

Open-Domain Question Answering 프로젝트

2024.10.02 ~ 2024.10.24 (3주)

본 프로젝트는 주어진 Open-Domain 질문에 대해 정확한 답변을 생성하는 모델을 개발하는 것을 목표로 한다. 네이버 커넥트재단 부스트캠프 AI Tech 7기 NLP 과정의 일환으로 진행되었으며, 질문에 대한 답변을 찾아내는 과정에서 최신 QA 모델 트렌드를 학습하고, 다양한 협업 도구를 활용하여 팀 내 원활한 의사소통과 의사결정을 경험하였다.

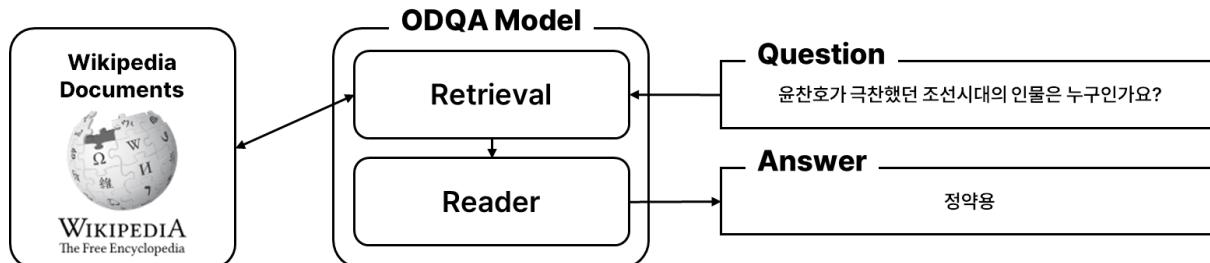
주요 성과

EM(Exact Match) 점수 66.11% 달성 (Baseline 33.06% 대비 33.05%p 개선)

Rank	Team Name	Team Member	EM	F1
My Rank 9	NLP_06조		66.1100%	77.3700%

모델 개발 및 접근 방식

ODQA 모델을 후보 문서를 검색하고 그 문서에서 정확한 답을 추출하는 두 단계 접근법을 중심으로 설계했다.



Passage Retrieval

- 다양한 접근 시도: 질문과 유사한 문서를 효과적으로 검색하기 위해 Sparse, Dense, Hybrid 접근법을 시도하였다.
- 결과: 실험 결과 Sparse 방식(BM25)이 데이터에 가장 적합하여 최종적으로 채택하였으며, Hit@10 기준으로 약 90%의 검색 성능을 기록했다.

MRC

- 모델 사용 및 최적화: 검색된 문서에서 정확한 답을 추출하기 위해 RoBERTa 기반 모델을 사용하고, 데이터 증강을 통해 성능을 개선하였다. 리더보드에서 EM 기준이 중점이므로 extractive 모델을 우선적으로 사용하였다.
- 생성형 모델 적용: extractive 모델의 맥락 이해 능력의 한계를 파악하고, 질문에 대한 답변 제공이라는 문제의 본질을 고려하여 Llama와 OpenAI GPT와 같은 생성형 언어 모델을 추가로 적용해 성능을 평가하였다.
- 앙상블 전략: 성능의 일관성과 정확도를 높이기 위해 hard-voting 방식의 앙상블을 수행했다.

협업 및 관리 도구 사용

Github

- 코드 스타일 통일: 커밋 컨벤션을 설정하고, pre-commit을 통해 코드 스타일의 일관성을 유지하였다.
- 효율적인 브랜치 관리: Git Flow 전략을 사용하여 브랜치를 관리하며, 작업 간 충돌을 최소화하였다.
- 이슈 및 PR 관리: 이슈와 PR 템플릿을 활용해 명확한 업무 분배와 코드 리뷰를 진행하였으며, 팀원 3인 이상의 확인과 승인 시에만 merge를 수행하였다.

Confluence

- 과정 기록 및 결과 공유: 시도의 배경, 구체적 방법, 결과, 개선 방향 등을 체계적으로 문서화하여 팀원들이 각 진행 상황과 차후 개선 사항을 쉽게 공유할 수 있도록 하였다.

개요

Open Domain Question Answering (ODQA)

현재 우리는 네이버, 구글 등 검색엔진에 찾고 싶은 것을 입력해서 원하는 정보를 쉽게 얻을 수 있다. Question Answering (QA)은 이러한 검색엔진과 유사한 Task로써, 사용자가 질문과 문서를 주고 인공지능이 문서를 기반으로 질문에 대답하는 Task이다. 그 중 Open-Domain Question Answering (ODQA)은 문서가 따로 주어지지 않고 미리 구축된 Knowledge Resource를 기반으로 질문에 대답할 수 있는 지문을 찾아내는 것이다. 본 프로젝트에서는 질문에 맞는 문서를 찾아내는 Retrieval 단계와 찾아낸 문서를 읽고 답변을 찾거나 생성하는 Reader 단계로 이루어진 Two-Stage ODQA 모델을 설계하고 Exact Match(EM)와 F1-Score를 기반으로 평가하였다.

팀 구성 및 역할

이름	담당 역할
서태영	데이터 전처리(Wikipedia Dataset), 데이터 증강 (GPT-Prompting)
오수현	DPR 실험(SBERT), Generative Reader 실험(GPT-Prompting), Extractive Reader 실험(KorQuad FineTuning)
이상의	데이터 전처리(Question, Answer Data), Hybrid Retrieval 실험
이정인	데이터 전처리(Context Data), 데이터 증강 (AEDA)
이정휘	Inference 구현, Extractive Reader 실험(RoBERTa, Multilingual BERT, KoELECTRA), Ensemble
정민지	Sparse Retrieval 실험(TF-IDF, BM25, Learned Sparse, BGE-M3, SPLADE), Generative Reader 실험(LLaMA-Prompting), github Issue, PR Template 작성,

프로젝트 목표

본 프로젝트에서는 네이버 부스트캠프 AI Tech 과정에서 학습한 내용을 실제 문제 해결에 적용하는 데 중점을 두었다. 주어진 문제를 이해하고 다양한 해결 방법들을 학습 및 체화하고, 특히 이전 프로젝트보다 더 체계적으로 협업하는 것을 주요 목표로 설정했다. 또한, 업무 분담을 명확히하여 각자 맡은 업무에 대해 책임감을 갖고자했다.

데이터셋 설명

Upstage에서 제공한 MRC 데이터와 위키피디아 문서들을 사용하였다. 총 4,192개의 학습 샘플과 700개의 평가 샘플로 구성되어 있다. 학습 샘플의 경우 모든 정보가 공개되어 있으며 평가 샘플은 id와 question만 공개되어 있다. 위키피디아 문서는 56,737건의 문서를 포함하고 있다.

분류	세부 분류	샘플 수	컬럼 명
Train_Datasets	Train	3,952	id, question, context, answers, document_id, title
	Validation	240	
Test_Datasets	Validation	240 (private)	id, question
		360 (public)	
Wikipedia	-	56,737	title, text, document_id

[표 1] 데이터셋 정보

평가 Metric

본 프로젝트는 Reader 모델에서 공백과 "."을 제외한 나머지의 단어가 모두 일치해야하는 Exact Match(EM)와 일부 분만 일치해도 점수를 받을 수 있는 F1-Score로 모델을 평가하였고, Retrieval 모델에서 특정 질문에 대해 상위 K개의 검색 결과 중 적어도 하나의 관련 문서가 포함되어 있는지를 측정하는 지표인 Hit@K와 상위 K개의 결과 내에서 얼마나 높은 순위에 위치하는 측정하는 지표인 MRR@K로 모델을 평가하였다.

데이터 분석

탐색적 데이터 분석(EDA)

ODQA의 전체적 데이터 흐름을 파악하고 원활한 데이터 처리를 위해 EDA 실시하였다.

