

DPR을 이용한 검색 솔루션

(with  **WISEnut**)

다차원 벡터 활용을 통한 문서 유사도 검색

팀명 현명한호두

팀장 홍세일

팀원 정재훈 이정호 유예지 조은상

목차

table of contents

- 1 프로젝트 배경
- 2 Dense Passage Retrieval
- 3 Pretraining Model
- 4 유사도 엔진
- 5 검색 엔진 연동
- 6 기대효과 및 한계점



1

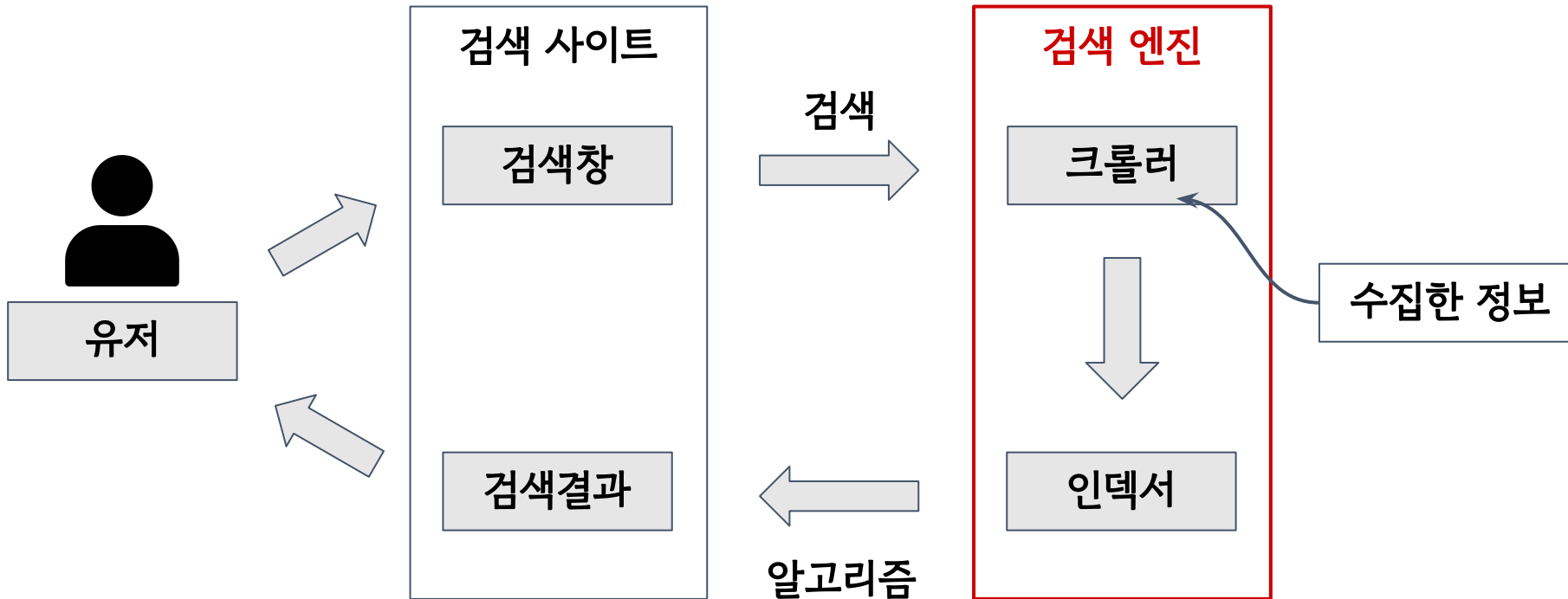
프로젝트 배경

'검색 엔진' 이란?

1. 프로젝트 배경

- 검색 엔진 (Search Engine)

→ 사용자가 웹에서 특정 정보를 찾을 때, 검색어를 입력하면 그 검색어와 관련된 웹페이지를 찾아주는 프로그램



BM25 / TF-IDF

키워드 파악은 가능하나
문맥 및 동의어를 구분하지
못함



ORQA

장점

- ICT를 사용함으로써 시간 단축
- 추가적인 pretraining 없이 적은 데이터로
dense vector/matrix를 활용하여 모델 생성 가능

단점

- Regular sentence 가 좋은지 명확하지 않음
- Fine-tuning 을 하지 않으므로 suboptimal 이 될 수 있음



DPR

in-batch negative
sampling 사용

문맥 및 동의어
파악 가능

2

Dense Passage Retrieval

참고 논문

Dense Passage Retrieval for Open-Domain Question Answering

<https://arxiv.org/pdf/2004.04906.pdf>

'DPR (Dense Passage Retrieval)' 이란?

2. Dense Passage Retrieval

< Retrieval 의 Backbone Model >

'사전훈련된 BERT model' 로 이루어진
'2개의 Encoder' (Question Encoder,
Passage Encoder)' 활용

< 유사도 측정 방법 >

question 벡터와 passage 벡터의 **내적**
(dot product)

$$\text{sim}(q, p) = E_Q(q)^T E_P(p).$$

< Model Training >

loss 를 구하는 식 :

NLL (Negative Log Likelihood)

$$L(q_i, p_i^+, p_{i,1}^-, \dots, p_{i,n}^-) \\ = -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^n e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

< Negative sample >

loss 가 감소하는 방향으로 학습하기 위해 중요

- Random : 코퍼스 내의 random한 passage를 선택
- BM25 : 코퍼스 내에서 BM25 기준으로 top-k의 문서 사용
- Gold : 학습셋 내의 다른 질의의 positive passage 선택

< In-batch negatives >

모델 학습 시 같은 batch 내에서 (In-batch) gold passage 를 negative passage 로 활용하는 것 (**재사용**)
계산적인 효율 뿐만 아니라, 좋은 성능을 낼 수 있게 함

$$S = QP^T S = QP^T \text{ (BxB)}$$

< test set 의 retrieval 정확도 >

Training0	Retriever	top20					top-100				
		NQ	TriviaQA	WQ	TREC	SQuAD	NQ	TriviaQA	WQ	TREC	SQuAD
None	BM25	59.1	66.9	55.0	70.9	68.8	73.7	76.7	71.1	84.1	80.0
Single	DPR	78.4	79.4	73.2	79.8	63.2	85.4	85.0	81.4	89.1	77.2
	BM25+ DPR	78.4	79.8	71.0	85.2	71.5	83.8	84.5	80.5	92.7	81.3
Multi	DPR	79.4	78.8	75.0	89.1	51.6	86.0	84.7	82.9	93.9	67.6
	BM25+ DPR	78.0	79.9	74.7	88.5	66.2	83.9	84.4	82.3	94.1	78.6

- Single : 각각의 Dataset에 대하여 학습시킴
- Multiple : SQuAD를 제외한 4개의 Dataset을 합쳐서 학습시킴
- 40 epoch (NQ, TriviaQA, SQuAD)
- 100 epoch (TREC, WQ)
- dropout ratio : 0.1
- optimizer : Adam
- top k : 상위 k개의 retrieval 정확도를 가지고 측정

DPR 학습 결과

< NQ 데이터셋에 대해
서로 다른 학습 방법을 테스트한 결과 >

Type	#N	IB	Top-5	Top-20	Top-100
Random	7	✗	47.0	64.3	77.8
BM25	7	✗	50.0	63.3	74.8
Gold	7	✗	42.6	63.1	78.3
Gold	7	✓	51.1	69.1	80.8
Gold	31	✓	52.1	70.8	82.1
Gold	127	✓	55.8	73.0	83.1
G.+BM25 ⁽¹⁾	31+32	✓	65.0	77.3	84.4
G.+BM25 ⁽²⁾	31+64	✓	64.5	76.4	84.0
G.+BM25 ⁽¹⁾	127+128	✓	65.8	78.0	84.9

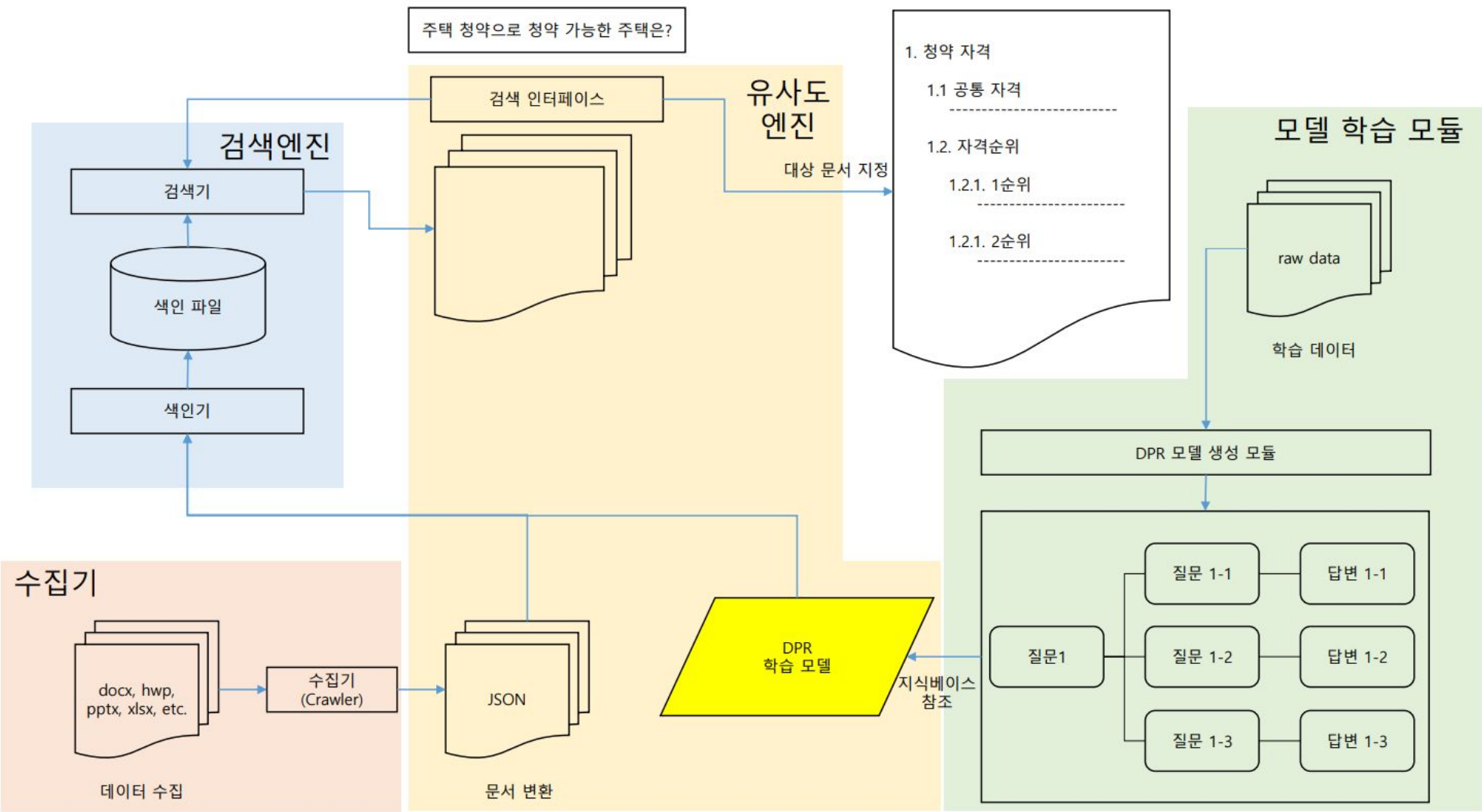
#N : Negative Sample 개수 , IB : In - Batch 적용여부

2. Dense Passage Retrieval

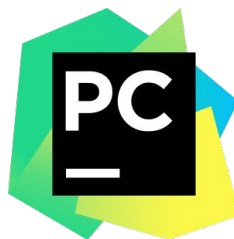
< end-to-end QA accuracy >

Training	Model	NQ	TriviaQA	WQ	TREC	SQuAD
Single	BM25+BERT (Lee et al., 2019)	26.5	47.1	17.7	21.3	33.2
Single	ORQA (Lee et al., 2019)	33.3	45.0	36.4	30.1	20.2
Single	HardEM (Min et al., 2019a)	28.1	50.9	-	-	-
Single	GraphRetriever (Min et al., 2019b)	34.5	56.0	36.4	-	-
Single	PathRetriever (Asai et al., 2020)	32.6	-	-	-	56.5
Single	REALM _{Wiki} (Guu et al., 2020)	39.2	-	40.2	46.8	-
Single	REALM _{News} (Guu et al., 2020)	40.4	-	40.7	42.9	-
Single	BM25	32.6	52.4	29.9	24.9	38.1
	DPR	41.5	56.8	34.6	25.9	29.8
	BM25+DPR	39.0	57.0	35.2	28.0	36.7
Multi	DPR	41.5	56.8	42.4	49.4	24.1
	BM25+DPR	38.8	57.9	41.1	50.6	35.8

프로젝트 구조도



사용한 tools



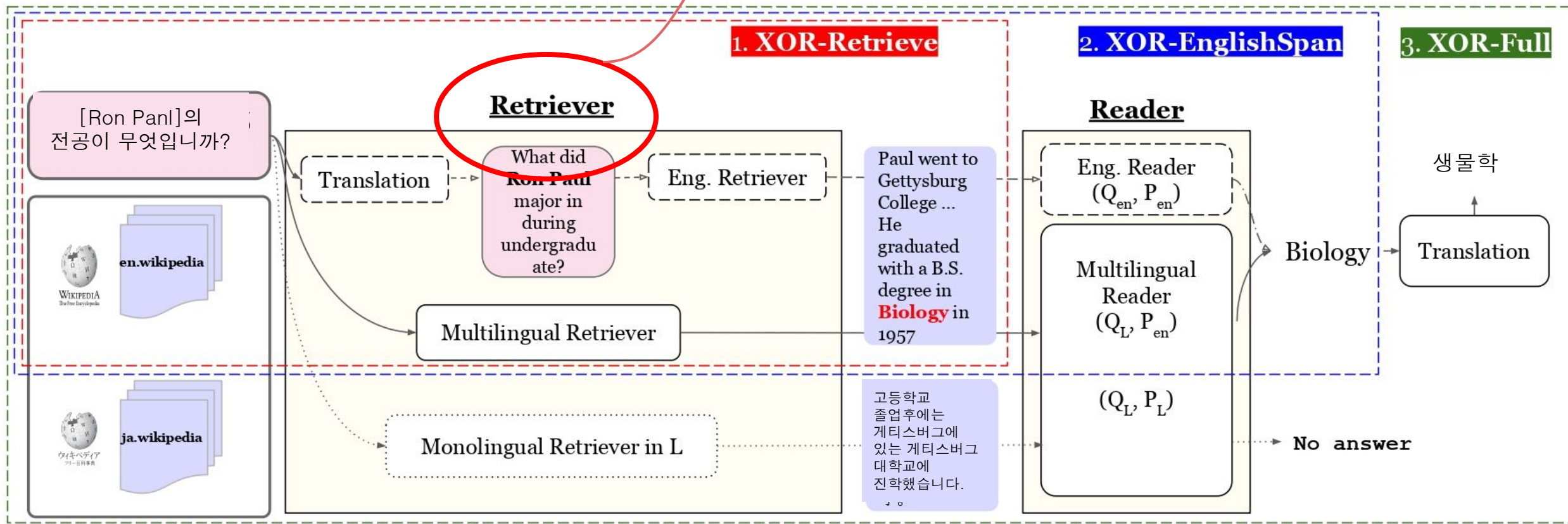
3

Pretraining Model

DPR 구조

3. Pretraining Model

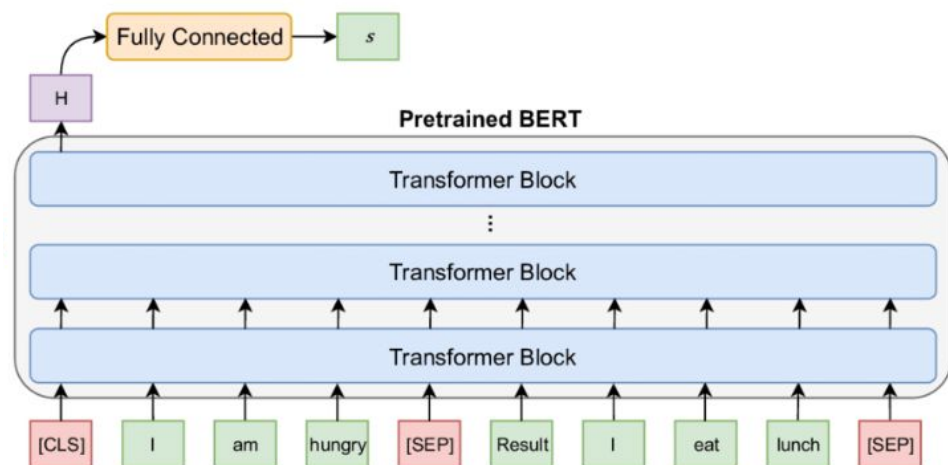
Bert 모델 사용 → 한국어 처리를 위해 **Kobert** 로 변경



BERT , KoBERT Model

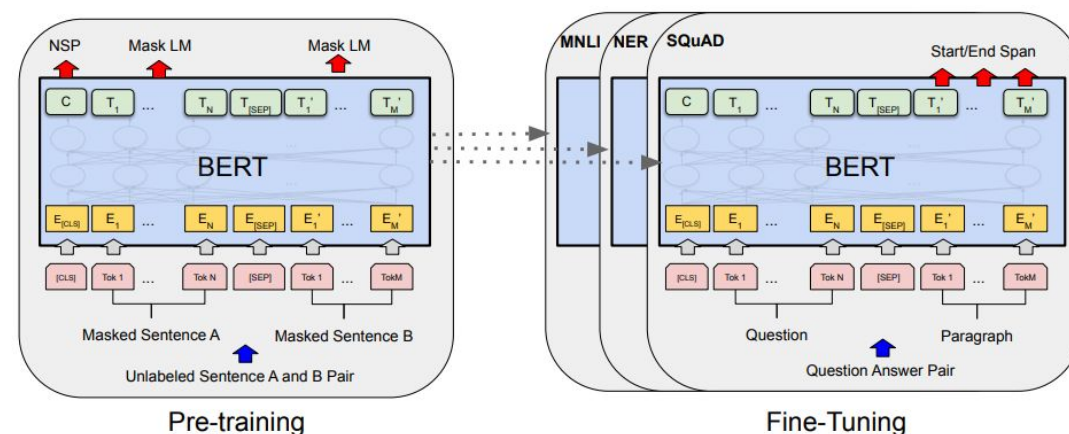
3. Pretraining Model

< BERT Model >



- 약 33억개의 단어로 pretrain 되어있는 기계번역 모델
- 사용목적에 따라 fine-tuning 이 가능
- 텍스트를 양방향 (앞뒤) 로 확인하여 자연어 처리
- 영어의 정확도는 높지만, 한국어의 정확도는 낮음

< KoBERT Model >



- BERT Model 에서 “**한국어 데이터**” 를 추가로 학습시킨 모델 (한국어 위키에서 5백만개의 문장과 5,400만개의 단어를 학습시킨 모델)
- **한국어 버전 BERT Model**
(한국어 데이터에서 높은 정확도)

Input data

3. Pretraining Model

< input_train.json >

```
[{'question': '바그너는 괴테의 파우스트를 읽고 무엇을 쓰고자 했는가?',  
  'answers': ['교향곡'],  
  'positive_ctxs': [{'title': '파우스트 서곡',  
    'text': '1839년 바그너는 괴테의 파우스트를 처음 읽고 그 내용에  
    마음이 끌려 이를 소재로 해서 하나의 교향곡을 쓰려는 뜻을 갖는다. 이  
    시기 바그너는 1838년에 빛 독촉으로 산전수전을 다 겪은 상황이라  
    좌절과 실망에 가득했으며 메피스토펠레스를 만나는 파우스트의 심경에  
    공감했다고 한다. 또한 파리에서 아브네크의 지휘로 파리 음악원  
    관현악단이 연주하는 베토벤의 교향곡 9번을 듣고 깊은 감명을 받았는데,  
    이것이 이듬해 1월에 파우스트의 서곡으로 쓰여진 이 작품에 조금이라도  
    영향을 끼쳤으리라는 것은 의심할 여지가 없다. 여기의 라단조 조성의  
    경우에도 그의 전기에 적혀 있는 것처럼 단순한 정신적 피로나 실의가  
    반영된 것이 아니라 베토벤의 합창교향곡 조성의 영향을 받은 것을 볼 수  
    있다. 그렇게 교향곡 작곡을 1839년부터 40년에 걸쳐 파리에서  
    착수했으나 1악장을 쓴 뒤에 중단했다. 또한 작품의 완성과 동시에 그는  
    이 서곡(1악장)을 파리 음악원의 연주회에서 연주할 파트보까지  
    준비하였으나, 실제로는 이루어지지 않는 않았다. 결국 초연은 4년 반이 지난  
    후에 드레스덴에서 연주되었고 재연도 이루어졌지만, 이후에 그대로  
    방치되고 말았다. 그 사이에 그는 리엔치와 방황하는 네덜란드인을  
    완성하고 탄호이저에도 착수하는 등 분주한 시간을 보냈는데, 그런 바쁜  
    생활이 이 곡을 잊게 한 것이 아닌가 하는 의견도 있다.'}]},  
  ...
```

< input_dev.json >

```
[{'question': '임종석이 여의도 농민 폭력 시위를 주도한 혐의로  
지명수배 된 날은?',  
  'answers': ['1989년 2월 15일'],  
  'positive_ctxs': [{'title': '임종석',  
    'text': '1989년 2월 15일 여의도 농민 폭력 시위를 주도한 혐의  
    (폭력행위등처벌에관한법률위반)으로 지명수배되었다. 1989년 3월 12일  
    서울지방검찰청公安부는 임종석의 사전구속영장을 발부받았다. 같은 해  
    6월 30일 평양축전에 임수경을 대표로 파견하여 국가보안법위반 혐의가  
    추가되었다. 경찰은 12월 18일~20일 사이 서울 경희대학교에서  
    임종석이 성명 발표를 추진하고 있다는 첩보를 입수했고, 12월 18일  
    오전 7시 40분 경 가스총과 전자봉으로 무장한 특공조 및 대공과 직원  
    12명 등 22명의 사복 경찰을 승용차 8대에 나누어 경희대학교에  
    투입했다. 1989년 12월 18일 오전 8시 15분 경 서울청량리경찰서는  
    호위 학생 5명과 함께 경희대학교 학생회관 건물 계단을 내려오는  
    임종석을 발견, 검거해 구속을 집행했다. 임종석은 청량리경찰서에서 약  
    1시간 동안 조사를 받은 뒤 오전 9시 50분 경 서울 장안동의  
    서울지방검찰청 公安분실로 인계되었다.'}]},  
  {'question': '1989년 6월 30일 평양축전에 대표로 파견 된  
    인물은?',  
    'answers': ['임수경'],  
    'positive_ctxs': [{'title': '임종석',  
      'text': '1989년 2월 15일 여의도 농민 폭력 시위를 주도한 혐의  
      (폭력행위등처벌에관한법률위반)으로 지명수배되었다. 1989년 3월 12일  
      서울지방검찰청 公安부는 임종석의 사전구속영장을 발부받았다.  
      ...
```

KoBERT code

3. Pretraining Model

1. RESOURCE_MAP에 한국어 데이터 추가

```
"data.retriever.input-dev": {  
    "s3_url": "https://dl.fbaipublicfiles.com/dpr/data/retriever/blencoder-squad1-train.json.gz",  
    "original_ext": ".json",  
    "compressed": False,  
    "desc": "SQUAD 1.1 train subset with passages pools for the Retriever training".  
},  
"data.retriever.input-train": {  
    "s3_url": "https://dl.fbaipublicfiles.com/dpr/data/retriever/blencoder-squad1-train.json.gz",  
    "original_ext": ".json",  
    "compressed": False,  
    "desc": "SQUAD 1.1 train subset with passages pools for the Retriever training".  
},
```

2. encoder train default.yaml과 retriever default.yaml 수정

```
input_train:  
    _target_: dpr.data.blencoder_data.JsonQADataset  
    file: data.retriever.input-train  
  
input_dev:  
    _target_: dpr.data.blencoder_data.JsonQADataset  
    file: data.retriever.input-dev
```

3. hf_bert.yaml에서 하이퍼 파라미터 수정

```
# @package _group_  
  
# model type. One of [hf_bert, pytorch_bert, fairseq_roberta]  
encoder_model_type: hf_bert  
  
# HuggingFace's config name for model initialization  
pretrained_model_cfg: skt/kobert-base-v1  
  
# Some encoders need to be initialized from a file  
pretrained_file:  
  
# Extra linear layer on top of standard bert/roberta encoder  
projection_dim: 0  
  
# Max length of the encoder input sequence  
sequence_length: 512  
  
dropout: 0.1  
  
# whether to fix (don't update) context encoder during training or  
not  
fix_ctx_encoder: False  
  
# If False, the model won't load pre-trained BERT weights  
pretrained: True
```


4. hf_model.py의 kobert 변환

```
def get_bert_tokenizer(pretrained_cfg_name: str, do_lower_case: bool = True):
    return KoBERTTokenizer.from_pretrained(pretrained_cfg_name,
do_lower_case=do_lower_case)

class BertTensorizer(Tensorizer):
    def __init__(self, tokenizer: KoBERTTokenizer, max_length: int, pad_to_max: bool =
True):
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.pad_to_max = pad_to_max

if transformers.__version__.startswith("4"):
    from transformers import BertConfig, BertModel
    from transformers import AdamW
    from kobert_tokenizer import KoBERTTokenizer
    from transformers import RobertaTokenizer
```

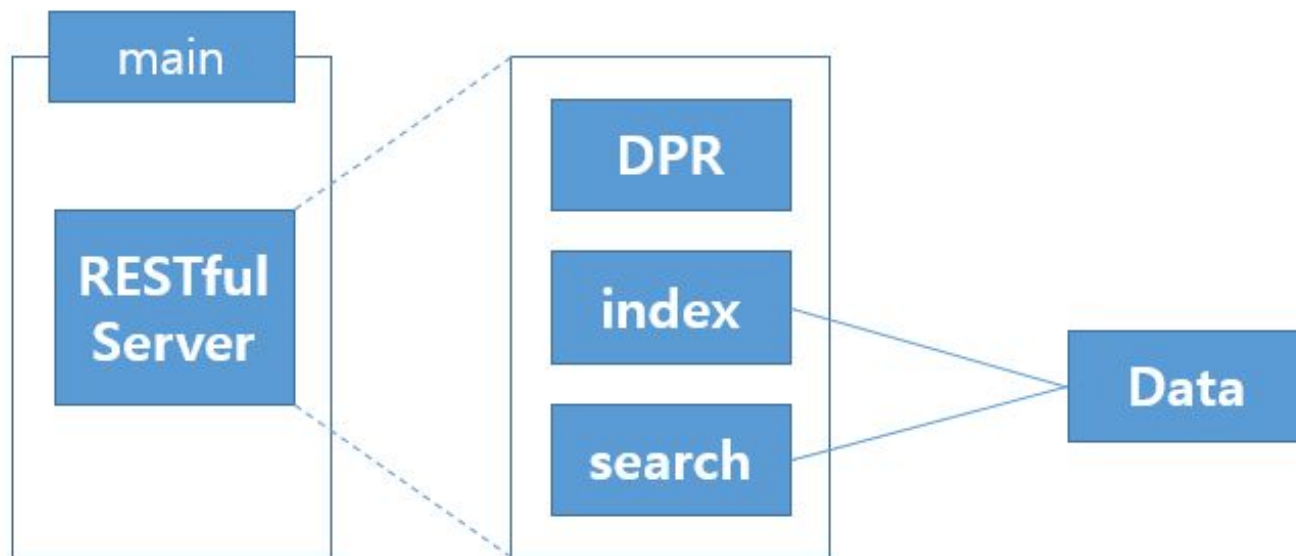
5. hf_model.py의 truncation 전략 수정

```
class BertTensorizer(Tensorizer):
    def __init__(self, tokenizer: KoBERTTokenizer, max_length: int, pad_to_max:
bool = True):
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.pad_to_max = pad_to_max

    if title:
        token_ids = self.tokenizer.encode(
            title,
            text_pair=text,
            add_special_tokens=add_special_tokens,
            max_length=self.max_length if apply_max_len else 10000,
            pad_to_max_length=False,
            truncation="only_second",
        )
    else:
        token_ids = self.tokenizer.encode(
            text,
            add_special_tokens=add_special_tokens,
            max_length=self.max_length if apply_max_len else 10000,
            pad_to_max_length=False,
            truncation="only_second",
        )
```

4

유사도 엔진



```
def get_idx(*args):
    idxs = []
    for _ in args:
        idxs.append(tokenizer(_, return_tensors="pt")["input_ids"])
    return idxs

def get_pooleroutput(List):
    embeddings = []
    for _ in List:
        embeddings.append(model(_).pooler_output)
    return embeddings

def dot_product_scores(q_vectors: T, ctx_vectors: T) -> T:
    """
    calculates q->ctx scores for every row in ctx_vector
    :param q_vector:
    :param ctx_vector:
    :return:
    """
    # q_vector: n1 x D, ctx_vectors: n2 x D, result n1 x n2
    r = torch.matmul(q_vectors, torch.transpose(ctx_vectors, 0, 1))
    return r
```

```
def cosine_scores(q_vector: T, ctx_vectors: T):
    # q_vector: n1 x D, ctx_vectors: n2 x D, result n1 x n2
    return F.cosine_similarity(q_vector, ctx_vectors, dim=1)

def get_total_scores(q_vector: T, ctx_vectors: T):
    cos_score=cosine_scores(q_vector, ctx_vectors)
    dot_pdt_score=dot_product_scores(q_vector, ctx_vectors)
    total_score = cos_score
    return round(total_score.item(), 4)

def get_title_dpr(item):
    _ = tokenizer(item, return_tensors="pt")["input_ids"]
    input_data = _.to(device)
    res = model(input_data).pooler_output.detach().numpy().tolist()
    return res

def get_content_dpr(item):
    _ = tokenizer(item,
    return_tensors="pt", truncation=True, max_length=512)["input_ids"]
    input_data = _.to(device)
    res = model(input_data).pooler_output.detach().numpy().tolist()
    return res
```

1. DPR : 두 문장 간의 유사도 파악

문장 유사도 찾기

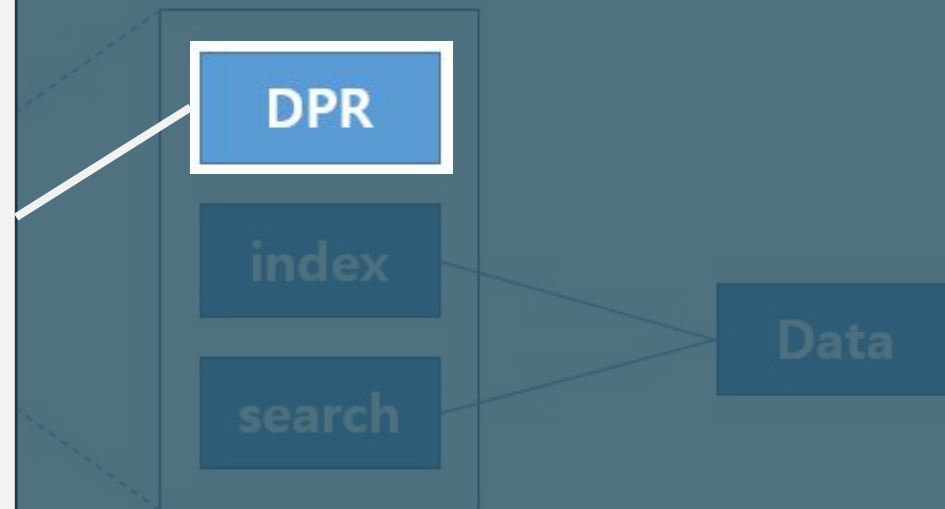
← → ↻ 주의 요함 | 172.30.1.53:5000/similarity/172.30.1.83

CAKD Home **similarity** index search

문장 1:

문장 2:

submit



Index code : index.py

```
mecab = Komoran()

def file_save(f, fname, upload_folder):
    try :
        file_path = os.path.join(upload_folder, fname)
        print(file_path)
        f.save(file_path)
        if fname in os.listdir(upload_folder):
            print('success')
            return (True, file_path)
        else :
            print('failed')
            return (False, file_path)

    except Exception as e:
        print('-----')
        print(e)
        return (False, file_path)

        new_dict['title'] = item['title']
        new_dict['title_dpr'] = title_dpr

return_tensors="pt", truncation=True, max_length=512)["input_ids"]).pooler_output.detach().numpy().tolist()

        content_dpr = get_content_dpr(item['content'])
        new_dict['content'] = item['content']
        new_dict['content_dpr'] = content_dpr
        new_dict['content_morphs'] = get_morphs(item['content'])
        new_dict['title_morphs'] = get_morphs(item['title'])
        new_dict['DOCID'] = item['DOCID']
        es.index(index='my_index', body=new_dict, id = new_dict['DOCID'])
        end = time.time()
        count += 1
        print(count, ': ', end-start)

        #bulk(es, json_file, index = 'my_index')
return True
```

```
def file_upload_in_db(db, document_obj):
    try:
        db.session.add(document_obj)
        db.session.commit()
        return True
    except:
        return False

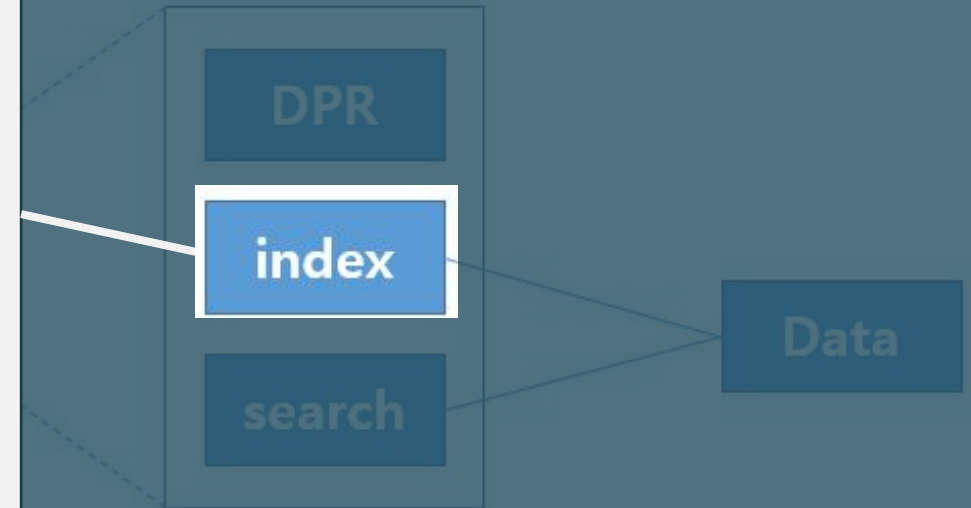
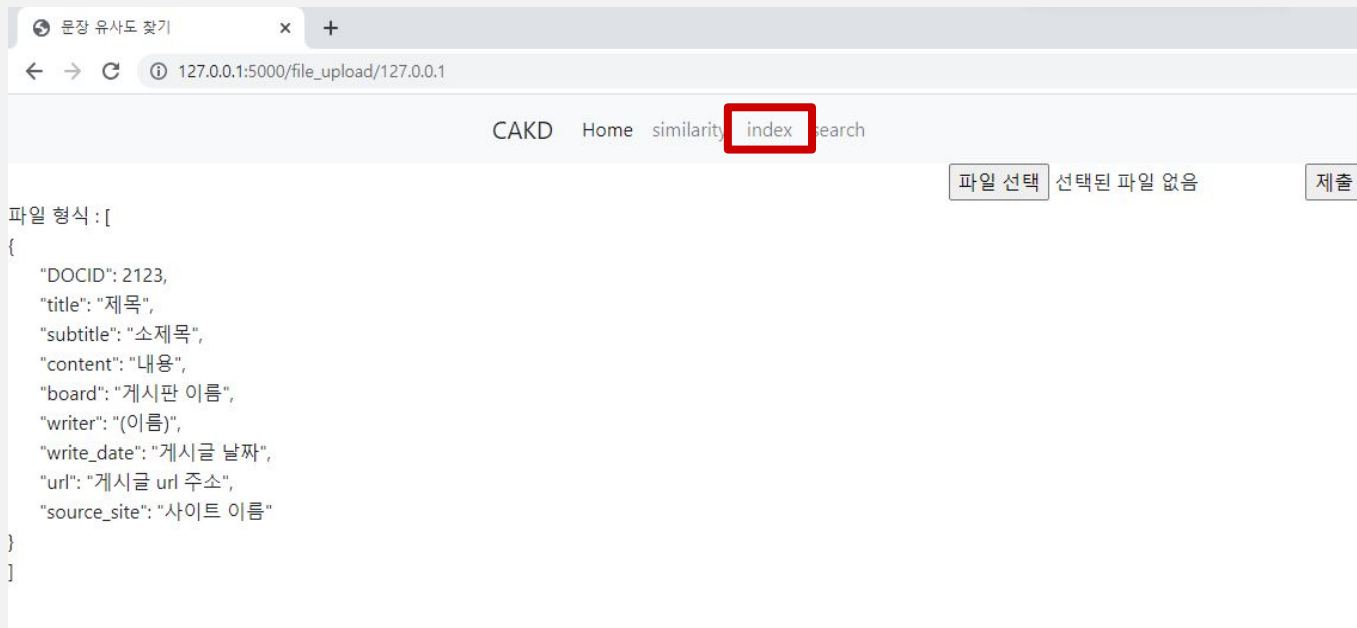
def get_morphs(item):
    res = mecab.morphs(item.replace('[^ㄱ-ㅎa-zA-Z0-9]', ''))
    return res

def file_indexing(path):
    with open(path, 'r', encoding='utf-8') as f:
        json_file=json.load(f)
        count = 0
        for item in json_file:
            start = time.time()
            new_dict = {}
            #title_dpr = model(tokenizer(item['title'],
return_tensors="pt")["input_ids"]).pooler_output.detach().numpy().tolist()

            title_dpr = get_title_dpr(item['title'])
            content_dpr = get_content_dpr(item['content'])
            new_dict['content'] = item['content']
            new_dict['content_dpr'] = content_dpr
            new_dict['content_morphs'] = get_morphs(item['content'])
            new_dict['title_morphs'] = get_morphs(item['title'])
            new_dict['DOCID'] = item['DOCID']
            es.index(index='my_index', body=new_dict, id = new_dict['DOCID'])
            end = time.time()
            count += 1
            print(count, ': ', end-start)

        #bulk(es, json_file, index = 'my_index')
return True
```

2. Index : 파일 인덱싱



Search code : data_util.py

```
def get_similarity_in_document(input_text, input_nbest, json_objs):
    for json_obj in json_objs:
        indicies = get_idx(input_text, json_obj['text'])
        embeddings = get_pooleroutput(indicies)
        json_obj['similarity'] = get_total_scores(embeddings[0], embeddings[1])
    json_objs.sort(key=lambda x: -x['similarity'])
    for i, json_obj in enumerate(json_objs):
        json_obj['nbest'] = i+1
        if (i+1) == input_nbest: break;
    return json_objs

def search_in_es(q, n):
    que = q.replace('[^A-Za-z0-9가-힣]', '')
    query_with_tag = mecab.pos(que)
    print(query_with_tag)
    print(mecab.tagset)
    query = []
    for item in query_with_tag:
        if (item[1] == 'XR') | (item[1] == 'NNG') | (item[1] == 'VV') | (item[1] ==
'NNB') | (item[1] == 'NNP') | (item[1] == 'VA') | (item[1] == 'NP') | (item[1] ==
'SN') | (item[1] == 'NA') | (item[1] == 'NR') | (item[1] == 'SL'):
            query.append(item[0])
    print(mecab.pos(que)[0][0])
    print(query)
    data = [] # 반환할 데이터를 담은 리스트를 초기화합니다
    for item in query:
        query = {
            "query": {
                "multi_match": {
                    "query": item,
                    "fields": ["title", "content", "title_morphs", "content_morphs"]
                }
            }
        }
        res = es.search(index = 'my_index', body = query)
        hits = res.get('hits', {}).get('hits', []) # 실제 데이터가 있는 hits 리스트를
추출합니다
```

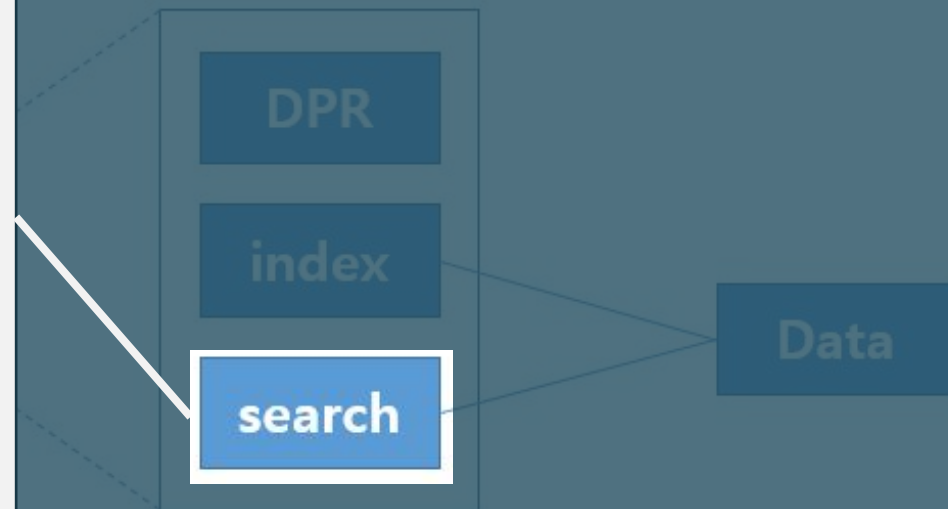
각각의 문서에 대해서 필요한 정보만 추출하여 data 리스트에 추가합니다

```
for hit in hits:
    source = hit.get('_source', {})
    item = {
        'title': source.get('title', ''),
        'content': source.get('content', ''),
        'title_dpr': source.get('title_dpr', []),
        'content_dpr': source.get('content_dpr', []),
    }
    if item not in data:
        data.append(item)

for item in data:
    question = model(tokenizer(q,
return_tensors='pt', truncation=True, max_length=512)["input_ids"]).pooler_output
    ctx = torch.tensor(item['content_dpr'])
    title = torch.tensor(item['title_dpr'])
    score1 = get_total_scores(question, ctx)
    score2 = get_total_scores(question, title)
    item.setdefault('score', score1+score2)
sorted_list = sorted(data, key=lambda x: x['score'], reverse=True)
print(len(sorted_list))
return sorted_list[:n]
```


3. Search : 유사도 top 문장 출력

The screenshot shows a web browser window with the title '문장 유사도 찾기' (Find Similar Sentences). The address bar displays '172.30.1.53:5000/search/172.30.1.83'. The navigation menu includes 'CAKD', 'Home', 'similarity', 'index', and 'search', with 'search' highlighted by a red box. The main content area has a 'query :' label above a text input field. Below it is an 'input n :' label above another text input field. A 'submit' button is located to the right of the second input field.



5

검색 엔진 연동

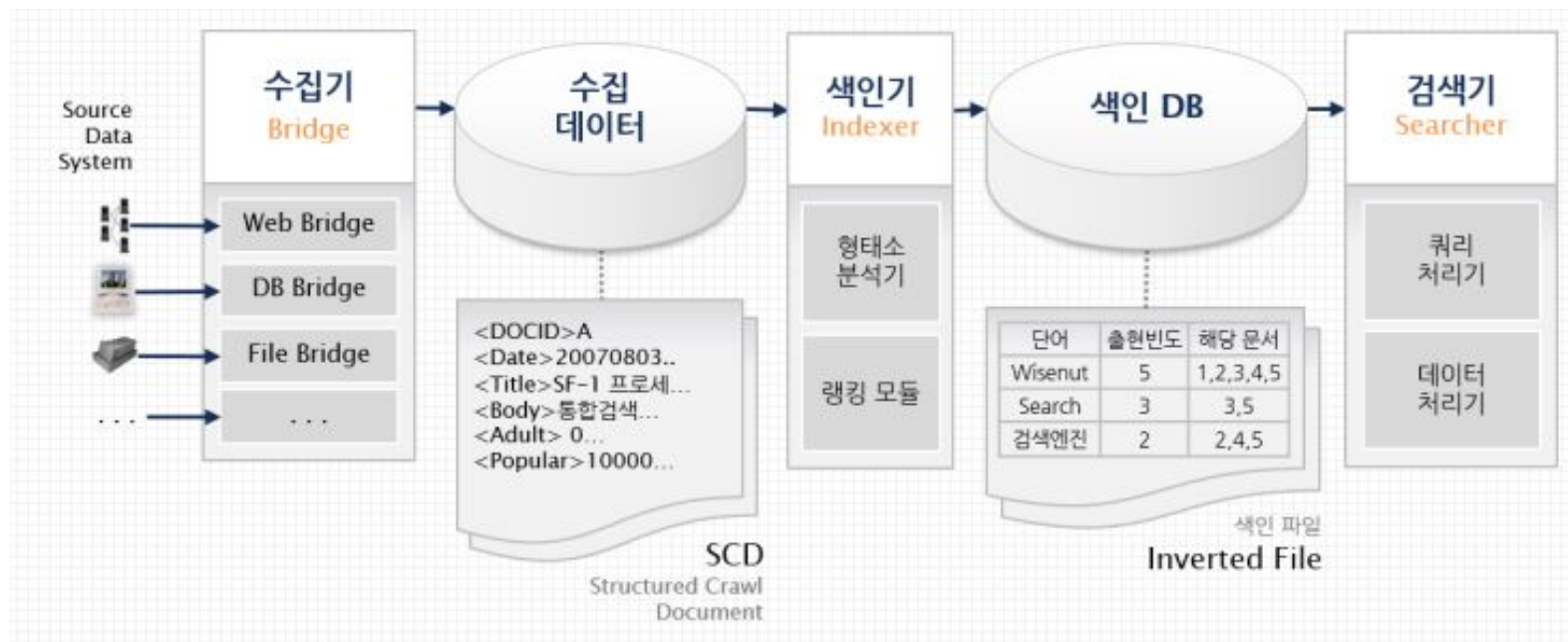
'색인'

5. 검색 엔진 연동

- 색인 (Indexing)

- 데이터베이스나 검색 엔진에서 검색 속도를 높이기 위해 사용되는 기술
- 특정 칼럼에 대한 정렬된 데이터 구조를 생성하여 검색 속도를 높임

- 색인 과정



‘색인어 및 검색어 분석’

- 검색의 기본

- “문서” 혹은 “검색어” 를 어떠한 방식으로 주요 단어를 추출할지 결정
- ‘어떻게 추출하는가’ 에 따라 검색의 품질 및 정확성이 보장

- 색인어 및 검색어 분석 과정

문서 → 레굴레이션 → 색인어 추출 → 색인 DB ← 검색어 추출 ← 레굴레이션 ← 검색어

- 레굴레이션 (Regulation) - 구분자 및 병합자 지정

- 특정 문자를 기준으로 문자열을 나누거나 병합하는 과정
- 분석 대상 문자열을 ‘토큰 (token)’ 으로 분리하고, 해당 토큰을 가지고 색인어 및 검색어 추출기의 입력으로 사용
- 별도의 설정이 없을 때는, “기본 구분자 (대부분의 특수문자)” 를 이용하여 토큰 분리

‘색인어 및 검색어 분석’

- 검색의 기본

- “문서” 혹은 “검색어” 를 어떠한 방식으로 주요 단어를 추출할지 결정
- ‘어떻게 추출하는가’ 에 따라 검색의 품질 및 정확성이 보장

- 색인어 및 검색어 분석 과정

문서 → 레굴레이션 → 색인어 추출 → 색인 DB ← 검색어 추출 ← 레굴레이션 ← 검색어

- 레굴레이션 (Regulation) - 구분자 및 병합자 지정

예시) “안녕하세요? #중앙정보! 홍길동+입니다.” → [안녕하세요], [중앙정보], [홍길동], [입니다]

'역색인'

5. 검색 엔진 연동

● 역색인 (Inverted index)

- 매우 빠른 풀텍스트 검색을 할 수 있도록 설계된 것
- 문서에 나타나는 모든 고유한 단어의 목록을 만들고, 각 단어가 발생하는 모든 문서를 식별
- java는 컴파일 언어이며, python보다 좀 더 빠르기 때문에 java를 사용

● 역색인 코드

```
1 .io.*;
2 .util.*;
3 .util.Map.Entry;
4 .nio.file.*;
5
6
7 s aufgabel {
8   static void main(String[] args) throws IOException {
9
10    System.out.println("Loading "+args[0]+" file");
11    long start = System.currentTimeMillis();
12    //비파괴적인 파일크기해 정황을 받음 파일크기는 약 35000byte이고 전체를 받으려 할것으로 생각됨
13    BufferedReader reader = new BufferedReader(
14      new FileReader("C:\\Users\\admin\\Downloads\\"+args[0]),40960
15    );
16    Path path = Paths.get("C:\\Users\\admin\\Downloads\\"+args[0]);
17    //문장의 개수를 미리 알려 속도향상해 도움이 될것으로 생각해서 문장의 개수 확인
18    int lineCount = (int) Files.lines(path).count();
19    String[] saetze = new String[lineCount];
20    final int [] count = {0};
21    //읽은 문장들을 배열에 저장
22    Files.lines(path).forEach(Line -> saetze[count[0]++] = line);
23    reader.close();
24
25    long end = System.currentTimeMillis();
26    double time = (end - start)/1000.0;
27    System.out.println(String.format("Complete! (%.3fs)", time));
28
29    //읽어쓰기를 사용하면 기존보다 더 빨라질것으로 생각됨
30    //containskey 보다 get함수가 빠르지만 큰 차이 없음
31    long start1 = System.currentTimeMillis();
32
33
34    //해쉬맵을 2중으로 사용
35    HashMap<String, HashMap<Integer,Integer>> doc = new HashMap<String, HashMap<Integer,Integer>>();
36
37    //문장들을 배열처리
38    Iterator<String> iterator = Arrays.stream(saetze).parallel().iterator();
39    while (iterator.hasNext()) {
40      String satz = iterator.next();
41      String[] temp = satz.toLowerCase().replaceAll("[^a-z0-9]", " ").split(" ");
42      temp = