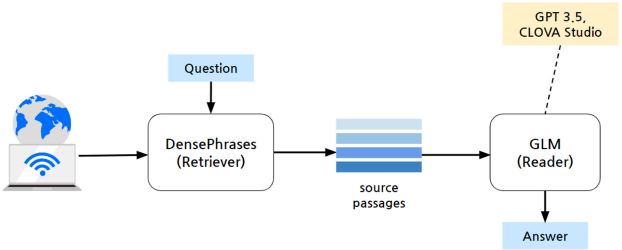
생성형 검색을 위한 **prompt** 경량화

NLP 06조

1. 프로젝트 개요

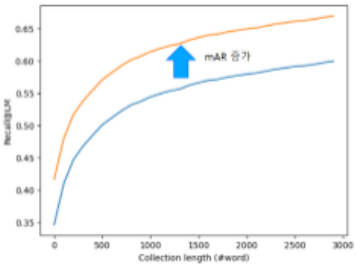
1.1. 개요



- ODQA란 질문이 주어졌을때, retriever가 해당 질문에 답할 수 있는 passage들을 지식 베이스로부터 찾아 근거 문서를 구성하고, 그렇게 구성된 근거 문서를 기반으로 reader가 답변하는 시스템이다.
- 본 프로젝트는 densephrase 모델을 retriever로, GPT와 같은 생성형 언어 모델을 Reader로 사용한다. 따라서 프로젝트의 최종 목표는 densephrase 모델의 성능을 향상시켜서 reader model에 prompt 형태로 들어가는 source document를 경량화 하는 것이다.

1.2. 평가 지표

- mAR (mean Average Recall)
  - 근거 문서의 길이에 따른 평균 정답 포함율
  - 정답 생성에 필요한 비용과 정답 포함율을 모두 고려한 지표



$$\mathrm{mAR} = \frac{\sum_{i=1}^{L} \mathrm{Recall@LM}(i)}{L}$$

$$\mathrm{Recall@LM}(i) = \frac{\sum_{q \in Q} \mathrm{recall}(q, i)}{n(Q)}$$

$$L = \text{max collection length}$$

1.3. 프로젝트 환경

- 컴퓨팅 환경 : 인당 V100 서버 할당
- 모듈 : Huggingface, PyTorch
- 협업 환경 : Github, Notion
- 의사소통 : Zoom, 카카오톡, 대면 회의

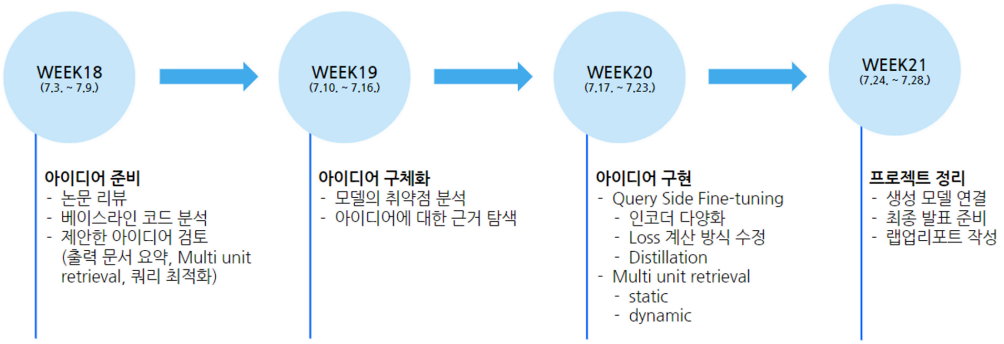
2. 프로젝트 팀 구성 및 역할

- 김민호 : Query loss의 단위 변경, Loss의 구성 요소 추가
- 김성은 : Query loss의 단위 변경, Dynamic retrieval
- 김지현 : Dataset 전처리, Query loss의 단위 변경, Knowledge Distillation

- 서가은 : Query loss의 단위 변경, Dynamic retrieval
- 홍영훈 : Query loss의 단위 변경, Static retrieval, Optimization

3. 프로젝트 수행 절차 및 방법

3.1. Timeline



3.2. 협업 문화

- Git을 활용하여 코드 공유
- Notion을 활용하여 아이디어 및 결과 공유

4. Data 소개

- **Natural Questions Dataset** : Question Answering Task 의 Benchmark Dataset 중 하나이다.
- id : 질문 id
- question : 질문 텍스트
- titles : 질문의 답이 속한 문서 제목
- sentence : 정답들이 속한 문장 리스트
- context : 정답들이 속한 문단
- answers : 정답들로 구성된 리스트

id	2126085010748850659
question	when does season 5 of bates motel come out
titles	Bates Motel (season 5)
sentence	[' The fifth and final season of "Bates Motel" premiered on **February 20, 2017**, and concluded on April 24, 2017.']
context	' The fifth and final season of "Bates Motel" premiered on **February 20, 2017**, and concluded on April 24, 2017. The season consisted of 10 episodes and aired on Mondays at 10 p.m. ET/PT on A&E. The series itself is described as a "contemporary prequel" to the 1960 film "Psycho", following the life of Norman Bates and his mother Norma prior to the events portrayed in the Hitchcock film. However, the final season of the series loosely adapts the plot of "Psycho". The series takes place in the fictional town of White Pine Bay, Oregon.'
answers	['February 20, 2017']