ODQA의 시스템의 구조는 Retriever와 Reader로 나눴고 활용할 데이터셋의 구성으로

(wikipedia, train, validation, test)로 구성된 것을 확인하였고 활용한 데이터는 위키피디아의 덤프 파일로 title 키워드를 위키피디아에 검색 후 중제목/소제목 제목을 기준으로 데이터를 가져왔다고 판단했다.

구조	Wikipedia	train	validation	test
Data	60613	3952	240	600
필드	문서	QA쌍, 문서	QA쌍, 문서	test

[표 2] 데이터셋 구조

wikipedia에서의 text문서 확인 결과 QA쌍의 train, validation은 wiki의 text를 가져와 context를 이룬 것을 확인하였고 wiki의 text를 전처리시 QA쌍의 데이터셋들도 전치리를 해야 한다는 것을 확인했다.

중복 데이터	wiki 전체데이터	passage 전체중복	title 동일 중복	title 비동일 중복
Data	60613	7677	7586	91

[표 3] Wikipedia에 대한 분석 (중복 데이터 및 전처리 해야할 데이터)

중복된 passage가 존재 하는 것을 파악하였고 전체 중복된 내용을 제거한 wiki 데이터를 활용하여 validation 과정을 거쳐 보았는데 train, validation의 passage 위치를 확인하는 document_id에 대한 위치 데이터의 삭제로 인한 평가 저하로 인해 다시 원본 wiki 데이터를 활용하기로 결정했다.

개행(\n), 마크다운 문장	'*(a) 영속적인 국민\n'* 성좌의 명칭으로 유엔에서 국제 승인을 받은 국가: 바티칸 시국\n'
수식	'::x×(y×z) + y×(z×x) + z×(x×y) = 0\n'* $ x \times y = x \cdot y - (x \cdot y)$ '
미흡한 파싱, 문서링크	'홍영식, 민영익, 서광범, 중국인 통역 우리탕.]]\n'* '(책) label=산술 en Arithmetica(3세기)'
띄어쓰기	'상황이 이렇다 보니 프로축구 현대 호랑이는 홈 앤 어웨이 제도로 바뀐 한편 주말 2연전이 새롭게 도입된 동시에 프로야구의 선례를 본따 '

[표 4] 전처리가 필요한 데이터 종류

wiki의 text 확인결과 원활한 평가를 위해서는 개행과 띄어쓰기는 필수적으로 전처리 필요하다 판단했고 다양한 전처리 wiki 파일을 만들어 평가 데이터를 뽑는 시도를 해보기로 결정했다.

context	question	answer
그리고리 신부(Father Grigori)는 게임 하프라이프 2의 레이븐홈 챕터에	고든 프리맨이 레이븐홈을 떠날 수 있도록 도와준 사람은?	그리고리 신부 (Father Grigori)
에릭 레이먼드가 쓴 <성당과 시장>(The Cathedral and the Bazaar...)	에릭 레이먼드의 자유 소프트웨어 철학을 대변하는 글은?	<성당과 시장> (The Cathedral and the Bazaar)

[표 5] train 데이터 (Answer 외래어 분석 및 QA의 관련성)

train 데이터의 context 문장을 확인해본 결과 answer를 뽑아오는 경우 answer의 필요없다고 판단되는 음차어와 음역어 까지 추출되는 것이 확인되어 외래어를 제거한 버전의 평가 데이터를 추출하기로 결정했다.

데이터 전처리 및 증강

데이터 전처리

train 데이터셋의 context 데이터에 불필요한 문자들이 많아 answer를 내뱉을 때 불필요한 문자도 같이 내뱉는 경향이 있어 전처리를 수행했다. 주로 개행, 마크다운과 같은 크게 의미가 없는 문자들에 대해 전처리를 수행했다.

개행

train 데이터셋의 context에는 약 28000개 정도, wikipedia_documents 데이터셋의 text에서는 약 29000개 정도의 개행 \n 이 나왔다. 그래서 개행 전체를 공백으로 대체를 했다. 공백으로 할 경우에는 개행이 연달아 나올 경우 공백이 여러개 생긴다. 이런 경우를 생각해 공백이 하나 이상 붙어 있으면 하나로 바꿔주는 식으로 했다.

기존 문장	변형 문장
사법관시보가 된 것을 시작으로 법조계에서 근무를 시작 했다.\n\n1939년에는 경성지방법원 검사대리가 되었고,	사법관시보가 된 것을 시작으로 법조계에서 근무를 시작 했다. 1939년에는 경성지방법원 검사대리가 되었고,

[표 6] 개행 데이터 전처리

마크다운

wikipedia_documents 데이터셋의 text에서 마크다운 문법이 많이 들어가 있었다. 제목(#), 리스트(*) 등 7개 종류의 마크다운을 re 라이브러리를 통해 정규표현식으로 전처리를 수행했다.

기존 문장	변형 문장
* 가이세이 역전 공원	가이세이 역전 공원
* 가이세이 역전 제2공원	가이세이 역전 제2공원
* 아미 티 가이세이	아미 티 가이세이

[표 7] 마크다운 데이터 전처리

외래어

wikipedia_documents와 train 데이터셋의 context에 다양한 외래어가 존재했다. 직접 확인한 언어가 10개 정도가 된다. (한국어, 영어, 한자, 일본어, 아랍어, 아이슬란드, 베트남어, 힌디어, 프랑스어) 대부분이 명사 뒤에 괄호와 함께 번역된 단어가 나오는 식이였다. (예시 : 나리타처럼 되지 말자(成田のようにならないようにしよう)) 큰 의미가 없다고 판단을 해서 전처리를 수행했다. 정규 표현식을 사용했으며, 괄호 사이에 한글, 숫자, 공백이 아닌 경우에는 괄호와 함께 전처리가 될 수 있도록 구성했다.

기존 문장	변형 문장
젊어서는 허자장, 번자소(樊子昭)와 함께 명성이 있었다. 후한 말, 효렴으로 천거되...	젊어서는 허자장, 번자소와 함께 명성이 있었다. 후한 말, 효렴으로 천거되...

[표 8] 외래어 데이터 전처리

데이터 증강

train 데이터셋의 경우 3900개 정도로 많은 편이 아니었다. 그래서 증강을 수행했고, context를 증강시키는 방법으로 AEDA를, question을 증강하는 방법으로 프롬프팅을 사용했다.

Prompting

GPT 모델을 사용해서 데이터 증강을 시도했다. train 데이터셋의 context를 알려준 후, 알려준 context에 대한 질문과 질문에 해당하는 답을 내도록 명령했다. 몇개의 데이터셋이 더 성능이 좋은지 확인하기 위해 약 7500개로 증강한 데이터셋과 약 5000개 정도로 맞춘 데이터셋을 비교를 수행했다. 약 7500개로 증강한 데이터셋의 성능이 약 5000개의 데이터셋의 성능보다 조금씩 더 좋은 결과를 얻을 수 있었다.

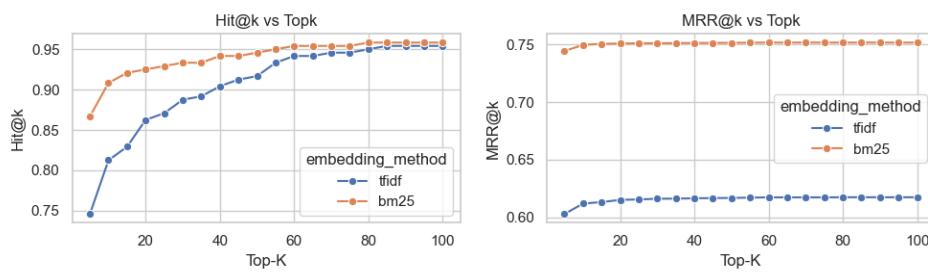
Retrieval 모델

본 모델 구조에서는 질문에 대한 답변을 포함한 문서를 정확히 검색해 reader 모델로 전달하는 것이 중요하다. 이를 위해 sparse, dense, hybrid 방식을 시도하여 성능을 비교했고, validation 데이터셋을 통해 Hit@K와 MRR@K 지표로 평가했다. 최종적으로 가장 우수한 성능을 보인 방법을 검색기로 사용하였다.

