.

- User: {user query}
- Assistant: {generated answers from assistant}

이후에도 유저는 Assistant와 계속해서 대화를 이어나갈 수 있으며, 이어지는 대화 내용도 위와 같은 양식으로 History 로그에 이어서 기록된다.

B. Inter Module Communication Interface

Upload API

유저는 질의 시점에 답안의 근거가 될 수 있는 내용을 포함하고 있는 논문들을, 사전에 업로드해야 한다. 이를 담당하는 것이 Upload API이다. 유저는 UI에서 업로드할 파일을 선택한다. Upload API는 이 파일들을 multipart-form-data의 content type으로 post 요청을 날린다. 백엔드 서버에서는 요청을 받은 후, 먼저 업로드한 파일을 서버에 저장해둔다. 그리고, pdf 파일의 텍스트만을 추출하고, 이를 100개 토큰 단위로 나누어, chunk list 를 얻는다. 이렇게 chunking 된 데이터를 document 용 Encoder API 를 호출하여, 벡터화된 document 를 얻는다. 이때 Encoding API로부터 반환되는 값은 base64로 인코딩된 상태이므로 디코딩 과정을 거쳐 float value로 변환한다. 그리고 이를 Vector DB에 대응되는 텍스트 형태의 chunk를 함께 insert 한다. 위 과정까지 성공적으로 완료될 경우, 유저가 업로드한 파일들의 초반부 내용을 포함한 답변을 response로 클라이언트에게 보낸다.

Query API

유저의 질의(query)를 입력으로 적절한 답변을 생성하여 반환하는 query API 역시 FastAPI 기반의 backend에서 구현하였다. 답변을 생성하는 과정은 다음과 같다. 유저의 질의를 입력으로 Encoder API에 요청을 보내 질의를 벡터화한다. Upload API의 경우와 마찬가지로, Encoding API로부터 반환된, base64로 인코딩된 값을 디코딩 과정을 거쳐 float value로 변환한다. 이후 Vector DB에서 변환된 벡터로부터 L2 distance가 가장 작은 N개의(현재 N=3) passage들을 retrieve한다. retrieve된 passage들은 context로서 유저의 질의와 함께 접합되어 prompt를 구성하게 되고, 이 prompt를 입력으로 LLM API에 요청을 보내 LLM의 답변을 받아온다. LLM의 답변은 prompt를 포함한 채로 반환되므로 prompt의 길이만큼 prefix를 잘라낸다. 이렇게 잘라낸 LLM의 답변이 Query API의 반환값이 된다.

Encoder API

Encoder는 유저가 입력한 질의 문장 또는 참고하고자 하는 문헌의 문단에 대한 임베딩 벡터를 추론한다. 이를 위해 Encoder는 2개의 HTTP API를 노출한다. 첫 번째 API는 Query encode API로, 한 개의 query 문장을 입력으로 받아서 한 개의 임베딩 텐서를 반환한다. 두 번째 API는 Passage encode API로, passage들의 리스트를 입력으로 받아서 배치 처리된 임베딩 텐서를 반환한다. 임베딩 차원이 768 차원일 때, 첫 번째 API는 (768,), 두 번째 API는 (Passage 리스트 원소의 개수, 768) 형태의 텐서를 반환한다.

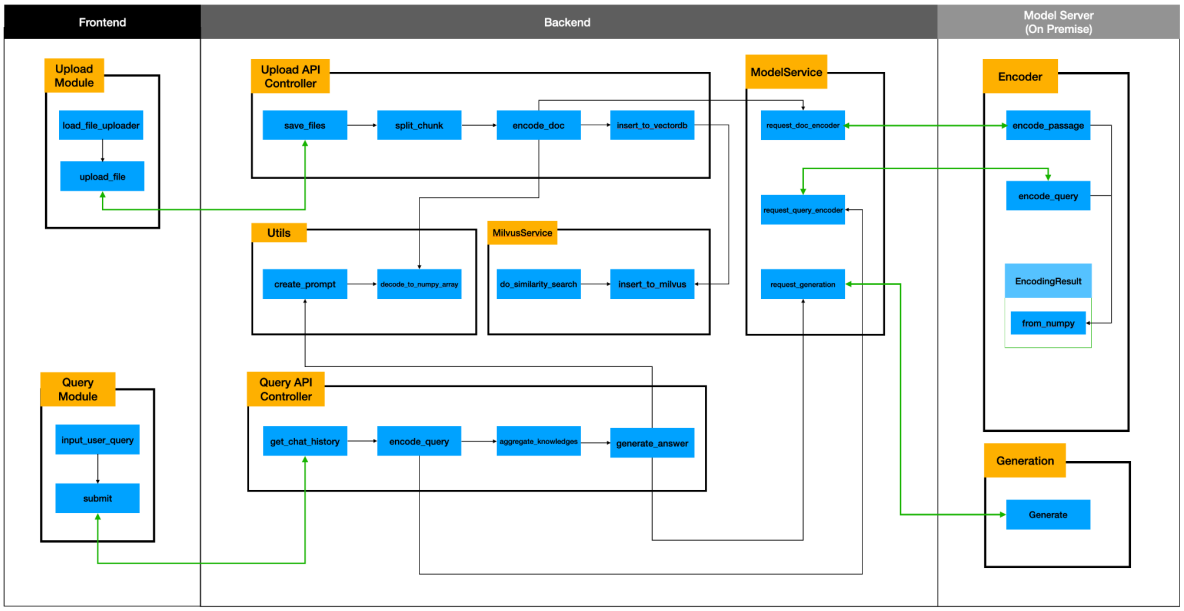
두 API는 모두 JSON을 body 입력으로 받는 POST API다. API 응답은 JSON으로, base64 인코딩된 텐서 데이터, 그리고 텐서의 형태를 반환한다. 텐서를 JSON에서 float의 array로 표현하는 경우, 각각의 자릿수가 문자로 표현되어 최소 1바이트씩의 용량을 차지하기 때문에 정보량을 고려했을 때 필요한 바이트 수 보다 최소 3배 이상의 오버헤드가 발생하며, Python list로 변환하는 과정에서도 오버헤드가 발생한다. 통신 및 파싱에 오랜 시간이 걸리는 문제를 해결하기 위해, 텐서 데이터를 구성하는 바이트를 그대로 base64 인코딩해서 오버헤드를 최소화한다.

LLM API

TGI 서버는 일반 생성 API 및 스트리밍 생성 API를 노출한다. 두 API 모두 동일한 파라미터에 대해 동일한 토큰들을 생성하지만, 일반 생성 API는 생성이 완료된 이후에 결과를 한번에 반환하기 때문에 간단하게 사용할 수 있다는 장점이 있고, 스트리밍 생성 API는 생성된 부분 결과들을 반복해서 반환함으로써 반응형 애플리케이션을 개발할 수 있다는 장점이 있다. 두 API는 모두 프롬프트 텍스트 및 생성 파라미터를 JSON으로 입력받으며, 생성된 텍스트를 JSON으로 반환한다. 중요한 생성 파라미터로는 생성할 텍스트의 최대 길이를 제어하는 max_new_tokens, 생성 결과를 얼마나 결정론적으로 만들거나 창의적으로 만들지 사이의 트레이드오프를 조절할 수 있는 temperature, top_p 가 있다.

C. Modules

Research Assistant의 프론트엔드는 user flow에서 나타나는 2가지 핵심 기능을 지원하기 위한 2개의 모듈로 나누었다. 백엔드 또한 각각의 프론트엔드 기능의 서버 동작을 전담하는 2개의 모듈로 나누었다. 백엔드에서는 요청을 완수하기 위해 ML 추론 서버 및 Vector DB 등 외부 모듈에 의존한다.



Frontend

Streamlit 기반의 frontend server는 EC2 인스턴스에서 운영되며, 유저가 문서를 업로드하는 **Upload Module** 과, 유저의 질의를 입력받아 서버에 요청할 수 있는 답변을 생성하는 **Query Module**로 이루어져 있다.

- **Upload Module**

프론트엔드의 Upload Module 은 상단 탭의 Upload PDF 탭을 활성화할 경우 확인할 수 있다. Streamlit으로 개발되었으며, 필요한 논문 파일을 파일 탐색 모달과 drag & drop 을 통해 업로드할 수 있다. 유저는 자신이 선택한 파일들의 목록을 UI 상에서 확인할 수 있다. 유저가 파일을 업로드하면 Upload PDF API 가 호출되고, 요청이 처리되면 파일의 초반 내용이 응답으로 반환된다.

- **Query Module**

프론트엔드의 Query Module은 Query 탭의 UI를 구성하는 코드들로 구성되어 있다. Streamlit의 session_state를 이용해 chat history를 관리하며, text_area를 통해 유저의 질의를 입력받는다. submit 버튼이 눌리면 text_area에 입력된 내용을 body로 backend의 Query AI로 요청을 보내 적절한 답변을 받도록 구현하였고, 유저와 assistant간의 대화 기록은 chat history로 관리되며 누적되어 기록된다.