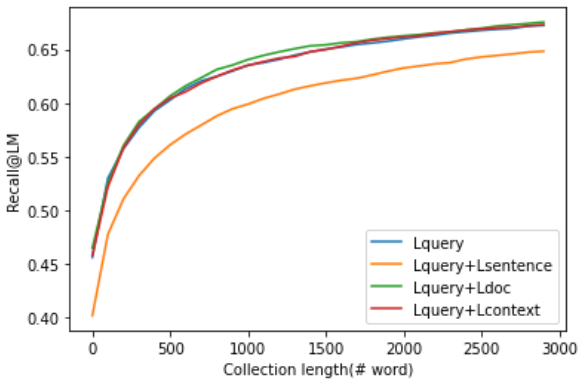
5. 프로젝트 수행 결과

5.1. Query-side fine-tuning(QSFT)

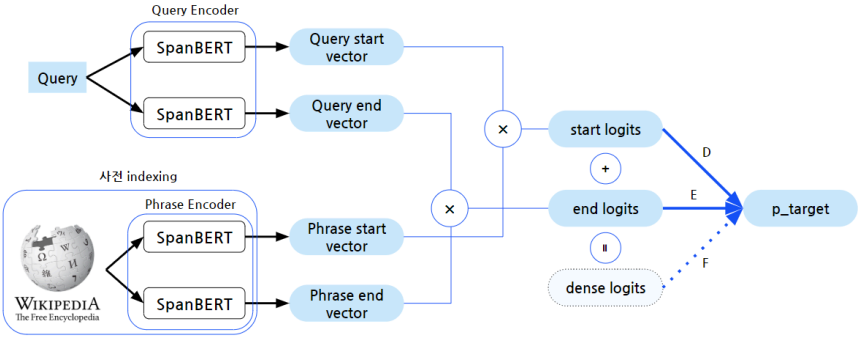
- **Query loss**의 단위 변경
  - Loss를 다양하게 사용하여 query encoder를 학습한다.
  - 기존의 DensePhrases에서 사용하는 L_query와 L_doc은 retrieval된 phrase가 gold phrase와 일치하도록 하거나 gold phrase의 document와 일치하도록 하는 목적함수이다.
  - 프로젝트의 base retrieval unit이 sentence이므로 gold sentence를 찾는 것이 중요하다고 생각하여 L_sentence와 L_context를 추가로 도입한다.

	설명	수식	**mAR**
L_query	추출된 phrase가 gold phrase와 일치하는지 여부를 계산	$\mathcal{L}_{\text{query}} = -\log \frac{\sum_{s \in \widetilde{S}(q), \text{TEXT}(s)=a^*} \exp\left(f(s\|\mathcal{D}, q)\right)}{\sum_{s \in \widetilde{S}(q)} \exp\left(f(s\|\mathcal{D}, q)\right)}$	0.6316
L_sentence	추출된 phrase가 gold phrase가 위치하는 gold sentence에 포함되는지 여부를 계산	$\mathcal{L}_{\text{doc}} = -\log \frac{\sum_{s \in \widetilde{S}(q), d(s) \in \mathcal{D}^*} e^{f(s,q)}}{\sum_{s \in \widetilde{S}(q)} e^{f(s,q)}}$	0.6251
L_context	추출된 phrase가 gold phrase가 위치하는 gold context에 포함되는지 여부를 계산	$\mathcal{L}_{\text{context}} = -\log \frac{\sum_{s \in \widetilde{S}(q), c(s) \in \mathcal{C}^*} e^{f(s,q)}}{\sum_{s \in \widetilde{S}(q)} e^{f(s,q)}}$	0.6322
L_doc	추출 된 Phrase 가 gold phrase 가 위치하는 gold document 에 포함되는지의 여부를 계산	$\mathcal{L}_{\text{sentence}} = -\log \frac{\sum_{s \in \widetilde{S}(q), sent(s) \in \mathcal{S}^*} e^{f(s,q)}}{\sum_{s \in \widetilde{S}(q)} e^{f(s,q)}}$	0.6351

  - 실험 결과



- **Loss**의 구성요소 추가
  - L_query + L_doc 인코더에서 Loss의 구성 요소를 추가한다.
  - DensePhrases 모델의 Query Encoder와 Phrase Encoder는 각각 start와 end의 2가지 벡터를 출력한다. 그리고 start 벡터와 end 벡터를 각각 곱하여 start logits와 end logits를 계산한다. 이 2가지 logits를 더해 dense logits를 생성하여 총 3가지 logits를 사용한다.
  - 한편, L_doc의 연산과정은 기본적으로 L_query와 동일하지만, dense logits는 생성하지 않는다. 따라서 dense logits를 사용하여 L_doc을 구현하고, L_query + L_doc 인코더를 실험한다.



○ 결과: gold phrase를 맞추지 못하였음에도 gold sentence를 retrieve하도록 하는 목적을 달성했다.