Sparse Embedding Retrieval

Sparse 방식은 각 토큰의 빈도에 기반한 전통적인 검색 방법으로, TF-IDF와 BM25가 대표적이다. 본 프로젝트에서는 질문과 정답 문서 간 용어 일치가 많고 문서들 간의 길이 차이가 커서, 문서 길이를 보정하는 BM25가 TF-IDF보다 우수한 성능을 보였다. BM25는 문서 길이와 빈도를 고려해 긴 문서에서도 주요 키워드에 더 집중된 결과를 반환할 수 있어, 질문과 정답 문서의 높은 일치도를 효과적으로 반영했다.

BM25의 성능을 Top-K=10으로 설정했을 때 Hit@10은 90.83%, MRR@10은 74.97%로, TF-IDF보다 우수한 결과를 기록했다 ([그림 1], [표 1]). 주요 파라미터($b=0.75$, $k_1=1.0$)와 konlpy.tag의 Mecab 토크나이저를 활용해 최적화한 결과, BM25가 높은 검색 성능을 제공함을 확인할 수 있었다. 이는 BM25가 질문과 정답 문서 간의 정확한 용어 일치도를 잘 반영하기 때문이다.



[그림 2] TF-IDF와 BM25의 Top-K에 따른 성능 지표 변화 (Hit@K, MRR@K)

Sparse Retrieval	Hit@10	MRR@10
TF-IDF	0.8667	0.6077
BM25	0.9083	0.7497

[표 9] 최적 설정에서의 TF-IDF와 BM25 성능 비교 (Top-K=10)

Dense Passage Retrieval

Dense Embedding 방식은 Sparse Embedding보다 낮은 차원의 벡터를 사용하며, 학습을 통해 텍스트 데이터의 의미를 보존할 수 있는 방식이다. 지문을 일일이 질문과 연결하지 않고 Sentence Transformer 모듈을 사용하기 위해 Bi-Encoder 방식으로 실험하였으며, 한국어 SBERT 모델 중 vocab의 크기가 비교적 큰 snunlp/KR-SBERT-V40K-klueNLI-augSTS를 사용하였다. 결과적으로 [표2]와 같은 점수를 확인할 수 있다. K값이 증가할 수록 Hit@k는 꾸준히 증가하였으며, MRR@K는 안정적인 형태이다. 제공받은 데이터셋은 지문을 보고 질문을 생성한 경향이 있기 때문에 질문과 지문의 의미론적 관계보다 키워드가 얼마나 겹치는지를 확인하는 것이 중요하다. 따라서 Sparse Embedding과 비교하여 Dense Embedding만으로는 좋은 결과를 얻기 어려웠다.

Dense	k = 10
Hit@k, MRR@k	0.5209, 0.3527

[표 10] k=10일 때 SBERT의 validation 성능

Hybrid (Two-Stage Retrieval)

Two-stage retrieval 방식은 sparse retrieval과 dense retrieval의 장점을 결합한 방법이다. 우선 Sparse retrieval로 단어 수준에서 질문과 관련성이 낮은 문서들을 제거한다. 그 다음 남은 문서들 중에서 다시 dense retrieval을 reranker로 사용하여 문맥 수준에서 질문과 관련성이 높은 문서들을 최종적으로 추출한다. 표에서 보듯이 TF-IDF와 sbert를 각각 개별적으로 수행한 것보다 그 둘을 결합한 two-stage retrieval이 더 높은 성능을 보였다.

Model		HIT@10	MRR@10
stage-1	stage-2		
TF-IDF		0.6365	0.3730
sbert		0.4609	0.2504
TF-IDF	sbert	0.6999	0.4004

[표 11] sparse, dense, two-stage method 비교

최종 선택 Retrieval 모델

validation 데이터셋을 사용하여 각 방식의 성능을 비교한 결과 BM25가 retrieval을 가장 잘 수행하는 것을 확인할 수 있다. 이는 제공받은 데이터셋의 질문들이 지문을 보고 만든 형태이기 때문에 키워드가 겹치는 정도를 통해 문서를 탐색하는 것이 좋은 방식임을 의미한다.

Reader 모델

Reader는 Retrieval에서 질문에 맞는 k개의 문서들을 입력받아 정답을 잘 예측해야하는 것이 중요하다. ODQA에는 정답의 시작 위치와 끝나는 위치의 토큰을 예측하여 실제 문자로 복원하는 Extractive 방식과 실제 문자 자체를 생성하는 Generative 방식으로 나누어져있다. 본 모델에서는 각각의 방식에 대해 Train 데이터로 학습하고 Validation 데이터를 사용하여 EM, F1-Score로 모델을 평가한다.

Extractive

Extractive 방식은 Retrieval에서 추출한 K개의 문서들을 공백으로 잊고 질문에 맞는 답의 시작 지점 토큰과 마지막 지점 토큰을 예측하는 방식이다. 따라서 각 단어가 양방향으로 다양한 의미를 파악할 수 있는 BERT계열의 모델이 사용된다. BERT계열 모델의 입력값은 질문과 지문에 대해 [SEP] 토큰으로 사용해 하나의 문장으로 주어진다. 시도한 모델은 아래와 같다.

1. klue/bert-base

초기 베이스라인 코드에 시도한 모델인 klue/bert-base이다. 해당 모델은 기본 BERT 모델로써 다양한 모델과의 비교를 위해 초기 테스트 모델로 사용하였다.

2. uomnf97/klue-roberta-finetuned-korquad-v2

기존 BERT 모델에 비해 더 긴 문장과 많은 문장들을 학습해 Robust하게 최적화한 BERT 모델입니다. EDA를 통해 데이터 셋의 문서 길이가 약 2,000글자가 넘어가는 경우가 존재했고, 따라서 긴 문장이 사전학습된 RoBERTa 모델이 효과적일 것으로 판단하였다. 추가로, 본 프로젝트의 Task와 비슷한 데이터셋인 KorQuad로 사전학습된 모델을 사용하여 FineTuning 하였다.

3. monologg/koelectra-base-v3-finetuned-korquad

예측한 문장을 생성해내는 Generator와 생성해낸 문장의 토큰들이 원래의 토큰과 일치하는지 판별하는 Discriminator로 구성되어 있는 모델로, BERT에 비해 모든 토큰들을 다 학습할 수 있는 장점을 가지고 있다. 또한, 다른 벤치마킹에서도 RoBERTa와 비슷한 성능을 보였기에 시도해보았다.

4. Leepaper/bert-multilingual-Korquad-v1-fintuned

EDA 결과에서 다수의 데이터에서 다른 나라의 언어가 있음을 보였다. 특히 한자와 일본어 등이 있었고 그 외 많은 언어들이 존재했다. 해당 언어들은 한국어에 특화된 토크나이저를 사용시 [UNK]로 처리되며 이는 학습에 영향을 미칠 것으로 판단하였다. 따라서 Multilingual 모델을 사용하여 다른 나라의 언어까지 토큰화할 경우 성능이 향상될 것으로 생각하였다. 해당 모델은 KorQuad v1으로 finetuning하고 Huggingface에 모델을 저장한 뒤, 다시 모델을 불러와 Train dataset으로 모델을 학습하였다.

최종 선택한 Reader 모델

시도한 모델에서의 Inference의 결과는 다음과 같다. Retrieval는 가장 좋은 성능을 보인 BM25를 사용하였다.

Reader	EM	F1
klue/bert-base	52.5	63.14
uomnf97/ klue-roberta-finetuned-korquad-v2	64.58	75.72
monologg/ koelectra-base-v3-finetuned-korquad	48.33	54.63
Leepaper/ bert-multilingual-Korquad-v1-fintuned	-	-

[표 12] Reader 성능비교

Multilingual 모델의 경우 Reader 자체 성능 EM이 45.41로, 다른 모델보다 30이상 차이가 발생하여 inference 과정에서는 제외하였다. 따라서, 이 중 가장 좋은 성능을 보인 RoBERTa 모델을 채택하였다.