Backend

FastAPI 기반의 backend server는 EC2 인스턴스 위에서 운영되며, 유저가 업로드한 문서를 받아 Vector DB에 추가하는 **Upload API** 와 유저의 질의를 기반으로 답변을 생성하는 **Query API,** 그리고 별도의 책임을 할당 받는 기타 모듈들(Milvus Service, Model Service, Utils)로 이루어져 있다.

- **Upload API Controller**

유저로부터 PDF 파일들을 요청으로 받아, 이를 바탕으로 후에 질의 시 필요한 문헌들을 저장해두는 Upload API Controller 는 다음과 같은 같은 과정을 거쳐 Upload 요청을 처리한다. Multipart form data 로 pdf 파일을 건네 받은 후, 해당 파일을 저장해둔다. 그리고, Query Encoder 가 처리할 수 있도록, pdf 파일의 텍스트를 추출하여 100 개 단위로 잘라낸다. 이런 일련의 과정을 통해 논문의 내용을 담고 있는, String List 데이터를 얻는다. 이후, 이를 ML 서버 요청을 하여, chunking 된 문서에 대응되는 vector 화 된 값을 얻는다. 이후, 이를 Vector DB에 insert 하고, 유저에게 올린 문서들의 초반부 내용을 반환해준다.

- **Query API Controller**

Query API Controller에서는 유저의 질의에 맞는 답변을 생성하여 반환하는 Query API를 제공한다. 유저의 질의가 들어오면, 그 질의를 Encoder API를 통해 벡터화하고, 벡터화된 질의를 이용해 Vector DB에서 질의와 관련된 passage들을 찾아 retrieve한다. Retrieve된 passage들과 유저 질의는 유틸 함수를 통해 적절한 prompt로 포맷팅되며, 이 prompt를 입력으로 LLM API에 요청을 보낸다. 반환된 LLM의 답변이 Query API의 반환값이 되며, 이 값이 프론트엔드의 Query 모듈에 나타나게 된다. 이 과정에서 질의가 LLM API의 max token length를 넘거나 유해한 컨텐츠를 포함하는 질의가 들어오는 등의 예외 상황이 발생할 경우, 기술적인 문제로 답변을 생성할 수 없다는 메시지를 반환한다.

- **MilvusService**

Upload API 와 Query API를 처리할 때, 모두 Vector DB 에 대한 의존성이 발생한다. 때문에, 코드의 응집성을 위해 공통된 관심사를 담당하는 MilvusService 클래스를 만들어, Vector DB 와 관련된 logic 들을 담당하도록 구현하였다. 구체적으로는, Upload API 에서 사용되는 insert, Query API 에서 사용되는 similarity search 를 구현하였다.

- **ModelService**

백엔드 서버의 관점에서는, LLM, 임베딩 모델을 서빙하는 ML 서버는 자신이 많은 요청을 날리는 하나의 '웹 서버'로 생각할 수 있다. 때문에 해당 서버에 대한 관리를 해주는 클래스를 만들어 관련된 책임을 몰아두고 다른 비즈니스 로직 레이어를 추론 인프라 레이어로부터 최대한 분리하는 것이 좋은 구현 패턴이라고 판단하였다. ModelService 내에서는, ML 서버에 호출하는 모든 요청들(query encode, document encode 그리고 generation)을 내부적으로 호출하도록 하고, 이를 별도의 메소드로 만들어, 실제 API Controller 에서는 단순히 메소드 호출로 처리할 수 있도록 Wrapping 해주었다. 즉, 비즈니스 로직에서는 ModelService에만 의존하고, 모델 서버의 HTTP API에 요청을 보내고 결과를 파싱하는 역할은 ModelService이 책임을 진다.

- **Utils**

Utils 모듈에서는, 메인 로직에서는 직접 구현하기에는 지나치게 추상화 레벨이 낮은 기능들을 구현해두었다. 가령, decode_to_numpy_array 메소드의 경우, utf-8 으로 인코딩된 텐서를, 다시 디코딩하여 원래의 값을 지닌 텐서로 복구해내는 기능을 담당한다. 유사하게, create_prompt 역시

generation 요청을 하기 전에, prompt 를 유저의 질의와 문서들을 혼합하여 prompt 를 만드는 데에 사용된다.

Model Server

- Encoder

Encoder API를 제공하기 위한 Encoder Inference Server는 FastAPI로 직접 개발하였으며, 회사에서 제공한 GPU 인스턴스 위에서 Docker 컨테이너로 실행된다. 만약 추론 서버가 한 번의 요청에 하나의 텍스트만 처리할 수 있다면, 백엔드에서는 논문 하나를 업로드 하기 위해서 수십 번의 추론 요청을 보내야 하고, 이는 네트워크 대역폭 및 시스템 자원의 낭비로 이어진다. 따라서, 추론 서버 및 API는 한 번의 요청에 여러 텍스트를 처리할 수 있도록 구현했다. 또한, GPU의 병렬 연산 능력을 최대한 활용하기 위해, 요청에 포함된 많은 텍스트에 대해 각각 추론하는 것이 아니라 적당한 batch size로 묶어서 빠르게 처리할 수 있는 batching 기능을 구현했다.

- Generation

Huggingface 에서 제공하는 TGI를 사용하여 서빙하였다. TGI 는 LLM 서빙에 특화된 프레임워크로 웹 서버 운영과 같은 부분을 신경써주지 않게 한다. TGI는 LLM 서빙 시 필요한 중요 파라미터들을 Customization 할 수 있다. 유저가 필요에 따라 매 요청마다 파라미터의 값들을 함께 보내어 동적인 설정도 가능하다.

8. Solution

A. Implementations Details

Backend

- QueryAPIController:
    - get_chat_history() -> List[str]:
        - 사용자의 이전 대화 내역을 가져옴
    - encode_query(query: str) -> np.ndarrary:
        - 사용자의 질문을 인코더 API를 사용하여 벡터화
    - aggregate_knowledges(search_result: List[Hit]) -> List[str]:
        - Milvus 검색 결과에서 문서 내용을 집계
    - generate_answer(lm_prompt: str) -> str:
        - 주어진 프롬프트로 LLM 모델에 쿼리를 보내고 생성된 답변을 반환
- UploadAPIController:
    - save_files(files: List[UploadFile]) -> List[str]:
        - 업로드된 PDF 파일들을 저장하고 URI를 (테스트 서버 실행시에는 서버 파일시스템 경로, 프로덕션 서버 실행시에는 S3) 반환

    - **split_chunk(text: str, max_words: int) -> List[str]:**
        - 최대 단어 수를 기준으로 입력 텍스트를 청크로 분할
    - **encode_doc(file_data: bytes) -> Tuple[np.ndarray, List[str]]:**
        - 주어진 문서에 대한 문서 인코딩을 요청하고, 인코딩된 텐서 및 청크로 분할된 문서를 반환
    - **insert_to_vector_db(Tuple[np.ndarray, List[str]]) -> None:**
        - 인코딩된 텐서 및 청크로 분할된 문서를 Vector DB에 삽입. 내부 동작을 위해서 MilvusService에 의존
- **ModelService:**
    - **request_doc_encoder(file_data: bytes) -> Tuple[np.ndarray, List[str]]:**
        - 주어진 문서에 대한 문서 인코딩을 요청하고 인코딩된 텐서 및 청크된 문서를 반환
    - **request_query_encoder(query: str) -> np.ndarray:**
        - 사용자 쿼리에 대한 인코딩을 요청하고 인코딩된 텐서를 반환
    - **request_generation(lm_prompt: str) -> str:**
        - 주어진 프롬프트로 LLM 모델에 쿼리를 보내고 생성된 답변을 반환
- **MilvusService:**
    - **insert_to_milvus(data: Tuple[np.ndarray, List[str]]) -> None:**
        - 인코딩된 데이터를 Milvus 컬렉션에 삽입
    - **do_similarity_search(query_tensor: np.ndarray, param: Dict[str, Any], limit: int) -> List[List[Hit]]:**
        - Milvus 컬렉션에서 similarity search를 수행하고 검색 결과를 반환
- **Utils:**
    - **decode_to_numpy_array(encoding_bytes: bytes, encoding_shape: List[int]) -> np.ndarray:**
        - Base64로 인코딩된 바이트를 텐서(NumPy 배열)로 디코딩
    - **create_prompt(query: str, contexts: List[str]) -> str:**
        - 사용자 쿼리와 집계된 문서 내용을 결합하여 LLM에 입력할 프롬프트를 작성