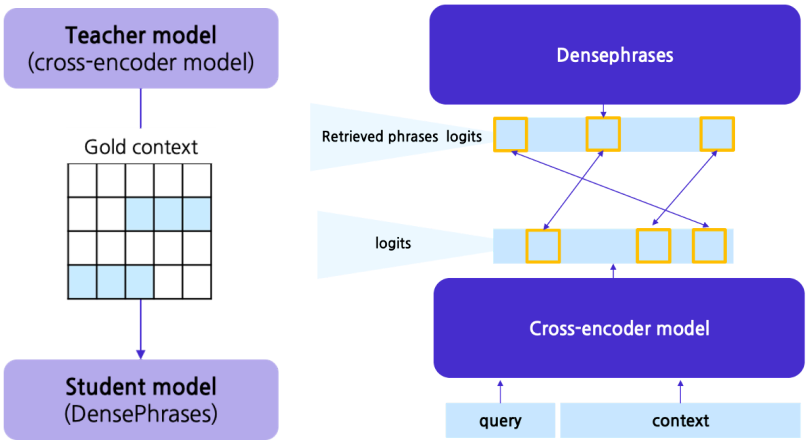
- phrase_correct : 출력된 sentence에 정답이 포함되어 있고, 그 정답이 해당 sentence를 출력하게 한 phrase일 경우

- sent_correct: 출력된 sentence에 정답이 포함되어 있지만 그 정답이 해당 sentence를 출력하게 한 phrase는 아닐 경우

	phrase_correct	sent_correct	sum	mAR	Loss의 구성 요소
L_query	2,505	1,773	4,278	0.6316	A + B + C
L_query+L_doc	2,385	1,925	4,310	0.6353	A + B + C + D + E
L_query+L_doc custom	2,302	1,993	4,295	0.6332	A + B + C + D + E + F

	phrase_correct	sent_correct	sum	mAR	Loss의 구성 요소
L_query	2,505	1,773	4,278	0.6316	A + B + C
L_query+L_context	2,105	2,187	4,292	0.6322	A + B + C + D + E
L_query+L_context custom	1,941	2,323	4,264	0.6289	A + B + C + D + E + F

- **Knowledge Distillation**
  - ○ Query side fine tuning 환경에서 pre-train 된 cross-encoder model 을 teacher model 로 , densephrases 모델을 student model로 활용하여 knowledge distillation 을 수행한다.
  - ○ Query encoder 가 context 내에서 gold phrase 를 보다 높은 확률로 예측할 수 있도록 학습하며 이로 인한 mAR 향상을 목표로 한다.



○ cross-encoder 에서 생성한 logit 과 densephrases 에서 반환 된 top-k 개의 logit 들 중 일부를 추출하여 비교하고, 그 차이를 distillation loss 로 사용한다.

○ 결과

- phrase_correct : 출력된 sentence에 정답이 포함되어 있고, 그 정답이 해당 sentence를 출력하게 한 phrase일 경우

- sent_correct: 출력된 sentence에 정답이 포함되어 있지만 그 정답이 해당 sentence를 출력하게 한 phrase는 아닐 경우

	phrase_correct	sent_correct	sum	mAR
L_query	2,505	**1,773**	4,278	**0.6316**
L_query & Distillation	**2,532**	1,758	**4,290**	0.6311

	phrase_correct	sent_correct	sum	mAR
L_query+L_doc	2,385	**1,925**	4,310	0.6353
L_query+L_doc & Distillation	**2,504**	1,821	**4,425**	**0.6362**

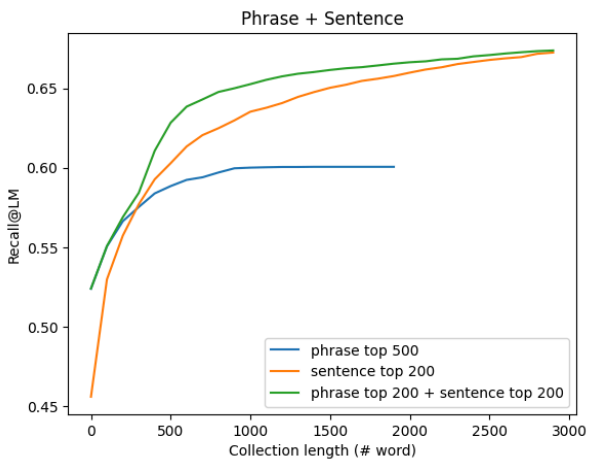
Loss	mAR
L_query	0.6316
L_query+L_sentence	0.6251
L_query+L_context	0.6322
L_query+L_doc	0.6351
L_query+L_doc & Distillation	**0.6362**

실험 모두에서 phrase_correct 향상을 관측하였다.

mAR 0.6316 -> 0.6362

5.2. Multi Unit Retrieval

- **Multi Unit Retrieval**
  - DensePhrases의 inference 단계에서는 retrieval unit(phrase, sentence, paragraph, document)과 top K를 입력하여 retrieval unit에 해당하는 K개의 text를 retrieve한다. Baseline의 retrieval unit이 sentence였기 때문에, sentence와 함께 길이가 보다 짧은 phrase를 같이 retrieve하여 prompt을 경량화하는 방법을 고려하게 되었다.
  - phrase와 sentence를 함께 retrieve하는 방식을 multi unit retrieval이라고 이름을 지었으며, 세부적인 방식에 따라 static과 dynamic 방법으로 나뉜다.
- **Static**
  - phrase만 반환하는 경우 초반에 높은 Recall을 가지지만, 낮은 recall 상승률을 가진다. 반면 sentence의 경우 recall 상승률이 높다. phrase와 sentence 모두의 장점을 이용하기 위해 phrase를 먼저 주고 sentence를 주는 방식으로 retrieve 결과를 재정렬한다.
  - 실험 결과 :