Reader 모델

Generative MRC

Extractive 모델이 정답을 잘 추출하지 못하는 경우를 분석했고, 질문에 특정 키워드가 문단 내 먼저 등장하면 해당 구문을 답변으로 반환하거나, 질문과 유사어가 포함된 경우 이를 정확히 이해하지 못하는 사례를 확인했다. 이는 extractive 모델이 문맥보다는 키워드를 기준으로 답변을 추출하기 때문으로 판단했으며, 이 문제를 해결하기 위해 문맥을 이해하고 더 자연스러운 답변을 생성할 수 있는 생성형 모델을 도입하였다.

OpenAI GPT-4o-mini Prompt Tuning

현재 가장 일반적으로 사용되며, 문맥 이해 능력이 뛰어나고 비용이 저렴한 GPT-4o-mini를 사용하여 Prompt Tuning을 수행했다. 시스템 프롬프트를 통해 MRC를 수행할 것임을 알렸고, 제한사항을 통해 출력력을 튜닝했다. few-shot 프롬프트 기법을 적용하여 일반적인 예시와 헷갈릴 만한 예시를 함께 입력했다. 대체적으로 답변들은 수식어를 포함한 긴 구의 형태였으며, 단순한 형태로 출력하기 위해 질문과 답변을 연결하였을 때 불필요하게 겹치는 수식어들은 제외하도록 지시했다. 결과적으로 답변의 형태는 간결해졌으나, 지문에서 추론이 필요한 경우, '두 번째', '마지막'과 같이 사건의 순서와 인과관계를 파악해야 하는 경우에 대해서 잘 응답하지 못하는 문제점이 있었다.

질문	문맥	정답	예측
오래플린과 부스의 마지막 계획에 따르면 그들은 어디서 링컨을 납치하려고 했는가?	오래플린은 부스의 초기 동료 중 한 명이었다. 1864년 가을에 오래플린은 링컨 대통령 납치 기도 음모의 공범이 되기로 한다. 그는 부스와 워싱턴 DC에서 부스의 돈으로 시간을 보내기 시작했다. 1865년 3월 15일, 오래플린은 부스와 다른 공범들과 펜실베이니아 가에 있는 고티어의 식당에서 만나서 대통령의 납치 가능성에 대해서 의견을 나눴다. ... (중략) ... 이번에는 링컨을 포드 극장에서 사로잡아 리치먼드로 데려가는 것이었다. 그러나 부스는 동료들에게 이 계획이 실현 가능할 것이라고 납득을 시킬 수는 없었다.\n\n오래플린에 따르면, 이것이 부스와 그들이 한 마지막 공모였다. ... (후략)	포드 극장	고티어의 식당

[표 13] GPT-4o-mini의 순서 정보 인식 불가 문제

Llama 3.1 8B Instruct 모델 Instruction 튜닝

Llama 3.1 8B Instruct 모델에 Q-LoRA를 적용한 Instruction tuning을 수행하여 모델이 질문 의도에 맞는 답변을 생성할 수 있는지 확인했다. 목표했던 예제에서 상당히 정확한 답변을 도출하였으나, 답변이 문장 형태로 반환되는 문제가 지속되었다. 이 문제를 해결하기 위해 우리 데이터와 유사한 KorQuAD v1.0의 일부(약 10%)를 추가하여 총 10,554개의 데이터로 fine-tuning하는 실험을 추가로 진행했으나, 문장형 답변 반환 문제는 여전히 해결되지 않았다. 이는 모델이 QA 작업의 특성상 짧고 정확한 답변보다는 문장 형태의 응답을 선호하는 경향 때문에 판단된다.

질문	문맥	정답	extraction 예측	llama 예측
문법 측면에서 더 보수적인 포르투갈어 표준은?	포르투갈어와 유사한 스페인 갈리시아 지방의 갈리시아어가 존재하나, 포르투갈어의 표준규범은 브라질 포르투갈어와 유럽 포르투갈어를 들 수 있다. 두 포르투갈어 표준은 발음 면에서 현저한 차이의 변화가 있으며, 문법 면에서는 대명사 체계에 차이점이 존재한다. 또한 브라질 포르투갈어는, 문법 면에서 덜 보수적으로 간주된다....(후략)	유럽 포르투갈어	브라질 포르투갈어	유럽 포르투갈어 표준이다.

[표 14] Extractive vs. Generative QA 모델 답변 예시

생성형 모델 적용에 대한 결론

생성형 모델은 문맥을 이해하는 능력에서 강점을 보였으나, 짧은 구로 답변해야 하는 본 프로젝트의 특성과는 다소 불일치하는 결과를 보였다. 특히 답변 형식이 고정되지 않거나 불필요한 수식어가 포함되는 등, QA 태스크에서 정답을 정확히 추출하기에 제약이 있었다. 그러나 생성형 모델을 통해 context 기반으로 정확한 답을 생성할 가능성을 확인할 수 있었으며, 본 태스크에 적합한 후처리 모듈을 추가하거나 추가적인 Instruction tuning을 수행한다면 의미 있는 성능 향상이 가능할 것으로 보인다.

결과

ODQA 수행 결과 분석

앞서 반복적으로 언급한 바 있으나, 제공받은 데이터셋의 질문은 지문을 보고 생성한 형태이기 때문에 범용적으로 사용될 수 있는 키워드가 다수 포함되어 있다. 이를 위해 BM25 방식을 사용한 Sparse Retrieval 기법을 적용하였으며, 그 결과 Dense Retrieval 방식에 비해 약 40% 높은 성능을 보여준다.

탐색을 통해 추출한 문서들을 바탕으로 Reader를 수행하였다. 이때 Extraction 기반과 Generation 기반의 방식을 모두 수행하였으며, 최종적으로 모델이 대체적으로 맞히지 못하는 데이터들의 사례는 아래와 같다.

1. 질문의 키워드가 문서의 앞에 등장하는 경우, 해당 키워드를 답으로 출력하는 경향이 있다.
2. '두 번째', '마지막'과 같이 순서나 인과관계 파악, 맥락적 추론에 취약하다.

사례	질문	문서	정답	예측값
1	멘데스가 요원들을 구하기 위해 간 도시는 어디인가?	(전략) 멘데스는 오타와의 캐나다 정부 관계자와 긴밀히 협력했고 ... (중략) ... 멘데스는 구출 작전을 지원하는 요원 한 명과 함께 테헤란에 도착하였고, (후략)	테헤란	오타와
2	오래플린과 부스의 마지막 계획에 따르면 그들은 어디서 링컨을 납치하려고 했는가?	(전략) 오래플린은 부스와 다른 공범들과 펜실베이니아 가에 있는 고티어의 식당에서 만나서 ... (중략) ... 포드 극장에서 사로잡아 리치먼드로 데려가는 것이었다. (중략) 이것이 부스와 그들이 한 마지막 공모였다. (후략)	포드 극장	고티어의 식당

[표 15] QA 수행 결과

해당 사례들을 개선하고자 Generation 기반의 방식 중 프롬프트 엔지니어링을 통해 추가적인 지시사항을 입력했으나, 사건의 순서 파악에 있어서 한계가 보였다. 결과적으로 Exact Match 점수가 보다 높았던 Extraction 기반의 방식을 채택하여 양상블을 수행했다.

Ensemble

성능의 일관성과 정확도를 향상시키기 위해 결과들을 Hard Voting 방식으로 양상블을 수행했다.

inference를 수행 후 저장되는 Predictions는 Test dataset의 ID와 질문에 맞는 답변이 기록되어 있다. 각 답변들을 취합하여 가장 많은 수의 답변으로 Hard Voting하여 최종 예측 답변으로 결정한다. 본 프로젝트에서는 데이터 전처리한 버전과 증강 비율에 따른 결과들을 기반으로 양상블하였다.