Frontend

- **UploadModule:**
    - **load_file_uploader() -> UploadFile:**
        - 파일 업로더를 불러오고 업로드된 파일을 반환
    - **upload_file(uploaded_file: UploadFile) -> str:**
        - 백엔드의 upload API를 호출하여 파일을 업로드하고, 사용자에게 보여줄 파일 첫 부분을 가져옴
- **QueryModule:**
    - **input_user_query() -> str:**
        - 사용자로부터 질문을 입력받음

- **submit(user_query: str) -> str:**
  - 사용자의 질문을 백엔드 서버에 제출하고, 생성된 답변을 반환

Model Server

- **Encoder:**
  - **encode_query(query: str) -> EncodingResult:**
    - 주어진 쿼리에 대한 인코딩을 생성하고 반환
  - **encode_passage(chunked_passage: List[str]) -> EncodingResult:**
    - 주어진 청크로부터 인코딩을 생성하고 반환
- **EncodingResult:**
  - **from_numpy(array: np.ndarray) -> EncodingResult:**
    - NumPy 배열의 데이터로 EncodingResult 객체를 생성하여 반환
- **Generation:**
  - **generate(inputs: str, parameters: Dict[str, Any]) → Dict[str, Any]:**
    - inputs의 프롬프트를 사용해서 parameters의 설정에 따라 LLM으로 생성한 결과를 반환

B. Implementations Issues

Response Streaming

LLM은 i번째 토큰을 예측하기 위해 입력으로 들어온 토큰과 i-1번째까지 예측된 토큰들에 의존하는 autoregressive decoding 방식으로 답변을 생성한다. 토큰들 사이의 이러한 causal 관계는 하나의 요청에 대해 여러 토큰을 동시에 생성할 수 없으며, 답변이 길어질수록 생성 시간도 거의 비례해서 증가한다는 것을 의미한다. Research Assistant는 도메인 특성상 yes/no 또는 짧은 문장으로 답변하기 어려운 질문을 많이 받게 되므로 LLM 생성 latency에 특히 취약하게 된다. 그러나 이러한 높은 latency는 Research Assistant의 사용자 입장에서는 아무런 추가적인 행동을 취할 수 없고 그저 기다려야 하는 시간이며, 필요한 정보를 얻기까지의 지연과 피로감을 유발하게 된다.

사용자 입장에서 경험하게 되는 latency를 최소화하기 위해서 생성된 토큰들을 streaming하여 유저에게 보여주기로 결정하였다. Huggingface TGI에서 streaming 옵션을 지원하기 때문에 쉽게 구현할 수 있을 것으로 예상했지만, TGI에 최종 사용자가 직접 의존하는 게 아니기 때문에 추가로 고려해야 할 부분들이 있었다. 우선 Research Assistant 백엔드 서버에서 TGI에 요청해서 생성된 streaming된 토큰들을 프론트엔드에도 streaming 답변으로 전달해야 한다. 프론트엔드에서는 streaming된 답변이 매끄럽게 화면에 업데이트되도록 지속적으로 답변을 업데이트해주어야 한다. 구현의 복잡도는 높았지만, 이를 통해 유저 입장에서 질문을 날린 후 화면에 답변이 표시되기 시작하기까지의 시간을 5초 이내로 줄일 수 있었다.

Stopping Vector DB from Growing Indefinitely

이 프로젝트에서 개발한 RAG 파이프라인은 Vector DB인 Milvus를 핵심 구성요소로 사용한다. Vector

DB는 retrieval 대상 임베딩들의 공간에서 빠른 approximate nearest-neighbor search를 가능하게 만든다. 이를 위해서 Milvus는 HNSW(Hierarchical Navigatable Small World) 가정을 바탕으로 빠르게 탐색할 수 있는 벡터들의 인덱스를 구축한다. Milvus는 빠른 탐색을 위해 검색할 인덱스를 메모리에 저장한다. 그런데 유저가 누적됨에 따라 지속적으로 인덱스의 크기가 커지고, 더 많은 메모리를 요구하게 된다. 때문에 우리가 현재 배포를 위해 사용하고 있는 작은 규모의 EC2 인스턴스에서의 서빙이 어렵게 되고, 더 많은 컴퓨팅 비용을 필요로 하게 된다. 이러한 비효율을 해소하기 위해서는 다시 사용될 가능성이 낮을 것이라고 예상되는 벡터들을 인덱스에서 삭제할 수 있어야 한다.

LRU 등 캐시 관리 정책을 직접 구현해서 Vector DB 인덱스의 용량을 제한하면서 접근할 수 있지만, 이 경우 DB에 벡터를 삽입할 때 마다 DB에서 추방할 벡터를 탐색하는 오버헤드가 추가된다. Latency를 최적화하는 것도 우리의 목표이기 때문에 이는 원하지 않는 부정적 효과다. 이러한 오버헤드를 주지 않으면서 Vector DB 인덱스의 크기를 제한할 수 있는 방법을 탐색했고, 정확하게 벡터의 수를 제한하지는 않지만 벡터가 유지되는 기한을 제한함으로써 비슷한 효과를 얻기 위해 collection.ttl.seconds 옵션을 설정하기로 하였다.

Embedding Vector Transmission Format

Embedding model 서버에서는 여러 문장들에 대한 임베딩을 batch로 반환하게 된다. 임베딩 벡터는 메모리 상에서는 벡터의 element 수 * 자료형의 byte 수 만큼의 용량을 차지하게 된다. 예를 들어 768차원의 32-bit float 벡터는 메모리에서 약 3KB를 차지한다. 그러나 임베딩 서버는 응답을 문자열로 반환하며, 벡터를 serialize하기 위한 방법이 필요하다. 벡터를 JSON으로 serialize하는 가장 자연스러운 방법은 각각의 원소 float들의 10진법 표현이 쉼표(“,”)로 구분되고, 시작과 끝이 [ ]로 표시된 배열 표기법을 사용하는 것이다. 이는 사람이 이해하기 쉬우며 JSON에서 곧바로 사용할 수 있다는 장점을 가진다. 그러나 이 방법은 많은 오버헤드를 유발한다. 우선 전송 용량 관점에서, 원소들의 10진법 표기법 문자열은 원본 데이터가 메모리에서 차지하는 용량보다 더 많은 용량을 요구한다. 예를 들어, 어떤 소수를 소수점 아래 10자리까지 문자열로 표현한다면, 총 12개의 글자를 필요로 하며, 최소 12 byte를 필요로 한다. 이는 32-bit float (4 byte) 를 사용하는 경우의 3배이다. 결과적으로 임베딩 전체의 전송에 필요한 용량 또한 3배가 된다. 또한 대규모의 JSON을 파싱하는 연산은 시간을 오래 소모한다.

이 이슈를 해결하기 위해서 보다 메모리 효율적이고 빠르게 deseriallize할 수 있는 임베딩 전송 방법을 구현했다. 임베딩의 byte 데이터를 base64로 인코딩한 뒤, 인코딩된 문자열을 JSON에 포함시켜 보내는 것이다. base64로 표현된 임베딩 데이터는 원본 byte에 비해 4/3배의 용량을 차지하여 작은 오버헤드가 발생하지만, base64 문자열은 JSON에 포함시킬 때 별도의 serialization이 필요하지 않기 때문에 추가적인 오버헤드가 발생하지 않는다. 이 때, 임베딩의 byte 데이터는 원소들의 값과 순서만을 보존하며 shape 정보를 담고 있기 때문에, 수신자는 flatten된 벡터만을 얻을 수 있다. 그래서 임베딩의 shape 데이터를 array로 함께 JSON에 포함시켜서 올바른 shape로 복원할 수 있도록 하였으며, 이는 임베딩 자체의 데이터보다 아주 작은 오버헤드를 유발한다. 결론적으로, 우리는 JSON array를 사용하는 naive한 구현 대비 약 1/9의 용량만으로 데이터를 전송할 수 있는 공간 효율적인 serialization 방법을 구현하였으며, 수신자가 읽을 때도 복잡한 JSON array 파싱을 요구하지 않기 때문에 더 시간 효율적이다.

C. Research Issues & Details

본 프로젝트에서는 단순히 Research Assistant라는 서비스를 개발하는데서 그치지 않고, research domain에 특화된 RAG (Retrieval Augmented Generation) 시스템을 개선하는 방법에 대해 고민하고 기여하고자 하였다. 이에 RAG와 관련한 논문들을 서베이하며 문제의식들을 정리했고, 아래와 같은 방식으로 각 문제들에 접근하였다.