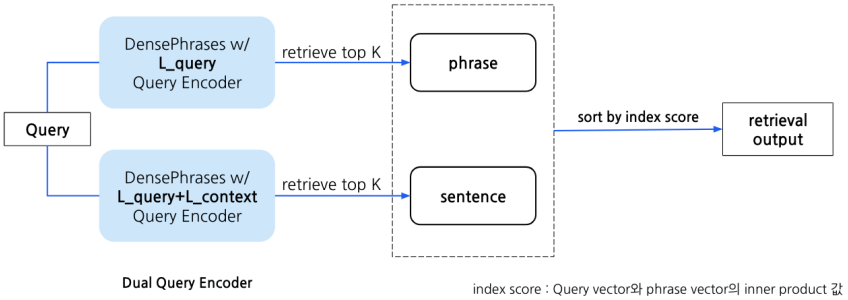
mAR 0.6316 -> 0.6450

- **Dynamic**
  - Dual query encoder : query와 phrase의 관계에 따라 유동적으로 retrieval unit을 결정하기 위해 query encoder를 unit에 따라 다르게 사용하는 방법을 고려하게 되었다. phrase와 sentence에 대하여 여러 encoder의 성능을 실험해본 결과, phrase에서는 L_query, sentence에서는 L_context와 L_doc query encoder가 높은 mAR을 보였다. 따라서 query를

각 unit에 따른 두 가지의 인코더로 각각 인코딩하여 index search를 진행하였다. unit과 query encoder에 따른 실험 결과는 다음과 같다.

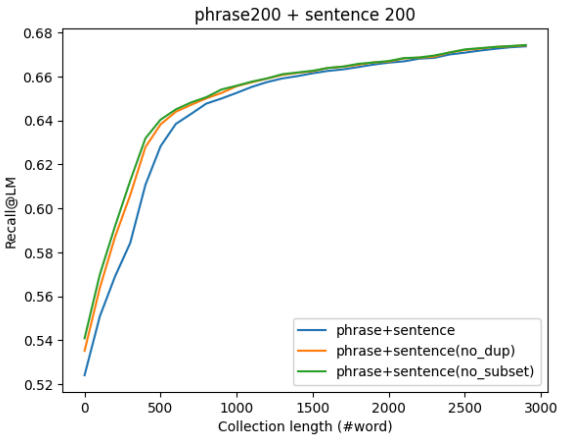
	phrase	sentence
L_query	**0.5629**	0.6316
L_query + L_context	0.5407	0.6322
L_query + L_doc	0.5509	**0.6351**

- Dynamic retrieval : L_query로는 phrase K개를 retrieve하고, L_context로는 sentence K개를 retrieve하여 총 2K개의 text를 index score를 key로 재정렬하여 retrieval text를 구성하였다.



- L_query encoder를 사용하여 sentence top 200개를 retrieve한 mAR 0.6316에서 phrase는 L_query, sentence는 L_context를 활용하여 각각 200개를 retrieve한 후 재정렬한 mAR 결과인 0.6564로 약 0.02의 mAR 상승이 있었다. 두 결과 모두 최대 3000 단어 기준으로 mAR을 계산하였다.

- Optimization
  - phrase를 반환하는 경우 이전에 나온 phrase에 새로 나오는 phrase가 포함되는 경우가 있다. 이런 경우에는 다시 반환할 필요가 없으므로 이를 제거하여 retireve 결과를 재정렬한다.
  - 실험 결과 :



mAR 0.6450 -> 0.6505

- Result

	mAR
static	0.6450

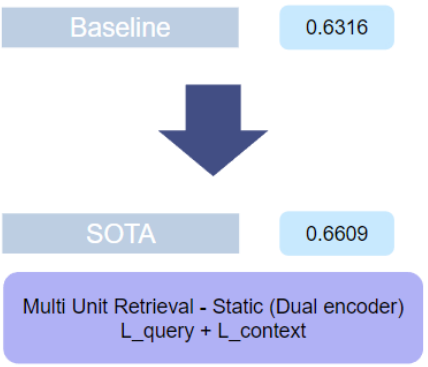
static + Opt	0.6505
dynamic	0.6564
static + Opt + Dual	0.6583
dynamic + Opt	0.6609

- Opt : 최적화(중복제거, subset 제거) 적용
- Dual : Dual query encoder 적용

6. Conclusion

6.1. Result

- 7가지 가설에서 성능 향상의 결과를 얻었다.
- QSFT
  - Query loss의 단위 변경
    - **L_query + L_doc**
    - L_query + L_sentence
    - **L_query + L_context**
  - Loss 구성 요소 추가
  - **Knowledge Distillation**
- Multi Unit Retrieval
  - Static
    - **Single Encoder**
    - **Dual Encoder**
  - Dynamic
    - **Dual Encoder**
  - 최적화