최종 모델 평가

Reader	Retrieval	Dataset	EM	F1
uomnf97/ klue-roberta-finetuned-korquad-v2	BM25	V0.1	58.75	67.48
		V0.1 + QG Augmentation (1:1)	58.75	67.13
		V0.1 + QG Augmentation (1:0.3)	57.50	65.95
		V0.2 + QG Augmentation (1:1)	58.33	67.47
		V0.2	48.33	54.63

[표 16] Validation Inference 결과 (Ensemble에 사용될 모델)

위 표를 기반으로 양상블을 수행한 결과 EM은 **65.42**, F1은 **76.39**를 기록하였다.

협업 방식

Github

이전 프로젝트에서 협업 시 다른 팀원의 작업 현황을 파악하기 어렵다는 피드백이 있었고, GitHub의 기능을 충분히 활용하지 못한 점에 대한 아쉬움이 있었다. 이에 이번 프로젝트에서는 GitHub의 Issue와 PR 기능을 적극 활용해 체계적인 협업을 실천했다.

모든 작업은 Issue로 등록하여 추적하고, 개별 개발 작업은 각 Issue에 맞는 Feature 브랜치를 생성해 수행했다. Commit은 Udacity commit convention을 기반으로 설정한 규칙에 따라 일관되게 관리했으며, Issue 템플릿과 PR 템플릿을 활용해 이슈와 PR의 형식 또한 표준화했다. ([그림1]과 [그림2] 참조)

특히, PR은 최소 3인의 리뷰와 승인을 거쳐야만 병합되도록 팀 규칙을 설정해 코드의 일관성과 품질을 유지하는 데 중점을 두었다. 이 과정에서 발생한 이슈와 PR은 각각 19개와 18개에 달하며, GitHub의 기능을 충분히 활용한 덕분에 팀원들이 각자 담당 업무와 진행 상황을 명확히 파악할 수 있었다.

[그림 3] GitHub Issue 예시
각 작업을 세부적으로 정의해 팀원들이 이해할 수 있도록 등록

[그림 4] GitHub Pull Request 예시
코드 리뷰를 위해 작업 배경, 실행방법, 주요 변경 사항 등을 설명

Confluence

[그림 5] Confluence 문서 예시

Confluence를 활용한 체계적인 프로젝트 기록 관리도 이번 프로젝트의 중요한 협업 방식 중 하나였다. 이전 프로젝트에서 각 작업의 상세한 기록이 필요하다는 점을 경험한 후, 팀 모두가 문서화의 중요성에 공감하며 이번 프로젝트에 더욱 체계적으로 기록을 수행했다.

이번 프로젝트에서는 기본 설정, 데이터 분석, 전처리, 성능 개선, 결과 분석 등을 포함해 총 51개의 문서를 작성하여 팀 전체가 쉽게 이해할 수 있도록 체계화했다. 각 문서에는 작업의 목적, 진행 방법, 결과 분석, 향후 계획이 포함되었으며, 작업이 완료될 때마다 관련 문서를 즉시 업데이트해 실시간으로 진행 상황을 공유했다. 특히, 매일 오전과 오후 두 차례 진행된 팀 회의는 각자 Confluence 페이지를 기반으로 이루어졌으며, 이를 통해 팀원들은 작업 현황을 명확히 파악하고 소통할 수 있었다. 이러한 체계적인 기록은 프로젝트 전반의 흐름을 이해하고 실험 의도를 명확히 파악하는 데 큰 도움이 되었다. 또한, 기록된 문서를 통해 실험 세부 사항을 쉽게 참조할 수 있어 향후 작업 계획 수립에도 유용했다.

팀 회고

프로젝트 팀 목표

우리 팀은 프로젝트 시작 시 "체계적으로 해보고 싶은 것을 모두 해보자"라는 목표를 세웠다. 이 목표는 ODQA라는 본 프로젝트의 문제 해결 과정 전반을 효율적이고 체계적으로 접근하고자 하는 의도를 내포한다. 또한 RAG와 연관지어 현업의 트랜드를 경험해보고자 하는 의욕 또한 담겨있다. 이를 위해 ODQA의 Retrieval, Reading Comprehension 단계에서 다양한 방식의 실험을 통해 여러 접근법의 장단점을 파악하고자 했다. 또한 GitHub과 Confluence를 사용해 코드 관리와 실험 기록을 투명하게 관리하여, 팀원들이 실험 의도와 결과를 언제든지 쉽게 파악하고 공유할 수 있도록 했다.

시도 및 결과

모델링에서는 다양한 접근 방식을 시도해 보며 QA 문제에 대한 폭넓은 이해를 쌓았다. Retrieval 단계에서는 sparse, dense, hybrid 방식을 실험했고, MRC 단계에서는 extractive와 generative 모델(Llama, GPT)까지 활용해 보았다. 최종적으로 sparse retrieval + extractive reader 조합이 리더보드에서 가장 우수한 성능을 보였으나, 다양한 실험을 통해 최신 QA 트렌드인 RAG 방식을 이해하는 데도 큰 도움이 되었다.

협업에서는 GitHub 활용 규칙을 정하고 모든 작업을 Issue로 등록하여 체계적으로 진행했다. 또한, Issue와 PR에 템플릿을 도입해 작업과 기록의 통일성을 높였고, PR은 팀원 3인 이상의 검토를 거치도록 설정하여 코드 품질을 유지했다. Confluence에 작업 목적과 세부내용, 결과, 다음에 시도할 방법 등을 기록해 총 51개의 문서를 작성함으로써 프로젝트 진행 상황을 팀원 모두가 명확히 이해하고 공유할 수 있도록 했다.

좋았던 점과 배운 점

이번 프로젝트에서는 협업과 기록이 특히 돋보였다. GitHub과 Confluence를 이전 프로젝트보다 활발히 활용하여 작업 흐름과 결정 과정을 투명하게 유지할 수 있었고, 다양한 QA 접근 방식을 실험하며 기술적 이해도를 넓힐 수 있었다. 이러한 경험은 추후 유사 프로젝트에서 큰 자산이 될 것으로 기대된다.

베이스라인으로 제공된 Retriever와 Extraction 기반의 Reader만을 사용하는 것에 그치지 않고, DPR과 Hybrid 방식, Generation 기반의 접근을 수행함으로써 현업의 트렌드를 경험할 수 있어 좋았다. 순위에 집중하는 것보다 다양한 방법론을 경험할 수 있는 기회가 되었다.

아쉬운 점

- 프로젝트가 본격적으로 시작되기 전 데이터 분석을 충분히 수행하지 못한 것이 아쉽다. 자연어, 특히 한국어에 있어서는 데이터의 특성과 모델의 출력 경향 등을 직접 눈으로 확인하고 분석하여 EDA를 수행하고 이에 맞춰 전처리와 증강을 수행해야 한다. 이러한 과정을 철저히 수행하지 못하고 본격적으로 프로젝트에 도입한 아쉬움이 있다.

- 기존 베이스라인 코드 구조에 매몰되어 있었던 점이 아쉽다. 새롭게 구현된 Generative Reader 방식이나 Dense, Hybrid Retrieval 방식과는 달리 Extractive Reader는 제공된 베이스라인 코드에서 크게 변경하지 않는 방향으로 설계하였지만 오히려 더 복잡하게 설계된 점이 있다. 그래서 Inference 코드는 최대한 새로 작성하였으나, 시간이 부족해 제대로 설계하지 못했던 점이 아쉬웠다.

- 각 실험 결과에 대한 심층 분석이 부족했던 점이 아쉬움으로 남았다. 실험 결과를 체계적으로 분석해 후속 실험 방향을 설정하는 데 더 많은 시간을 투자할 필요가 있었다. 또한, reader 부분에서 extractive 모델의 구조를 적극적으로 개선하거나, generative 방식의 후처리 로직을 보완하는 시도가 부족했던 점도 아쉬웠다.