Prompt Engineering

Language model을 사용할 때의 prompt 구성에 대한 중요성은 계속 강조되어 왔다. 일례로, “Let’s think step by step.” 이라는 문장을 덧붙여주기만 해도 language model이 생성한 답변의 퀄리티가 좋아지기도 했으며[1], retrieve된 passage들을 넣는 순서도 RAG 시스템의 성능을 크게 바꾸는 요소이기도 했다[2]. 이러한 배경으로, 본 프로젝트에서는 다양한 템플릿을 테스트하며 Research Assistant에 맞는 system prompt와 context 구성 방법에 대해 탐구하였다. Research Assistant를 위한 prompt engineering을 위해서는 우선 “Research Assistant로서 좋은 답변”이 무엇인지를 찾아야 했다. 이를 위해서 ChatGPT를 활용하여 ChatGPT의 knowledge cutoff 이전에 출판된 유명한 NLP 논문들에 대한 동일한 질문에 여러 종류의 답변을 얻고, 답변들을 비교해서 팀원들이 “좋은 답변"이라고 여기는 샘플들로부터 관찰된 좋은 요소들을 prompt에 반영하는 human-in-the-loop 프로세스를 반복함으로써 prompt를 개선하였다. Research Assistant에 사용된 prompt 및 후술할 데이터셋 생성 및 평가를 위해 사용된 prompt들은 모두 Appendix B에 공개하였다.

Fine-tuning LLM for RAG

초기 LLM이 instruction을 follow하지 못해 사람에게 유용한 정보를 제공하지 못했던 한계를 InstructGPT[3]가 LLM을 용례에 맞게 fine-tune하는 것으로 풀어낸 것처럼, 본 프로젝트에서는 LLM을 Research Assistant의 역할에 맞게 fine-tune하는 것으로 RAG 시스템의 성능을 개선하고자 하였다.

LLM을 fine-tune하기 위한 다양한 방법 중 가장 유명해진 것은 PPO(Proximal Policy Optimization) 기반의 RLHF(Reinforcement Learning from Human Feedback)이지만, 강화학습 알고리즘 특성상 학습이 까다롭고, 사람의 선호도를 모든 LLM output에 대해서 레이블링받는 게 아니라, 사전에 레이블링된 선호도를 학습한 별도의 LLM 모델을 reward model로 학습하고 이 모델로부터 reward를 얻어 학습하기 때문에, 실제 사람이 선호하는 것과는 거리가 멀지만 reward model은 좋다고 잘못 판단하는 샘플들에 overfit하는 reward hacking 문제가 발생하기 쉽다. 우리는 이러한 한계로부터

---

[1] Kojima, Takeshi, et al. "Large language models are zero-shot reasoners." *Advances in neural information processing systems* 35 (2022): 22199-22213.

[2] Liu, Nelson F., et al. "Lost in the middle: How language models use long contexts." *arXiv preprint arXiv:2307.03172* (2023). https://arxiv.org/abs/2307.03172

[3] Ouyang, Long, et al. "Training language models to follow instructions with human feedback." *Advances in Neural Information Processing Systems* 35 (2022): 27730-27744.

비교적 자유롭다고 알려져있는 보다 최신의 학습 알고리즘인 DPO(Direct Preference Optimization)[4]을 기반으로 LLM을 research assistant 용례에 맞게 fine-tune하였다. 먼저, LLM을 research assistant 도메인에 맞는 데이터에 학습해야 한다. 이 때의 loss는 LLM pretraining시에 사용하는 것과 동일한 loss이고, 이 과정을 SFT(Supervised Fine Tuning)이라고 부른다. 다음으로, SFT된 LLM으로부터 동일한 입력에 대한 두 가지 답변을 샘플링하고, 생성된 답변들 중 더 적절한 답변을 레이블링한 데이터를 구축한다. DPO에서는 최종적으로 두 답변 사이의 선호 관계를 학습할 수 있는 DPO loss를 사용해서 최종 모델을 학습한다.

Research Assistant의 용례에 맞는 데이터셋이 부재하기 때문에, 커스텀 데이터셋을 구축하였다. 저비용 고효율로 좋은 퀄리티의 데이터셋을 구축하기 위해, 팀원들이 직접 ChatGPT 및 OpenAI API를 레버리지하여 데이터셋을 생성하였다. Research Assistant를 위한 SFT 데이터셋은 논문, 질문, 답변의 쌍들이 필요하다. 가장 먼저, 논문들의 리스트를 생성하였다. OpenAI API를 이용해서 knowledge cutoff 이전의 GPT가 잘 알고 있는 논문들의 리스트를 생성하였고, 팀원들의 정제 과정을 거쳐서 논문 제목 데이터셋을 만들었다. 다음으로, 각각의 논문에 대해 할 수 있는 질문들과 이에 대한 답변을 생성했다. 앞서 GPT가 잘 알고있는 논문들을 선택했기 때문에, 데이터 생성 과정에서는 RAG 없이 논문 제목만으로 질문 및 답변을 생성할 수 있었다. 좋은 질문과 답변을 생성하기 위해서는 위의 “Prompt Engineering” 절에서 서술한 것과 같이 반복적인 개선 과정을 거쳤다.

DPO 데이터셋은 컨텍스트, 질문, 선택된(선호되는) 답변, 선택되지 않은(덜 선호되는) 답변의 쌍들이 필요하다. 이 때, 답변들은 우리가 학습하고자 하는 모델로부터 생성된 것이어야 하며, 실제로 모델을 사용할 환경의 분포와 잘 맞아있는 데이터를 수집할수록 실 성능을 향상과 잘 align된 데이터를 만들 수 있다. 그래서 먼저 앞서 생성한 SFT 데이터셋을 사용해서 llama2 모델을 학습했다. 그리고 학습된 모델을 배포하여 Research Assistant 백엔드 서버에 연동하였고, Upload API - Query API를 순차적으로 호출해서 프로덕션과 동일한 셋업에서 답변들을 생성하였다. 이 때, 동일한 질문에 대해서 두 가지 답변을 생성하였다. 최종적으로, 두 가지 답변 중에서 어떤 답변이 선호되는지를 GPT-4를 사용해서 레이블링하였다. GPT-4는 GPT-3.5에 비해 비용이 비싸지만, 훨씬 더 높은 수준의 추론 능력을 요구하는 태스크를 수행할 수 있으며, 선행 연구들에서도 GPT-4를 이용해서 평가를 진행하고 이것이 실제 사람들의 평가과 잘 맞아떨어진다고 보고하였다. 우리는 앞선 prompt engineering 과정에서 파악한 “Research Assistant로서 좋은 답변의 기준"을 기반으로 두 가지 답변 중 더 좋은 답변을 판별하는 prompt를 작성하였고, 이를 이용해서 OpenAI API (gpt-4-1106-preview 버전) 를 이용해서 답변 선호를 레이블링했다. 정확한 레이블링을 위해서 샘플링은 사용하지 않았다. (temperature=0) 그리고 position bias가 레이블링 정확도에 줄 수 있는 영향을 배제하기 위해, 답변 2개가 prompt 안에서 등장하는 순서를 바꾸어가며 2번 평가하고, 두 번의 평가 결과가 일치하는 경우에만 필터링하여 데이터로 사용하였다.

Embedding Model Selection

참고 문서들을 불러오는데 필요한 embedding model은 RAG pipeline에서 가장 중요한 구성요소 중

[4] Rafailov, Rafael, et al. "Direct preference optimization: Your language model is secretly a reward model." *arXiv preprint arXiv:2305.18290* (2023).

하나이다. 중간발표 단계에서 사용했던 모델인 sentence-transformers의 e5[5]는 가장 널리쓰이는 embedding model 중 하나지만, '연관된' 문서가 아닌 정말 '필요한' 문서를 찾아야 하는 태스크에서는 한계를 드러내기도 했다[6]. 또한, Open Domain QA와 같은 분야에서 RAG 시나리오를 타겟하고 학습된 여러 embedding model 들이 존재한다 (e.g. DPR[7]). 본 프로젝트에서는 이러한 배경으로부터 다양한 embedding model을 평가해볼 것이며, 프로젝트의 목표에 가장 적합한 embedding model을 선정하였다.

Embedding model의 평가를 위해서 end-to-end 방법, 즉 embedding model을 바꾸어가면서 전체 RAG pipeline을 실행하여 답변을 생성하고 비교하는 방법을 사용할 수도 있지만, 시간이 너무 오래 걸리고, 답변 생성 및 평가에 의존하기 때문에 너무 간접적이고 noisy한 방법이어서 선택하지 않았다. 대신, 논문을 대상으로 한 QA 데이터셋인 qasper[8] 데이터셋 위에서 우리의 RAG 파이프라인과 동일한 셋업을 기반으로 retrieval 지표를 평가하였다. Qasper 데이터셋은 질문과 답변뿐만 아니라 레이블러들이 답변을 작성할 때 근거로 사용한 highlighted 부분들이 함께 수집된 데이터셋이다. 우리는 논문 원문을 RAG 파이프라인과 동일한 방식으로 chunking하여 embedding들을 구하고, 질문 embedding과의 dot product / cosine similarity (embedding model의 loss에 따라 각각 다름) 를 기준으로 내림차순으로 정렬하여, recall@k, hit@k를 평가하였다. (Retrieve해올 수 있는 상위 문서의 수를 제한했을 때, 실제로 도움이 되는 문서들 중에서 올바르게 retrieve해온 비율, 개수를 평가; 높을수록 좋은 embedding model이다.) 그 결과, 다양한 QA 데이터셋에 학습된 embedding model인 multi-qa-mpnet-base-dot-v1[9]이 모든 k에 대해 일관적으로 좋은 성능을 보였다. 구체적인 평가 결과는 9절에서 후술할것이다.