7. 자체 평가 의견

- 의의
  - Retriever에서 recall과 length의 trade-off를 고려하며 retrieval model의 성능을 향상시키기 위해 다양한 시도들을 적용
  - 프롬프트의 경량화를 통해 reader model의 계산 비용을 최소화하며 성능은 유지할 수 있음
- 한계
  - 실험에 사용된 NQ Dataset의 한계
    i. NQ Dataset은 단답형 질문으로만 구성되어 있고, 질문의 근거가 되는 source document가 하나로 정해져 있음
    ii. 그러나, 실제 상황에서는 1) 단답으로 해결 불가능한 complex query들이 포함될 가능성이 있고, 2) 질문의 근거가 되는 source document가 여러개 필요할 수도 있음

iii. 따라서 실제 상황을 반영하는 data는 아니므로, 추후에 이러한 한계점들을 보완하는 여러 data들로 실험을 진행할 필요가 있음

- Indexing 파일의 크기가 너무 큼 -> QSFT 시 소요되는 시간이 너무 큼(1epoch 당 평균 3시간)
  i. 전체 가설을 통합한 결과를 내지 못함
  ii. 각 가설들을 더 구체화하지 못함

8. 참고 문헌

- Jinhyuk Lee, Mujeen Sung, Jaewoo Kang, and Danqi Chen. 2021. Learning Dense Representations of Phrases at Scale. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6634–6647, Online. Association for Computational Linguistics.
- Jinhyuk Lee, Alexander Wettig, and Danqi Chen. 2021. Phrase Retrieval Learns Passage Retrieval, Too. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3661–3672, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

개인 회고

김민호

- 나의 학습 목표
  - 단순 성능 향상 뿐 아니라 명확한 문제상황 정의와 원인 분석을 통해 목표를 달성한다.
  - 최종적으로 생성모델과 연결하여 mAR 뿐 아니라 목적을 달성했는지 확인한다.
- 모델 개선을 위해 시도한 것
  - Loss의 구성요소를 추가하여 모델의 성능 개선을 도모하였다. DensePhrases 모델이 기본적으로 사용하는 Loss 함수인 L_query에서 사용하는 요소는 3가지를 사용하는 반면, L_doc은 2가지만을 사용한다. 따라서 L_doc에도 나머지 하나를 추가하였다. 기본적인 2가지는 start logits와 end logits로 각각 쿼리의 시작과 종료 토큰이 정답과 유사하도록 한다. 나머지 하나는 둘을 더한 logits로, 시작 위치와 종료 위치를 모두 포함하여 정답과 유사하도록 전체적인 맥락을 파악하는 역할을 한다고 판단했다. 그 결과, 정확히 정답을 맞추지는 못했지만 정답 문장을 추론하는 능력은 상승한 것을 확인할 수 있었다. 하지만 전체적인 mAR은 떨어졌다.
- 새롭게 시도한 것
  - 서비스를 목적으로 하는 다른 조의 최종프로젝트와 달리, 성능 향상을 목적으로 하는 프로젝트를 진행하였다. 기존의 대회 형식의 프로젝트와 유사하게 진행되었으나 다른 팀과 경쟁하기 위한 리더보드가 없었기 때문에 온전히 팀의 가설에 집중할 수 있었다. 덕분에 리더보드와 같이 성적에 연연하지 않고 프로젝트를 진행할 수 있었다.
  - 기업연계프로젝트로, 매주 기업 멘토님께 피드백을 받는 상황이 매우 새로웠다. 또한, 팀에 배정된 멘토님 또한 대학원에서 Information Retrieval을 연구하고 계시며 DensePhrases를 사용하고 계셨기에 팀의 진행 상황을 잘 알고 계셔서 많은 도움을 받을 수 있었다.

DensePhrases가 생소한 기술이고 자료도 많이 없었지만 여러 멘토님들의 도움 덕분에 가설에 대한 구체적 방법을 고안하거나 근거를 찾을 때 이론적인 도움을 많이 받을 수 있었다.

- 아쉬웠던 점
  - 위키 데이터의 인덱스 사이즈가 매우 커서, 학습과 추론에 많은 시간이 필요했다. 기존의 DensePhrases에서 사용하는 인덱스는 74 GB이고, 서버의 저장소 용량 문제로 작은 사이즈의 21 GB를 사용하였음에도 불구하고 1에포크에 3시간이 걸리고, 추론시에도 2시간 가량 필요했다. 시간과 자원의 제약으로 인해 온전한 실험을 진행하지 못한것이 아쉬웠다. 이로 인해 다양한 특성을 가진 모델을 혼합하여 Multi Unit Retrieval에 응용하고 싶었으나, 충분한 시도를 하지 못한것이 아쉬움으로 남는다.
- 다음 프로젝트에서 시도해볼 것
  - 이번 프로젝트와 유사하게, LLM을 사용한 프로젝트를 시도 해 볼 것이다. 프롬프트를 통해 LLM의 성능을 조절할 수 있다는 것이 흥미로웠다. 프롬프트를 통해 LLM을 튜닝하거나 원하는 방향으로 유도하여 최종적으로는 실생활에 유용한 서비스를 개발하고자 한다.