향후 발전 계획

다음 프로젝트에서는 실험 결과에 대한 분석을 더 심화하고, generative 방식을 보완해 QA 시스템의 성능을 개선할 것이다. 특히 후처리 로직을 추가해 generative 모델이 평가 기준에 부합하는 답변을 더 잘 생성할 수 있도록 할 예정이다. 덧붙여, 이번 프로젝트에서 구축한 체계적인 협업 방식은 다음 프로젝트에서도 계속 발전시켜 나갈 것이다.

개인 회고 - 서태영

프로젝트 개인 목표

지난 프로젝트에 대한 아쉬움을 많이 채우고 싶었다. confluence나 github 등을 활용한 협업을 하면서 프로젝트에 조금 더 적극적으로 참여하는게 목표였다. 이번 프로젝트에서는 나름 confluence나 github을 활용한 협업을 열심히 하려했고, 지난 프로젝트 보다는 스스로가 느끼기엔 조금 더 적극적으로 했다고 생각한다.

그 다음 개인적인 목표는 모델쪽을 조금 해보고 싶었다. 코딩을 조금 하는식으로.. 하지만 초반에 다같이 한 데이터 탐색, 전처리가 재미가 있었고, 베이스라인 코드를 이해하는데 어려움이 있어서 스스로 모델 코딩을 못하겠다라고 생각 각을 해서 데이터쪽을 계속했던 것 같다.

시도 및 결과

데이터 전처리에서 3가지(개행, 외래어, 마크다운)를 전처리했다. 어떤 것을 전처리 해야 성능 개선이 이루어지는지 여러 조합을 통해 실험을 진행했다. 결과적으로는 개행만 전처리와 외래어만 전처리 한 것이 성능 개선이 이루어졌다. 처음엔 개행과 마크다운 전처리가 들어간 조합의 데이터셋이 성능이 제일 좋을 줄 알았다. 하지만 오히려 그냥 개행만 전처리 한 것이 더 성능이 좋았다. 나는 마크다운 전처리가 꽤 효과적일 줄 알았지만 반대로 외래어 전처리가 더 효과적이었다. 처음 외래어 전처리를 이야기한 나도 전처리를 하면서도 성능 개선이 될지 의문이 들었다. 하지만 내 생각과는 반대로 마크다운보다도 성능이 좋아졌다.

좋았던 점과 배운 점

사람이 봤을 때 필요없는, 해당 task에 필요없어 보이는 문자들을 없애면 당연히 성능이 올라갈 줄 알았지만 올라가지 않았다. 외래어 전처리는 의문이 있긴 했지만 마크다운 문법 전처리는 성능이 올라갈 줄 알았다. 하지만 오히려 애매했던 외래어 전처리에 비해 성능이 올라가지 않았다. 이런 부분에서 사람이 판단하는 것과 모델이 판단하는 것을 다르다는 것을 느끼게 된 것 같다.

아쉬운 점

깃협 협업을 나름 열심히 했지만, 조금 아쉬운 것 같다. 깃 사용법에 대해 정말 잘 정리를 해주셨는데, 그것을 100% 활용을 못해 아쉽다.

데이터 전처리 작업을 할 때, 중간 중간 놓친 부분들이 있어 여러번 다시 하는 일이 있었다. 꼼꼼히 확인을 안하고 작업을 해서 여러번 하게 된 점이 아쉬웠다.

제일 중요한 데이터 분석, 탐색을 너무 대충한 느낌이 든다. 꼼꼼하게 분석을 하고, 탐색을 하고, 여러 경우를 다 꼼꼼히 봤으면 전처리나 증강에서 조금 더 좋은 결과를 얻을 수 있었을 것 같다.

앞으로의 목표

이번 프로젝트에서는 그래도 지난 프로젝트에서의 아쉬움을 조금이나마 덜 수 있었다. 조금 더 적극적으로 했고, 지난 번 보다는 git을 더 활용을 했다는 점. 하지만 지난 프로젝트에서의 아쉬움인 모델쪽 코딩을 못해본 게 여전히 아쉽기 때문에 모델쪽 코딩을 해보고 싶고, 하기 위해 프로젝트 이해나 베이스라인 코드 이해를 최대한 빠르고 정확하게 해야 할 것 같다.

개인 회고 - 오수현

프로젝트 개인 목표

본 프로젝트에서는 이전 프로젝트에서의 아쉬움을 보완하고자 했다. Confluence와 블로그를 보다 활발히 활용하여 수행한 작업의 배경, 과정, 결과를 철저히 분석하여 기록하고자 했으며, 데이터와 관련된 분석을 보다 심도 있게 진행하고자 했다. 모델에 대한 작업을 수행하기에 앞서 제공받은 데이터의 형태를 분석하고, 일차적으로 모델의 결과를 확인하여 어떤 유형의 데이터에서 오답이 발생하는지 알아보고자 했다.

시도 및 결과

모델, 코드 작업에 앞서 데이터의 개요를 파악했다. 개수, 길이, 특수문자 등의 형태를 파악했으며, 지문을 보고 만든 질문이라는 특성을 알 수 있었다. 다음으로 Retrieval 기법 중 Dense Passage Retrieval을 구현하기 위해 pytorch의 train 코드와 SBERT를 사용한 코드를 모두 시도했다. 하지만 데이터 분석에서 얻은 결과와 같이 질문의 키워드를 통해 지문을 탐색해야 하기 때문에 의미적 유사성을 파악하는 것은 큰 역할을 하지 못했다. 마지막으로 Reader에서 Generation 기반의 방식을 사용해보고자 GPT-4o-mini로 프롬프트 튜닝을 수행했다. validation dataset을 기준으로 GPT-4o는 63.75%, GPT-4o-mini는 60.4167%의 EM값을 얻을 수 있었다. 하지만 test dataset을 사용하여 탐색한 문서 열 개를 한 번에 입력했을 때는 출력이 문장 형태이거나, 적절한 응답을 출력하지 못하는 경우가 많았기 때문에 수행을 중단했다.

좋았던 점과 배운 점

미약했지만 데이터 분석을 시도한 것이 좋았다. 질문과 지문의 관계를 파악한 덕분에 Dense Passage Retrieval의 성과가 좋지 않았을 때 당황하는 시간을 줄일 수 있었다. 또한 모든 내용을 Confluence에 기록하였기 때문에 프로젝트의 전체적인 흐름을 확인하기 편리했다. 마지막으로, 평소 경험해보지 못했던 GPT 프롬프팅을 통해 보다 나은 프롬프트란 무엇인지, GPT-4o와 GPT-4o-mini에 각각 어떻게 다른 형태의 프롬프트를 제공해야 하는지 알 수 있었다. 보다 나은 프롬프트 작성을 위해 모델이 출력한 결과를 확인하며 모델이 잘 못 맞히는 데이터의 유형은 어떤 것인지 알 수 있었다.

아쉬운 점

데이터의 개요는 파악했으나, 이를 통해 어떤 모델을 사용할지, 어떤 방식으로 접근할지 등에 대한 힌트는 얻기 어려웠다. 데이터 분석을 제대로 수행해본 적이 없었기 때문에 나의 인사이트 부족이라고 생각된다.

일부 코드 작업에 시간을 많이 소요하여 결과 분석에 크게 신경쓰지 못했다. GPT 프롬프팅 외에 SBERT를 사용한 Retrieval이 어떤 형태의 출력을 내는지 대략적으로 확인한 바 있지만, 이를 통해 결과적으로 어떻게 개선해야 할지를 파악하지 못한 점이 아쉽다.

전체적인 코드를 작성해보지 못한 것이 아쉽다. 특히 Reader를 작성해보지 못해 어떤 흐름인지만 대략적으로 파악했고, 튜닝을 위한 옵션 추가, 양상불을 위한 결과 출력 등 커스텀 할 수 있는 부분이 한정적이었다.