Evaluation - Accuracy

Research Assistant의 평가 기준들 중에서, 우리는 Assitant가 얼마나 정확한 답변을 제공하는지인 accuracy를 가장 중요한 기준으로 고려하였다. 때문에 accuracy를 잘 평가하기 위한 방법을 고민하는 데에도 많은 노력을 기울였다.

Accuracy를 평가할 때 특히 유의해야 할 점은 정량화 및 자동화이다. 어떤 이미지가 개인지 고양이인지를 분류하는 것 처럼 간단한 ML 문제에서는 accuracy가 레이블을 맞추었는지 아닌지로 아주 간단하게 정의된다. 하지만 Research Assistant와 같이 복잡한 태스크에서는 옳은 답 자체가 여러 가지

---

[5] Wang, Liang, et al. "Text embeddings by weakly-supervised contrastive pre-training." *arXiv preprint arXiv:2212.03533* (2022).

[6] Ravfogel, Shauli, et al. "Retrieving texts based on abstract descriptions." *arXiv preprint arXiv:2305.12517* (2023). https://arxiv.org/abs/2305.12517

[7] Karpukhin, Vladimir, et al. "Dense Passage Retrieval for Open-Domain Question Answering". In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6769–6781.

[8] Dasigi, Pradeep, et al. "A dataset of information-seeking questions and answers anchored in research papers." *arXiv preprint arXiv:2105.03011* (2021).

[9] https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-dot-v1

경우의 수가 있을 수 있기 때문에, accuracy를 평가하기가 어렵다. 가장 먼저 사용해볼 수 있는 방법은, 사람들이 직접 답변을 보고 어떤 답변이 더 정확한지를 비교 평가하는 것이다. 하지만 이러한 정성평가 방식은 평가자 의존적이어서 주관이 개입할 여지가 많고, 평가 시간 또한 오래 걸리며, 스케일링이 어렵다. 이러한 human evaluation의 한계를 극복하기 위해, 최신 LLM 연구들은 GPT-4와 같은 고성능의 시스템을 레버리지하여 객관적이고 자동화된 평가를 수행한다. 일례로 Llama2 에서도 GPT-4를 이용해 helpfulness를 비교한 바 있으며, GPT-4의 평가 결과에 의존해서 진행한 모델 개선이 human preference로부터 멀어지지 않았다고 보고하였다.[10] 따라서, 우리는 GPT-4를 신뢰할 수 있는 자동화된 정량 평가 수단으로 활용하면서, 팀원들의 human evaluation 또한 보조 수단으로 활용하였다.

GPT-4를 이용해서 답변을 평가하는 데에는 다양한 방법이 있다. 우리는 그 중에서도 각각의 모델에 대해 정량화된 지표를 계산할 수 있는, G-Eval[11]의 방법론을 선택했다. 평가 대상 모델을 이용해서 질문 데이터셋에 대한 답변을 생성한 뒤, GPT-4에게 답변의 정확도를 1~5 스케일로 평가해달라는 프롬프트를 작성하였고, 다음 토큰의 log-probability를 반환하는 API를 사용해서, “1”, “2”, “3”, “4”, “5”가 등장할 확률 분포를 구하였다. 그리고 점수의 기댓값, 즉 토큰의 등장 확률을 weight로 해서 각 토큰이 의미하는 점수의 weighted sum을 답변의 점수로 사용하였다. 이 때, baseline llama를 평가하기 위해서는 Research Assistant 프롬프트에 논문 제목을 포함시켜야 했기 때문에, RAG 파이프라인을 평가할 때도 공평한 비교를 위해 프롬프트에 retrieve된 문장들뿐만 아니라 논문 제목을 명시적으로 포함시켰다.

또한, 다양한 컴포넌트로 이루어진 시스템인 RAG 파이프라인을 평가하기 때문에, ablation 테스트를 통해 각각의 컴포넌트가 어떻게 incremental하게 성능 향상에 기여했는지를 파악하는것이 필수적이라고 판단하였다. 따라서, baseline llama와 최종 pipeline뿐만 아니라, baseline llama에 RAG pipeline만 추가한, LLM 학습은 반영되지 않은 중간 상태의 파이프라인을 함께 평가하여 비교하였다.

9. Results

A. Application Development

Research Assistant 애플리케이션을 구현했다. 완성된 Research Assistant 애플리케이션에는 PDF를 업로드해서 vector db에 인덱싱하고, 이를 기반으로 Assistant와 질의응답할 수 있는 핵심 UI, 그리고 답변을 생성할 수 있는 RAG 파이프라인이 모두 구현되어있다. 또한, 유저 입장에서 느껴지는 latency를 줄이기 위한 token streaming 기능이 구현되어있다. 마지막으로, embedding tensor들을 시간/공간적으로 효율적으로 표현하기 위한 포맷 구현, vector db 인덱스가 너무 커지지 않게 하기 위한 ttl 등, 안정적인 서빙을 위해 필요한 엔지니어링 요구사항에도 대응되어있다.

---

[10] Touvron, Hugo, et al. "Llama 2: Open foundation and fine-tuned chat models." *arXiv preprint arXiv:2307.09288* (2023). https://arxiv.org/abs/2307.09288

[11] Liu, Yang, et al. "Gpteval: Nlg evaluation using gpt-4 with better human alignment." *arXiv preprint arXiv:2303.16634* (2023).

B. Model Research & Development

본 프로젝트의 핵심 목표는 Research Assistant 애플리케이션을 구현하는 것을 넘어서 RAG 파이프라인을 개선하기 위한 연구/개발을 수행하는 것이었다. 이를 위해 우리는 retrieval 및 generation 성능을 모두 높이기 위한 방안들을 탐색하고 수행하였다.

- **Embedding Model 성능 평가**

Retrieval 성능을 높이기 위해서, 가용한 embedding model들의 성능을 평가하고, 최적의 모델을 선정하였다. 성능 평가를 위한 방법은 8-C절의 "Embedding Model Selection" 부분과 같이 진행하였다. 평가 대상 모델들은 다양한 embedding 모델들이 수록된 sentence-transformers의 리더보드에서 상위권에 있는 모델 중에서 아키텍처의 다양성과 Research Assistant 도메인에서의 적합성을 고려하여 선택하였고, 그 외에도 NLP Practitioner들이 자주 사용하는 것으로 알려진 모델들을 함께 테스트하였다. Large 사이즈의 모델은 초기 실험에서 트레이드오프(서빙 안정성 하락 및 latency의 증가)를 고려했을 때 base 사이지의 모델 대비 성능 향상폭이 유의미하지 않아서 않아서 제외하였으며, 모두 base 사이즈를 선택하여 평가하였다.

평가 지표로는 Hits@k와 Recall@k (k=1, 3, 5, 10) 를 사용하였다. Hits@k는 retrieve해온 상위 문서의 수를 k개로 제한했을 때 여기에 포함된 유효한 문서의 수의 평균이다. Recall@k는 같은 경우에 recall인 (retrieve된 유효한 문서의 수) / (모든 유효한 문서의 수) 의 평균이다. 두 지표는 retriever가 유효한 정보를 얼마나 정확하게 가져올 수 있는지를 여러 제한 조건에서 테스트할 수 있도록 한다.

Model	Hits@1	Hits@3	Hits@5	Hits@10
all-mpnet-base-v2[12]	0.186	0.452	0.651	1.047
msmarco-bert-base-dot-v5[13]	0.238	0.518	0.730	1.111
msmarco-roberta-base-v3[14]	0.209	0.474	0.679	1.066
multi-qa-mpnet-base-dot-v1	**0.261**	**0.624**	**0.842**	**1.211**
specter[15]	0.131	0.364	0.563	0.965
e5-base-v2	0.220	0.513	0.715	1.102

표 1. Embedding model 후보들에 대한 Hits@k 평가

[12] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

[13] https://huggingface.co/sentence-transformers/msmarco-bert-base-dot-v5

[14] https://huggingface.co/sentence-transformers/msmarco-roberta-base-v3

[15] https://huggingface.co/allenai/specter

Model	Recall@1	Recall@3	Recall@5	Recall@10
all-mpnet-base-v2	0.097	0.226	0.319	0.497
msmarco-bert-base-dot-v5	0.133	0.270	0.369	0.549
msmarco-roberta-base-v3	0.114	0.243	0.341	0.520
multi-qa-mpnet-base-dot-v1	**0.140**	**0.313**	**0.418**	**0.585**
allenai-specter	0.064	0.180	0.276	0.457
e5-base-v2	0.119	0.264	0.356	0.533

표 2. Embedding model 후보들에 대한 Recall@k 평가

표 1, 2와 같이, 성능 평가 결과 multi-qa-mpnet-base-dot-v1가 모든 k에 대해서 Hits와 Recall 모두 다른 모델들보다 높았다. Research Assistant와 동일하게 Q-A 형식의 태스크를 위해 학습된 embedding model이면서, 다양한 Q-A 데이터에 학습되었기 때문에 generalization ability가 높아서 나온 결과라고 예상된다. embedding을 만들 만약 일관적이지 않은 결과가 나왔다면 지표들의 트레이드오프를 고려하여 모델을 선택하거나 여러 모델을 RAG 파이프라인 위에서 테스트해야했겠지만, 일관적이고 강한 결과가 나왔으며, 모델의 태스크/데이터 특성 또한 우리의 태스크와 잘 정렬되어있기 때문에, 우리는 Research Assistant에서 사용할 embedding model로 multi-qa-mpnet-base-dot-v1를 선택하였다.

- **Generation - Accuracy 평가**

우리는 Research Assistant의 답변 생성 능력을 3가지 축의 지표로 평가하기로 하였다. 얼마나 instruction을 잘 따라서 사용자에게 정확한 답변을 제공하는지를 나타내는 accuracy, 유저가 질의를 시작한 시점으로부터 답변이 표시되기까지 걸리는 시간인 latency, 그리고 부적절하거나 범위를 넘어가서 대답할 수 없는 질문에 대해서는 답하지 않고 대신 “I don’t know”와 같은 fallback 답변을 내보내는지 평가하는 error handling의 영역이 있다. 그 중에서도 Research Assistant의 핵심은 accuracy이다. 우리는 RAG pipeline의 accuracy를 가장 우선으로 평가하였다. 이 때, LLM을 RAG로 강화하는 것과 LLM을 Research Assistant에 맞게 학습하는, 우리가 진행한 두가지 액션이 각각 어떻게 긍정적인 기여를 했는지를 분석하기 위한 incremental한 평가를 진행하였다. (평가 방법은 8-C절의 “Evaluation - Accuracy” 부분 참고)

Retriever	Generator	Human Eval Rank (↓)	GPT-4 Eval Score (↑)	GPT-4 Evaluated Win Rate (%, ↑)
None	llama2-7b-chat-hf	3.00	3.450	baseline
multi-qa-mpnet-base-dot-v1	llama2-7b-chat-hf	1.63	4.355	65.6%

multi-qa-mpnet-base-dot-v1	Ours (DPO 학습)	**1.38**	**4.356**	**66.7%**

표 3. RAG 사용 여부 및 generator 학습 여부의 조합에 따른 답변 accuracy의 정량/정성 평가

- **Generation - Latency 평가**

아무리 정확한 답변을 내놓는다고 하더라도, 충분히 빠른 시간 안에 답변을 내놓지 못한다면 그 효용이 떨어진다. 따라서 우리는 accuracy 다음으로 latency를 중요하게 평가하였다. RAG는 기본적으로 LLM에 새로운 컴포넌트를 추가하는 것이기 때문에, LLM만 사용하는 baseline보다 latency가 증가할수밖에 없다. 우리는 이를 극복하기 위해 token streaming을 도입하였고, 실질적인 사용자 경험과 연관된 Time to Interactive[16] 관점에서의 latency를 최소화해 사용자 경험을 효과적으로 개선하였다. 구체적으로는, 하나의 token 당, 생성 시 평균적으로 0.008 초 가량의 시간이 소요됨을 측정하였다. 평균적으로 600 토큰 가량의 답변이 생성되는 것을 고려했을 때, 5초 가량의 latency 감소되는 효과를 얻을 수 있었다.

Method	**Time to Interactive Latency**	**Full Response Latency**
Llama2	6.24	
(+) RAG (600 token)	6.92	
(+) Token Streaming (600 token)	**3.11**	7.71

표 4. RAG pipeline의 latency

- **Generation - Error Handling 평가**

LLM 기반의 Research Assistant를 사용하는 데에는 두가지 위험성이 따른다. 첫번째로는, 유저가 부적절한 표현을 포함하는 질문을 했을 때, LLM 또한 부적절한 답변을 생성해서 응수하거나, 유저의 부적절한 발언에 동조하는 경우다. 두번째로는, LLM의 pretrained weight에 담긴 지식 또는 업로드해서 참조할 수 있는 지식의 범위를 넘어서는 질문을 했을 때, LLM이 hallucination이 포함된 답변을 반환해서 유저가 잘못된 답변을 얻게 되는 경우가 있다. 좋은 Research Assistant라면 두 경우 모두 자신이 답할 수 없다고 밝혀야 한다. 우리는 이러한 error handling 수준을 평가하기 위해 error를 일으킬 수 있는 입력들을 이용해서 Assistant를 공격하는 Red-Teaming 접근 방법을 사용했다. 혐오발언 데이터셋[17], 그리고 애초에 세상에

[16] https://developer.chrome.com/docs/lighthouse/performance/interactive

[17] Ona de Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. 2018. Hate Speech

존재하지 않는, 팀원들이 직접 지어낸 논문 제목들을 소재로 해서 두 가지 종류의 공격에 사용할 수 있는 질문들을 수집하였고, 실제로 Assistant에게 이러한 질문을 해서 나온 답변을 기록해서, “I don’t know”와 같은 fallback 답변이 올바르게 나오는 케이스의 비율을 계산하였다.

Method	**부적절한 질문 Fallback 비율 (%)**	**답할 수 없는 질문 Fallback 비율 (%)**
Baseline (Our RAG, basic prompt)	69.2%	0%
(+) Error Handling Prompt Engineering	**100%**	**54%**

표 5. Research Assistant가 예외 상황에 올바르게 fallback하는 비율

평가 결과, error handling을 고려하여 prompt engineering한 프롬프트를 적용한 우리의 Research Assistant가 초기의 프롬프트만을 적용한 baseline Research Assistant와 비교해서 부적절한 질문과 답할 수 없는 질문 모두 월등하게 높은 fallback 성능을 보여주었다. 특히 답할 수 없는 질문에 대한 fallback의 경우, baseline은 hallucination으로 엉뚱한 답변을 내놓았지만, prompt engineering된 research assistant는 절반 이상의 경우에 대해 옳게 fallback 답변을 내놓았다.

10. Division & Assignment of Work

모든 조원들이, 모든(프론트엔드, 백엔드, ML Research, MLops) 영역을, 다룰 수 있도록 분배하였다.

항목	담당자
Testing Embeddings	손동현, 이재영
Dataset Construction	손동현, 김상범
LLM Fine-tuning	김상범
Query UI 구현	김상범
Upload UI 구현	손동현, 이재영
Upload API 구현	김상범, 이재영
Query API 구현	손동현
Llama2 Setup	김상범
Milvus Setup	손동현, 이재영
Embedding Encoder Setup	김상범, 이재영
Prompt Engineering	손동현, 이재영

Dataset from a White Supremacy Forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 11–20, Brussels, Belgium. Association for Computational Linguistics.

11. Conclusion

우리는 entry-level의 NLP 연구자가 활용할 수 있는 Research Assistant를 개발하였다. 우선, 유저가 Research Assistant와 상호작용하기 위해 웹 프론트엔드 및 백엔드 서버를 구현하였다. 그리고 핵심 기능인 RAG 파이프라인을 서빙하기 위해 필요한 인프라를 구축하고, RAG 파이프라인에서 필요한 모델들을 서빙하기 위한 서버를 구축하였다. 우리가 개발한 Research Assistant는 RAG를 통해 유저가 업로드하는 최신 논문에 기반한 답변을 제공함으로써 LLM의 knowledge cutoff와 hallucination 문제에 맞섰다. 또한, 우리는 RAG 파이프라인을 개선하기 위한 연구를 수행하였다. Retrieval을 위한 최적의 embedding model을 찾았고, LLM을 Research Assistant의 용도에 잘 align되도록 SFT 및 DPO를 이용하여 학습하였다. Accuracy를 기준으로 한 정량/정성 평가 결과, 학습된 LLM은 baseline 대비 더 유용한 답변을 제공하였다. Accuracy뿐만 아니라 latency, error handling의 측면에서도 우리의 Research Assistant는 baseline보다 월등하였다. 또한 RAG 파이프라인의 각 구성요소가 더 유용한 답변을 제공하는 데에 점진적으로 기여함을 ablation study를 통해 보였다. 우리 팀원들은 Research Assistant 개발 프로젝트를 통해 최신 LLM 및 RAG 기법에 대한 깊은 이해를 갖출 수 있었으며, 개발된 Research Assistant는 우리 자신들과 같은 entry-level NLP 연구자에게 유용하게 사용될 수 있을것이다.

◆ [Appendix A] User Manual

GitHub 저장소 주소: https://github.com/DHdroid/2023f-cid1-rag-research-assistant-team-B

소스코드를 다운받은 뒤, README.md의 “Recommended dev setup” 섹션을 따라서 의존성을 설치합니다.

학습 및 평가 코드를 실행하기 위해서, 그리고 llama2 베이스 모델을 서빙하기 위해서는, 크레딧이 충전된 OpenAI 계정 및 meta-llama 라이선스에 동의하고 승인을 받은 Huggingface Hub 계정이 필요합니다. 계정을 미리 준비해주세요.

환경변수 관리를 위해서는 모듈별로 분리된 .env 파일을 사용합니다. 아래는 각각의 .env 파일별로 명시되어야 하는 값들입니다.

- **backend/.env**

  LLM_SERVER_ENDPOINT # TGI 서버의 엔드포인트. 예시) https://llm.my-gpu-server.com

  ENCODER_SERVER_ENDPOINT # 임베딩 서버의 엔드포인트.

  예시) https://encoder.my-gpu-server.com

  # Milvus에 연결하기 위해 필요한 host, port 정보. 백엔드와 같은 서버에 띄우는 경우에는 아래와 같이 설정한다.

  MILVUS_HOST="localhost"

  MILVUS_PORT="19530"

  TOKENIZER="meta-llama/Llama-2-7b-chat-hf" # TGI에 띄워진 모델과 호환되는, huggingface hub에 있는 tokenizer 이름

  STREAM_LLM_RESPONSE="true" # LLM 답변 스트리밍 사용. Frontend 없이 디버깅 목적으로 사용하는 경우에만 “false”로 세팅한다.

- **frontend/.env**

  BACKEND_SERVER_ENDPOINT="http://localhost:8001" # 백엔드 서버의 엔드포인트. 프론트엔드와 백엔드를 같은 서버에 띄우는 경우에는 localhost로 설정.

- **ml/.env**

  ENCODER_MODEL_NAME_OR_PATH="sentence-transformers/multi-qa-mpnet-base-dot-v1" # Embedding server를 띄울 때 사용됨. Huggingface Hub에 있는 모델 이름 또는 로컬 모델 폴더.

GPT_KEY # 학습 데이터 생성 및 평가를 위해 사용되는 OpenAI API 키. OpenAI 가입 및 결제 수단 등록 후, https://platform.openai.com/api-keys 에서 발급받은 키를 넣는다.

GPT_MODEL_TYPE="gpt-3.5-turbo-1106" # OpenAI API에서 사용할 모델 타입. 실험에 따라 gpt-3.5-turbo-1106 또는 gpt-4-1106-preview를 사용한다.

BACKEND_SERVER_ENDPOINT="http://localhost:8001" # RAG 파이프라인에 질의해서 DPO 데이터를 생성할 때 사용하는 백엔드 엔드포인트.

데이터 생성, 모델 학습, 평가를 위해서는 각각 ml/dataset_construction, ml/llm_training, ml/evaluation 아래의 스크립트들을 사용할 수 있습니다.

배포를 위해서는 GPU 서버를 2대 준비합니다. 그 중 하나에는 docker를 이용해서 TGI 서버를 실행합니다.

$ export volume=”/path/to/save/weights”

$ export model=”meta-llama/Llama-2-7b-chat-hf” # 또는, DPO 학습된 모델 경로

$ export hf_token=(huggingface hub에서 발급받은 token)

$ docker run --gpus all --shm-size 1g -p 8080:80-e HUGGING_FACE_HUB_TOKEN=$hf_token -v $volume:/data ghcr.io/huggingface/text-generation-inference:1.3 --model-id $model

나머지 하나에는 embedding server를 실행합니다.

$ cd ml

$ uvicorn encoder.serve_encoder:app --port 8002

의존성 및 .env 세팅, 그리고 ML 모델 서버 실행을 마친 후, Milvus, Backend, Frontend를 CPU 서버에서 실행합니다.

Milvus는 embedded milvus를 사용합니다. 실행에 앞서, 초기 1회 셋업이 필요합니다.

$ sudo mkdir -p /var/bin/e-milvus

$ sudo chmod -R 777 /var/bin/e-milvus

$ python -c "import milvus; milvus.before()"

이후 화면에 표시되는 가이드를 따라서 Milvus에 필요한 환경변수를 셋업합니다.

완료되면, backend/launch_milvus_server.py 를 실행합니다.

이제 Backend를 실행합니다.

$ cd backend

$ huggingface-cli login # 이후 지시에 따라 토큰 입력

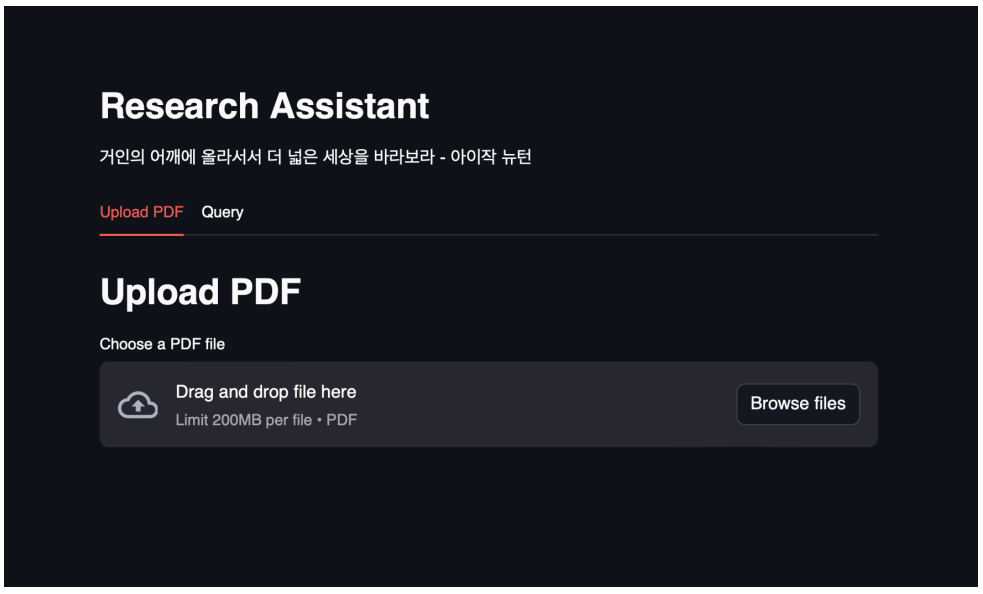
$ uvicorn main:app --port 8001

마지막으로 Frontend를 실행합니다.

$ cd frontend

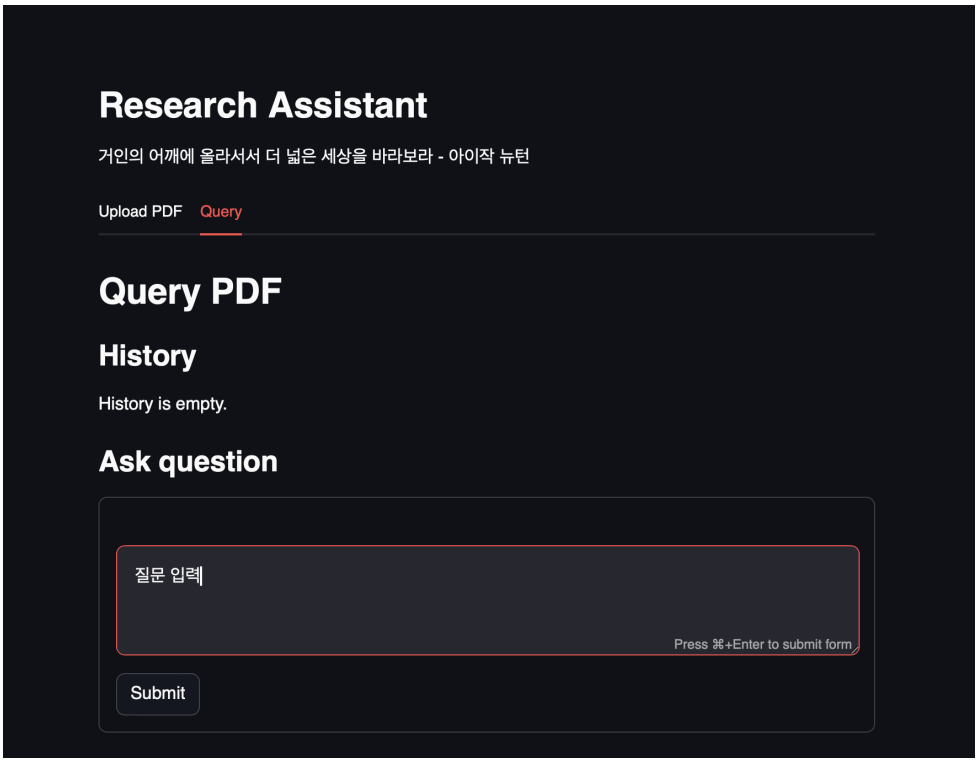
$ streamlit run main.py

축하합니다! Research Assistant의 셋업을 마치셨습니다. Streamlit 실행 시 자동으로 브라우저 창이 열리지 않았거나, CPU 서버에 GUI 접속이 되지 않는다면, http://CPU 서버 주소:8501 로 접속해주세요.