김성은

- 나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

phrase와 sentence를 혼용하여 retrieve함으로써 prompt를 경량화하는 목표를 세웠다. 총 K개의 text를 retrieve할 때 question과 각 text에 따라 unit을 결정하고자 하였다. unit을 판단하기 위한 수단으로 query encoder를 unit에 따라 다르게 사용하는 것을 시도해보았다.

한 쿼리에 대해 같은 phrase를 여러 번 retrieve하게 되는 경우가 있는데, 이는 source sentence가 다른 경우이다. 즉 phrase 자체는 같지만 문맥이 다른 경우인데, L_query를 사용하게 되면 이러한 경우를 구분하지 못한다. 또한 query에서 필요한 문맥이 아닌 경우가 있기 때문에, 이 경우는 phrase 주변 문맥을 같이 retrieve하는 것보다 phrase 자체만 retrieve하는 것이 경제적이다. 두 번째로 query와 관련한 문맥이 어느 정도 필요한 경우이거나, 복수 정답이 모두 phrase의 source sentence에 있는 경우에는 phrase보다 source sentence를 retrieve하는 것이 recall 면에서 이득이다. 두 번째 경우의 phrase는 L_query보다 larger granularity에 대한 loss인 L_context query encoder가 더 잘 찾아낼 것이라고 보았다. 실제로 각 unit에 대하여 query encoder를 다르게 하여 실험한 결과, retrieval unit이 phrase일 때는 L_query encoder가 mAR 성능이 가장 좋았고, sentence일 때는 L_context, L_doc encoder를 사용한 결과가 L_query encoder를 사용한 결과보다 높았다.

따라서 phrase는 L_query로, sentence는 L_context 또는 L_doc encoder로 encoding한 query로 retrieve하는 dual query encoder를 사용하여 dynamic retrieval을 구현하였다. unit 별로 다른 encoder를 사용하여 각각 k개의 phrase와 sentence를 retrieve한 후 나온 index score를 토대로 재정렬을 진행하여 retrieval output으로 사용한다. index score는 query vector와 phrase vector의 inner product 값으로, 높을수록 쿼리와 유사한 phrase라고 볼 수 있다.

이러한 방식으로 dynamic retrieval을 구현한 결과 baseline의 결과보다 약 0.02의 mAR 향상이 있었다.

- 전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

단순히 성능을 높이기 위해 가설을 세운 것이 아니라 baseline의 한계를 파악하여 이를 해결하기 위한 방법을 생각해 보았다. 내가 파악한 baseline의 한계는 retrieval unit이 고정되어 있었다는 점이었고, 이를 해결하기 위해 dynamic retrieval을 시도해보게 되었다. 성능 지표만으로 가설을 평가하기보다 이 가설을 통하여 baseline의 한계를 어떻게 해결할 수 있을지 고민하고 시도해 본 것이 새로웠다.

- 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

처음에 구현한 dynamic retreival은 Lquery 쿼리 인코더로 phrase K개를 retrieve하고, Lcontext 쿼리 인코더로 sentence K개를 retrieve한 tsv 파일을 각각 뽑아서 이들을 score 순으로 sorting하여 새로운 tsv 파일을 만드는 것이었다. 즉 이미 만들어진 tsv 파일을 재정렬하는 후처리 방식이었다.

이와 다르게 코드 상에서 구현하는 방식을 시도하였다. 서로 다른 encoder로 인코딩된 query vector 두 개로 각각 index search를 진행하여 k개의 index를 찾고, 각 k개의 교집합에 대하여 index score을 토대로 unit을 결정하는 방식이다. 이 방식이 좀 더 일반화된 dynamic retrieval에 가까운 방식이라고 생각하였으나, 후처리 방식만큼 성능이 나오지 않았다. 추후에 이 부분을 더 개선해보고자 한다.

- 한계/교훈을 바탕으로 다음 프로젝트에서 시도해보고 싶은 점은 무엇인가?

모델링에 집중하여 모델 구조를 변경함으로써 경량화할 수 있는 방식을 시도해보고 싶다.

김지현

- 나의 학습 목표
    - densephrases 를 논문과 코드 모두에서 최대한 이해하고 활용하기
    - 소폭이라도 mAR 향상에 기여하기
    - 지금 까지 해보지 않은 역할 수행해보기
- 내 학습목표 달성을 위해 시도한 것**(**모델 개선을 위해 시도한 것**)**
    - 1.2번 목표를 위한 시도 : Knowledge distillation (mAR 0.6316 -> 0.6362)
      densephrases 는 각 모듈들이 매우 유기적으로 연결되어 있다는 점에서 어렵게 느껴지는 모델이었다. 때문에 Ideation 과정에서 논문과 함께 코드의 흐름을 이해하는 작업이 선행되었음에도 완벽히 코드를 이해했다는 생각이 들지 않았다. 이러한 아쉬움을 knowledge distillation 을 구현하면서 많이 해소할 수 있었다. 물론 모든 코드의 구성과 흐름을 완벽히 이해했다고 할 수는 없겠지만, 해당 기능을 구현하면서 dataloader를 새로 구현하기도 하고 RAM memory 오류를 마주하기도 하며 densephrases 를 보다 더 이해하고 활용할 수 있는 기회가 되었다.
      knowledge distillation 은 densephrases 의 fine-tuning 과정에서 사용되는 cross-encoder 모델을 query-side fine tuning 환경에서도 활용하여phrase를 더 정확한 확률로 예측하고 싶다는 목적을 가지고 구현되었다. 해당 아이디어의 기저는 phrase를 retrieval unit으로 반환할 때에도 높은 정확도를 가지게 된다면 이를 활용해 prompt 경량화가 가능할 것이라는 동기가 첫번째였는데, phrase 와 sentence 를 섞어 반환하는 multi-unit retrieval 이 아니더라도 실제 단답만이 필요한(문맥이 필요하지 않은) query 가 입력되었을 때 정확한 phrase retrieval 이 도움이 될 수 있을 것이라고 생각했기 때문이다. 두번째는 정확한 phrase 의 예측이 sentence retrieval 의 mAR 향상에도 도움이 될 수 있다는 동기였는데, 이를

증명하기 위하여 실험한 결과 실제로 L_query+L_doc 등으로 gold document 를 보다 잘 예측할 수 있도록 한 후 knowledge distillation 을 적용한 결과 QSFT 에서 시도한 방법론 중 가장 높은 mAR score 를 확인할 수 있었다. 이는 retrieval algorithm 수정 측면의 multi-unit retrieval 등과 함께 사용하여 mAR 을 더 높일 수 있다는 측면에서 2번 목표가 달성되었다고 할 수 있다.

  - 3번 목표를 위한 시도: Dataset 전처리, Query loss의 단위 변경(refined_title_L_doc)
    이전까지의 프로젝트는 data centric project 를 제외하고는 주로 모델링 과정에서 기여했던 것 같다. 때문에 이번 프로젝트에서는 data 측면에서 팀에 기여하고 싶다고 생각 하였고, L_context 와 L_sentece 구현을 위해 natural questions single-qa data 를 전처리 하는 역할을 수행하였다. 데이터를 탐색하고, 기능 구현을 위해 densephrases 에서 사용하는 방법을 고려하여 전처리 하면서 흥미로움과 함께 책임감도 느낄 수 있었다. 혹시나 잘못 처리한 데이터 때문에 기능 구현 과정에서 문제가 생기거나, 데이터 전처리 과정이 너무 늦어져서 일정이 지체되지 않도록 주의해야 겠다는 생각이 들었기 때문이다. 그 만큼 해당 데이터를 기반으로 기능들이 잘 구현되는 것을 보면서 뿌듯함을 느낄 수 있었다.

    또, train_data 의 일부 title 에 wiki data 에는 없는 suffix가 붙어있어 L_doc 의 학습에 방해가 될 수 있다는 판단을 기반으로 train data 의 title 을 전처리하여 suffix를 제거하는 refined_title_L_doc 을 학습하였다. 그러나 단일 Loss로는 L_query(base) 를 넘지 못하였다. (mAR 0.5874)

- 마주한 한계 **&** 아쉬웠던 점
  - densephrases 는 무엇보다도 학습과정에 많은 시간이 소요된다는 점이 한계와 아쉬움으로 남았다. 디버깅시에도 실제 학습시에도 크기가 매우 큰 wiki data 의 index 파일을 사용하는 과정에서 시간 소요가 매우 컸기 때문에 기능 구현에 시간이 많이 소요되었고, 결과적으로 다양한 시도를 해보지 못한 것이 아쉬움으로 남는다.
- 한계 **&** 교훈을 바탕으로 다음 프로젝트에서 시도해볼 것들
  - 이후의 프로젝트에서는 코드를 분석하는 과정과 가설을 구현하는 과정을 병행하는 시도를 하고 싶다. 물론 올바른 가설 제안을 위해서는 어느 정도의 코드 분석 과정이 선행되어야 하겠지만, 이 과정을 최대한 간단히 수행하고 가설 구현에 필요한 흐름을 보다 정확히 분석하는 관점에서 접근하는 것이 시간 절약에 도움이 될 것이라고 판단하였기 때문이다.

서가은

- 모델 개선 방법
  - 위에 서술한 L_sentence를 구현했다. 처음에는 해당 Loss가 문장 단위로 retrieval하는 우리 task에 좋은 영향을 끼칠 것이라 예상하였으나, 모델이 gold sentence를 맞추는 경우가 거의 없어서 오히려 mAR의 하락을 불러일으켰다. (0.6316 → 0.6251)
  - 위에는 서술하지 않은 index별 multi unit retrieval 방법을 구현했다. index와는 무관하게 두 개의 인코더로 각각 나온 결과를 하나로 합쳐서 score를 기준으로 재정렬하는 방법인 앞서 언급한 dynamic retrieval 방법과 달리, 동일한 index에서 L_query 인코더로 계산한 score와 L_query + L_doc 인코더로 계산한 score를 비교하여 L_query가 더 높게 나오면 phrase를,

L_query + L_doc이 더 높게 나오면 sentence를 반환하도록 하는 방법이다. 그러나, 결과적으로 0.6152 정도의 mAR 수치를 기록했다. 이는 baseline인 L_query의 mAR 값인 0.6316보다 0.0164 하락한 값이었다.

- 아쉬웠던 점
  - index별 multi unit retrieval 방법을 구체화할 시간이 부족해서 아쉬웠다. 해당 방법은 코드 면에서 더 보완해야 할 부분이 많이 있었고, 더 다양하게 시도할 수 있었는데, 프로젝트 막바지이다 보니 보완할 시간이 부족해서 구체화하지 못했다.
  - 원본 코드 자체에 존재하던 심각한 문제점을 프로젝트 막바지에 가서야 깨달았다. 원본 코드 자체가 모듈화가 많이 되어있고 거대하고 복잡한 코드라서 모든 코드를 하나 하나 뜯어보지는 못했는데, 원본 코드 자체에 중요한 전처리가 적용되지 않는 큰 문제가 있었다. 때문에 몇주간 했던 실험들이 전부 training과 inference 시 전처리가 서로 다르게 적용되었다는 사실을 알게 되었다. 이를 수습하기 위해 다시 제대로 된 환경에서 실험을 한 결과 이전과 많이 다른 결과들이 반환되었고, 팀 내에서도 큰 혼란이 있었다.
  - training 시 학습 시간이 너무 오래 걸려서 training 과정에서 다양한 방법들을 적용해보지 못했다. fine tuning 시 1epoch당 평균 3시간이 소요되었고, L_sentence의 경우에는 phrase가 속한 sentence를 찾아내는 과정 때문에 1epoch당 평균 9시간이 소요되었기 때문에 한번 학습을 시작하면 오랜 시간 동안 할 수 있는 게 없었다.
  - 협업이 효율적으로 이루어지지 못했다. 초창기 계획은 모두가 하나의 가설을 구체화하자는 목표가 있었는데, 프로젝트가 진행되며 각자 자연스럽게 개인이 하고 싶었던 것을 하는 방향으로 바뀌었다. 이러한 연구 프로젝트에서의 분업은 어떻게 해야 하는지, 또 모두가 하나의 가설을 구체화하는 방향보다는 각자 해보고 싶은걸 하는 방향이 애초에 맞는건지 의문이 들었다.
- 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것
  - baseline 코드가 아무리 복잡해도 변수 하나 하나 다 뜯어봐야겠다는 생각이 들었다.
  - 모델이 학습되는 중에도 해당 방법을 어떻게 구체화할 수 있는지, 또는 어떤 새로운 방법을 적용할 수 있는지 고민해봐야겠다.

홍영훈

- 프로젝트 목표
  - 코드를 하나씩 살펴보며 baseline 코드를 완벽히 이해하기
  - 논문을 읽고 논문에서 아이디어를 얻어서 구현하기
- 목표를 위해 시도한 것
  - Densephrases 코드를 보고 이해하는데 초반에 베이스라인을 이해하는데 집중하고 나중에 아이디어를 다시 내보고 싶었으나, 일정이 촉박하여 베이스라인 이해와 실험을 병행하게 되었다. 그러다보니 대충 넘어간 부분에서 실험 실수를 하기도 했다. 그래도 다행인 점은 결국엔 nota에서 제안한 부분의 코드를 대부분 읽어볼 수 있었다. 이러한 과정에서 코드에서 이상한 점이 발견되기도 하고 이해를 좀 더 정확히 할 수 있었던 것 같다. (QSFT 과정에서 loss 계산 시 정답이 없으면 loss에 0을 주는 부분, truecase 이슈 등등)

- 이번 프로젝트에서는 논문을 읽고 어떤 아이디어를 구현하려고 했으나 관련 주제에 대한 논문이 많지 않았다. 그래도 densephrases 관련된 논문을 찾아보았는데, nota에 애초에 제안한 query optimization 부분은 inference 시간이 너무 오래 걸리게 되어 실행할 수 없었다. 논문에서 아이디어를 얻지 못하니 자연스럽게 현재 모델이 못하는 부분을 찾고 개선점을 찾게 되었다. 덕분에 phrase + sentence를 해야하는 이유 등을 얻을 수 있었다.

- 총평
  - baseline을 제대로 이해하는게 실험에 있어서 중요한 부분을 차지하는 것을 다시 한 번 알게되었다. 제대로 이해하지 않는 순간 실험은 실수할 확률이 높아진다는 것 또한 실수로부터 배울 수 있었다. 항상 그렇지만 다음 실험에서는 실험을 좀 더 체계화 할 수 있는 방법을 개발해야할 필요성을 느꼈다.
  - 내가 시도한 static 같은 방식은 머신러닝을 한다기 보다는 그냥 문제해결을 위한 수학문제를 푸는 느낌을 받았다. 그렇다고 모델링 작업을 하기에는 아직 나에게 실험을 시작하기 위한 근거가 부족하다고 느꼈다. 엔지니어링 관점이 아닌 머신러닝 관점에서의 모델링을 작업하기 위해서는 완벽한 근거를 찾기는 힘들다고 생각이 들었다. 확실히 아직 해당 분야에 대한 지식이 부족하기 때문에 더욱 어려운 것 같다고 느꼈다. 또한 실험 하나하나가 시간이 오래 걸리는 만큼 더욱 실험 설계가 중요하고 확률이 높은 실험을 선별하는 능력이 중요하다고 느꼈다.