RAG로써 elasticsearch 등의 현업 트렌드를 반영하지 못한 것이 아쉽다. GPT 프롬프팅 외에도 현업에서 사용하는 RAG 기법들을 적용해보면 보다 넓은 견해를 가질 수 있었을 것 같다.

앞으로의 목표

데이터를 살펴보고 그 구조와 특징을 파악하는 것에서 그치지 않고, 어떤 부분에 집중하여 작업할지를 고민해봐야겠다. 즉, 데이터 분석과 모델링을 유기적으로 연결할 수 있는 프로젝트를 경험해보고 싶다. 뒤늦게 데이터의 특징을 깨닫는 것이 아닌, 모델의 결과를 직접적으로 비교해가며 작업을 수행하고자 한다.

시간과 자원에 매몰되지 않고 적용할 수 있는 방법론을 보다 다양하게 적용해볼 수 있으면 좋겠다. 순위도 좋지만, 현업에서 본 프로젝트의 작업을 수행할 때는 어떤 방식으로 진행되는지, 편향된 데이터셋이 아니라 일반화된 상황에서는 어떻게 데이터를 가공하고 작업을 수행해야 하는지를 보다 적극적으로 알아보고 적용해보고 싶다.

개인 회고 - 이상의

프로젝트 개인 목표

Hybrid retrieval을 구현하는 것에 중점을 두었다. 이를 달성하기 위해 sparse retrieval과 dense retrieval을 이해하고 그 둘을 결합하는 hybrid retrieval 방식에 대해 탐구하였다. hybrid retrieval 방식에는 two-stage와 hybrid embedding, ensemble 등이 있었다. 이 중 two-stage와 hybrid embedding을 수행하고자 하였다. two-stage에서는 sparse retrieval, dense retrieval을 구현 및 결합하고 hybrid embedding은 embedding 수준에서 sparse embedding과 dense embedding 각각 가중치를 두고 결과를 얻었다.

시도 및 결과

Two-stage retrieval 방법과 hybrid embedding 방법을 시도하였다. 이를 위해 sparse retrieval 방법에선 tf-idf, bm25를 dense retrieval 방법에선 bert-base, sbert 등 사용해 결합하고자 하였다. 그 결과, bm25는 구현을 못 했으나 two-stage retrieval에선 tf-idf에 대해 bert-base, sbert를 결합했을 때 각각 개별적으로 사용한 것보다 더 높은 성능을 얻을 수 있었다. 반면, hybrid embedding 방법에서는 embedding 값끼리 weighted sum을 하거나 concat을 해봤으나 각각 개별적으로 사용한 것과 성능에서 큰 차이가 없었다.

좋았던 점과 배운 점

프로젝트를 진행하면서 hybrid retrieval을 구현해볼 수 있었고 sparse retrieval과 dense retrieval의 장점을 결합한 결과를 볼 수 있었다. two-stage retrieval을 구현하면서 dense reranker의 역할을 이해할 수 있었다. hybrid embedding의 경우 서로 학습한 방식이 다른 sparse embedding과 dense embedding을 결합했으나 효과는 없었다. 시도해보진 못 했지만 각 retrieval을 수행한 결과에 대해 가중치를 주는 ensemble이 더 직관적이고 효과적이라고 느꼈다.

아쉬운 점

BM25를 직접 구현하려 했으나 결국 기한 내에 구현하지 못 하였다. 실행 속도가 느려 개선하기 위해 matrix 표현, multi processing 등 여러 방법을 썼지만 효과가 없었다. BM25도 sparse retrieval을 수행하는 방법인데 만약 sparse matrix에 최적화된 data structure를 사용했다면 더 나은 결과를 얻을 수 있었을 것으로 생각한다.

ODQA 프로젝트를 하면서 더 개선된 결과를 보여주지 못한 것이 아쉽게 느껴진다. hybrid retrieval 구현이 자체되면서 실험을 제때에 진행할 수 없었고 이는 더 다양한 방법을 시도하는데 시간적 제약을 받게 되었다. 그에 따라 팀에 결과를 공유하는 것 또한 지연되어 더 나은 retrieval 모델을 보일 수 없었다.

앞으로의 목표

- 모듈을 기능별로 더 구분하여 각 부분에서 개선 가능한 부분을 확인해볼 것이다.
- 전반적인 코드 구조와 작업에 대해 면밀히 검토해볼 것이다.
- 전체적인 파이프라인에 대해 고려해볼 것이다.
- 추가적인 레이어 구성을 꾀해볼 것이다.

개인 회고 - 이정인

프로젝트 개인 목표

목표를 잡고 특정 부분에 집중하기!

멘토님께서 말씀하신 특정 부분에 집중하라는 조언에 따라 데이터 관련 지식을 넓히고 이를 포트폴리오에 남기는 것을 목표로 정하고 프로젝트2 진행하였습니다

시도 및 결과

- 데이터 분석 및 내용 통합 : git의 새branch 생성 feature/EDA 개설 후 팀원들의 다양한 데이터 분석 과정을 종합하여 Confluence에 올림 이를 통해 git을 통한 협업에 대한 이해도 대폭 상승
- 데이터 전처리 : 개행, 마크다운, 외래어등 다양한 방법으로 전처리 파일 생성 후 테스트 이를 통해 좋은 성능을 나타내는 다양한 전처리 데이터 셋을 찾을 수 있었습니다.
개행, 외래어, 개행+외래어, 개행+마크다운+외래어를 활용
- 데이터 증강 AEDA 시도 이를 통해 MRC train에 대한 성능을 확인 할 수 있었으나 평가 데이터 까지 확인을 하지 못함

좋았던 점과 배운 점

- git에 장시간 오류를 해결하지 못한 부분이 있어 git이 많이 어려웠는데 팀원들의 도움으로 다시 시도해 볼 수 있었고 이번에는 오류 없이 무사히 과정을 수행할 수 있었습니다.
- 데이터 분석에 대한 내용을 통합 하는 과정에서 전처리를 어떤 식으로 진행하면 좋은가에 대한 지식도 대폭 상승하여 앞으로 어떤 부분을 고려해서 데이터를 전처리해야 할지에 대한 이해도가 많이 올랐던 거 같습니다(개행, 마크다운, 외래어, 특수문자)

아쉬운 점

- 데이터 시각화의 다양한 툴들을 활용 하지 못함
다른 팀들은 Streamlit 및 다양한 방식으로 데이터를 시각화 하여 원활한 데이터 분석을 진행하였지만 자신은 그런 정도를 제공해 주지 못한 아쉬움이 너무 크게 남는다.
이는 추후 프로젝트에서 한번 다른 툴들을 활용해 데이터를 시각화 해보겠습니다.
- 데이터 증강 AEDA에 대해 너무 매몰되어 있다 보니 다른 시도를 해보지 못한 것에 너무 아쉽습니다.. 프롬프트 도 진행해 보고 싶었는데... 다음 기회에 프롬프트도 한번 진행해 보면 좋을 거 같다.

앞으로의 목표

- 다양한 분야에 균형 있게 접근하기
- 최신 데이터 분석 및 시각화 툴 학습 (예: Streamlit)
- 프롬프팅 등 다양한 데이터 증강 방법 시도
- 멘탈 관리의 중요성 인식 및 실천

개인 회고 - 이정휘

프로젝트 개인 목표

본 프로젝트에서는 전체적으로 모든 업무에 참여하기도 하지만, 특히 모델관리나 모듈화하는데에 목표로 하였다. 최대한 모든 팀원들이 공통적인 코드, 환경으로 테스트할 수 있도록 하는 것이 중요하다 생각하였고 각자 만든 모듈을 inference로 통합시키는 일을 하였다.

시도 및 결과

기존 베이스라인 코드를 분석하고 모듈들을 분리하는 과정을 거쳤다. 하지만 Nested Function으로 이루어진 구조가 많아 이를 해결하는데에 오랜 시간을 들였다. 그 후 inference 코드를 작성할 때에는 최대한 모듈화에 중점을 두었다. 팀원들이 구현한 Reader, Retrieval의 Class를 보고 input과 output을 일관되게 맞추는 시도를 하였고, Arguments의 옵션을 변경함에 따라 자유자재로 테스트할 수 있도록 구현하려고 시도했다. 그 결과 설정값과 옵션만 바꿔서 바로 결과를 볼 수 있도록 하였다.

추가로 Multilingual Model에 KorQuad 데이터셋을 Finetuning 하고 해당 모델을 HuggingFace에 올려 Trina dataset를 다시 Finetuning할 수 있도록 하였다.

좋았던 점과 배운 점

inference 모듈화를 시키는 것에 초점을 두면서 앞으로 어떤 코드를 짜야할지 생각할 수 있게 되었다. 그리고 전혀 이 코드를 모르는 사람도 실행할 수 있도록 코드를 짜야하는 것에 방향성을 볼 수 있게 된 점이 크다. 그리고 Multilingual 모델에 다른 데이터셋을 finetune 시키는 작업을 하면서 좀 더 HuggingFace 모델을 활용하는 방법을 배울 수 있어서 좋았다.

아쉬운 점

아무래도 모듈화, 베이스라인 코드 재구성을 하면서 너무 많은 시간을 소요한 것이 아쉽다. 빠르게 베이스라인을 짜고 실험을 하며 Task를 어떻게 해결해나가야 할지에 대해 연구를 했어야 했는데, 코드에만 매몰되어 있어서 아쉬웠다. 그리고 코드면에서도 베이스라인 코드에 너무 매몰되어 있던 점도 아쉬웠다. 나중에 Inference코드를 작성하면서 너무 복잡하게 짜여진 기존 베이스라인을 그대로 옮기기보다는 아예 새로 구성하는게 더 나았을 것 같고 더 공부가 될거라고 생각한다. 그리고 저번 프로젝트부터 “기록”에 중점을 두었어야 했는데, 여전히 기록에 익숙하지가 않다. 모델을 변경하거나 기능을 추가하거나 등등 어떤 issue가 발생했을 때 모든 것들을 기록해야 추후에 결과를 비교할 때 더 좋은 인사이트를 얻을 수 있는데, 이것을 아직 습관화시키지 못했다는 것이 아쉬웠다.
마지막으로, 프로젝트 진행 방향이다. 여태까지 “기능”과 “방법”에서 최고 성능을 보이자는 것이 목표였다는 것이 문제였다. 모든 진행은 밑단부터 차근차근 쌓아야 제대로된 비교가 가능한데, 이 점이 너무 아쉬웠다.

앞으로의 목표

우선은 “기록”이다. 일에 치여 기록을 멀리하지 말자. 가장 중요하다. 이번 결과보고서를 쓰면서도 분명 실험했던 내용인데, 기록이 없어 결국 작성하기 어려운 경우도 발생하게 되었다. 그리고 똑같은 결과를 여러번 내게되는 문제가 발생하게 된다. 그래서 특히나 기록, 또 기록이다.

그리고 프로젝트 진행 방향이다. 다른 팀의 발표를 보고 기초 모델 선정에서부터 필요한 방법론 적용까지 모두 하나하나씩 쌓아올려야 한다는 것을 깨닫게 되었다. 다음 프로젝트부터는 이를 중점적으로 보아야겠다.

마지막으로, 베이스라인 코드에 매몰되지 말자.

개인 회고 - 정민지

프로젝트 개인 목표

이전 프로젝트에서는 AI 프로젝트의 전체 흐름은 이해했지만, Huggingface 라이브러리나 PyTorch로 모델을 불러와 학습하는 과정은 미숙했다. 그래서 이번 프로젝트에서는 데이터 분석이나 이론보다는 직접 학습과 평가 흐름을 코드로 구현하는 것을 목표로 삼았다. 더불어, GitHub의 branch와 commit 기능을 넘어 Issue와 Pull Request를 적극적으로 활용해 협업을 경험하고자 했다.

시도 및 결과

원활한 협업을 위해 "개발자처럼 Github 사용하기" 강의를 참고해 우리 팀의 Github 사용 규칙을 정리해 공유하고, Issue 작성법과 PR 템플릿 등을 도입했다. 지나치게 세세한 규칙을 좋아하지는 않지만, 최소한의 규칙이 있으면 협업에 통일성이 생겨 효율적이라는 것을 깨달았다. 덕분에 이번 프로젝트에서 Github 활용 목표를 이루었다.

코딩 측면에서는, ODQA 코드 흐름을 이해하기 위해 AI Stage의 코드를 분석하면서 retriever와 reader 모듈을 분리하려 시도했으나, 전체 흐름을 이해하기보단 분리에만 집중해 실패했다. 이를 통해 전체 흐름을 이해하고 코드를 새로 작성하는 편이 효율적이라는 교훈을 얻었다. 이후 sparse retriever와 generative (Llama) reader 모듈 개발을 맡아 input과 output 형상, config 파일 구성을 고민하며 작성했다. learned sparse retriever를 구현하며 임베딩 벡터의 효율적인 저장과 연산 방법을 고민하다가 Milvus 벡터DB를 빠르게 습득해 적용했던 점이 기억에 남고, 한편으로는 BM25로 쿼리 별 유사한 문서를 검색하는 속도가 느려, 이 속도를 개선하기 위해 소스코드를 새로 작성한 경험도 유익했다. 프로젝트 마지막 즈음에는 약 하루 안에 Llama fine-tuning 코드를 작성하고 이후 약 이틀 간 여러 프롬프트 및 데이터에 대한 파인튜닝 실험을 수행했다. 제한된 시간 내에 목표를 구현해낸 점에서 의미있는 시도였고, 코딩 실력과 자신감이 많이 늘었다. 클래스에서 `__init__`만 봐도 겁먹던 나였는데, 이번 프로젝트에서만 직접 3개 이상의 클래스를 작성하고 이 모듈들을 전체 코드에 병합해 넣을 수 있게 된 내 모습을 보며 많이 성장했구나 느꼈다.

좋았던 점과 배운 점

ODQA 문제를 다루고 retrieval과 reader 모듈을 모두 작성해볼 수 있었던 점이 좋았다. ODQA 문제를 이해하며 결국은 이 문제가 최근 뜨거운 감자인 RAG와 일맥상통함을 알게 되었고, 다양한 방법론을 실험했다. 특히 BM25의 안정적인 우수한 검색 성능을 확인했고, Learned Sparse Retrieval 방법을 실험하며 벡터DB의 사용법도 익힐 수 있었다. 마지막에 시도한 Llama 모델의 Q-LoRA 파인튜닝 작업도 흥미로웠다. 이름만 익숙했던, RAG의 다양한 세부 방법론들을 직접 코드로 구현하고 실험해볼 수 있어서 즐겁게 프로젝트를 진행할 수 있었다.

아쉬운 점

학습 결과를 체계적으로 기록하지 못한 점이 아쉬웠다. sh 파일로 실험을 편리하게 수행했으나 성능 개선 결과가 체계적으로 정리되지 않아 성과 비교가 어려웠다. 한편 Llama 실험도 조금 더 일찍 시도했다면 프롬프트나 데이터 후처리 등 다양한 가설을 더 깊이 검증할 수 있었을 텐데, 시간 제약으로 충분히 시도하지 못한 점이 아쉽다.

앞으로의 목표

RAG 시스템을 더욱 공부하고, Llama를 활용해 챗봇 파이프라인을 구성해보고 싶다. 리더보드용 프로젝트는 마무리되었으나, 생성형 접근을 통해 위키피디아 데이터를 활용한 QA 챗봇을 만들어보고자 한다. 이번에 가볍게 시도했던 벡터 DB를 더 본격적으로 사용할 수 있기를 기대하며, 이 과정에서 RAG와 Llama 모델을 더 깊이 이해해보고자 한다.