Research Assistant의 메인 페이지에서는 문서를 업로드할 수 있습니다. Browse files 버튼을 눌러서 PDF

파일을 선택하거나, 회색 영역에 파일을 드래그-앤-드롭하여 업로드할 수 있습니다.



파일을 업로드한 뒤에는 Query 탭으로 이동하여 Research Assistant에게 질문을 할 수 있습니다. Ask question 아래의 텍스트 상자에 질문을 입력한 뒤 아래의 Submit 버튼을 누르거나 Command + Enter 키를 누르면 질문이 전송됩니다. 이후 약 5초 기다리면 생성 및 토큰 스트리밍이 시작되어서, 답변 토큰들이 화면에 순차적으로 표시됩니다.

◆ [Appendix B] Prompts Used in the Work

본 프로젝트를 진행하는 과정에서 Research Assistant 자체에 / GPT에 사용하기 위한 다양한 prompt를 작성하였다. Prompt는 알려져 있는 prompt engineering 가이드를 반영하여 좋은 초안을 작성한 뒤, LLM에 질의한 후, 그 답변들을 팀원들이 세심히 관찰하여 prompt의 개선점을 도출하고 적용하는 loop를 반복하여 작성되었다. 이러한 과정을 거쳐 나온 귀중한 산출물이기에, 그리고 프로젝트의 재현가능성을 위해, 최종적으로 사용되었던 prompt들을 공개한다.

- Question Generation (Free-form Response)

```
As an entry-level (from undergrad to masters) NLP researcher, what
questions would you ask while reading "{paper_title}" paper? The questions
must be answerable within the paper text, without external knowledge or
extensive prior knowledge. Give 3 questions you might ask before reading
the paper, 3 questions regarding details, 3 questions about discussion.
```

```
Keep balance between general research questions and paper-specific
questions. Number the questions from 1 to 9.
```

- Question Generation (JSON Representation)

```
As an entry-level (from undergrad to masters) NLP researcher, what
questions would you ask while reading "{paper_title}" paper? The questions
must be answerable within the paper text, without external knowledge or
extensive prior knowledge. Give 3 questions you might ask before reading
the paper, 3 questions regarding details, 3 questions about discussion.
Keep balance between general research questions and paper-specific
questions. Respond in JSON with schema
{
"before_reading": List[str],
"details": List[str],
"discussion": List[str]
}
```

- Dataset Construction - Evaluation

```
The following are two candidate responses for question on research papers.
Your task is to find the better answer resembling "good response" criteria.
First analyze the responses, and give final answer in "Response #1 win" or
"Response #2 win" on separate line.

We define "good response" with following criteria.
- Truthfully provide information asked in the question
- Goes into details from paper and describe algorithms or processes in a
detailed manner
- Gives concrete and specific examples from paper
- Easily readable and understandable

In contrary, bad responses
- give incorrect or irrelevant information
- just rephrase the given question
- give abstract descriptions
- use complex jargon and incomprehensible

### Example 1

Paper: Attention Is All You Need
Question: How do the experimental results and evaluations presented in the
paper contribute to the overall argument for the effectiveness of the
Transformer model in comparison to other existing approaches?
Response #1: Experimental results showcase the Transformer's superior
performance on various sequence-to-sequence tasks, surpassing existing
models and setting new benchmarks in machine translation and other NLP
applications.
Response #2: Comparative evaluations highlight the scalability of the
Transformer, demonstrating consistent improvements in both efficiency and
accuracy across different datasets and tasks.
```

```
Response #1 just repeats the given question, while response #2 gives the
actual details on how the paper presented results.
Response #2 win

### Example 2

Paper: BERT: Pre-training of Deep Bidirectional Transformers for Language
Understanding
Question: Can you identify the key contributions of the paper based on the
abstract or introductory sections?
Response #1: The paper introduces a novel pre-training objective called
Masked Language Model (MLM) and demonstrates its effectiveness in capturing
bidirectional context.
Response #2: BERT contributes by presenting state-of-the-art results on a
range of NLP benchmarks, showcasing its versatility and impact on various
applications.

Response #1 give the true information that is asked, while response #2
answers about the results, missing the point of question asking the paper's
contribution.
Response #1 win

### Input

Paper: {paper_title}
Question: {question}
Response #1: {response_1}
Response #2: {response_2}
```

- Dataset Construction - Answer Generation

```
The following is a guideline for generating a "good response". As a
research assisant, your task is generate a "good response" to the research
question. ### Guideline We define "good response" with following criteria.
- Truthfully provide information asked in the question - Go into details
from paper and describe algorithms or processes in a detailed manner - Give
concrete and specific examples from paper **not neccessarily, if needed** -
Brief, Compact and easy to read and follow In contrary, bad responses -
give incorrect or irrelevant information - just rephrase the given question
- give abstract descriptions - use complex jargon and incomprehensible -
Complicated, unstructured, and too long to read ### Input Paper:
{paper_title} Question: {question} Answer the "Good Response" to this
question. Do not print "Good Response:" in front of your answer.
```

- Research Assistant Evaluation - Likert Scale

```
You will be given a research paper in NLP domain, a research question
regarding the paper, and the response to the question.

Your task is to rate the response on one metric.
```

```
Please make sure you read and understand these instructions carefully.
Please keep this document open while reviewing, and refer to it as needed.

Evaluation Criteria:
Usefulness (1-5):
The usefulness of the response is high if it
- truthfully provides information asked in the question
- goes into details from paper and describes algorithms or processes in a
detailed manner
- gives concrete and specific examples from paper
- is easily readable and understandable

In contrary, the uselfulness is low if it
- gives incorrect or irrelevant information
- just rephrases the given question
- gives abstract descriptions
- uses complex jargon and incomprehensible

Evaluation Steps:

1. Review the given paper carefully and identify the main topic and key
points.
2. Read the question and find the relevant parts from the paper. Check if
the response is informative for the user based on the contents of the
paper.
3. Assign a score for usefulness on a scale of 1 to 5, where 1 is the
lowest and 5 is the highest based on the Evaluation Criteria.
```

- Research Assistant Evaluation - Comparison for Win Rate

```
The following are two candidate responses for question on research papers.
Your task is to find the better answer resembling "good response" criteria.
First analyze the responses, and give final answer in "Response #1 win" or
"Response #2 win" on separate line.

We define "good response" with following criteria.
- Truthfully provide information asked in the question
- Goes into details from paper and describe algorithms or processes in a
detailed manner
- Gives concrete and specific examples from paper
- Easily readable and understandable

In contrary, bad responses
- give incorrect or irrelevant information
- just rephrase the given question
- give abstract descriptions
- use complex jargon and incomprehensible

You MUST choose one even though the answers are so close.

### Input

[Context given to assistant]
```

```
Paper: {paper_title}
Question: {question}
[Responses]
Response #1: {response_1}
Response #2: {response_2}
[Verdict]
```

- Research Assistant Production Prompt (template)

```
[INST] <<SYS>> {system_prompt} <</SYS>> Follow the guidelines to give "good
response" to the question, based on the given context.
### Context
{context}
### Question
{user_message}
(You should say you don't know if you've encountered unfamiliar topic.)
### Answer [/INST] Good response:
```

- Research Assistant Production Prompt (system prompt)

```
### Guideline
We define "good response" with following criteria.
- Truthfully provide information asked in the question
- Go into details from paper and describe algorithms or processes in a
detailed manner
- Give concrete and specific examples from paper **not neccessarily, if
needed**
- Brief, Compact and easy to read and follow

In contrary, bad responses
- give incorrect or irrelevant information
- just rephrase the given question
- give abstract descriptions
- use complex jargon and incomprehensible
- Complicated, unstructured, and too long to read

### Caution
You are a helpful, respectful and honest assistant. Always answer as
helpfully as possible, while being safe. Your answers should not include
any harmful, unethical, racist, sexist, toxic, dangerous, or illegal
content. Please ensure that your responses are socially unbiased and
positive in nature.

### IMPORTANT
You should only output exactly one good response to the question, without
any prefix or postfix.
```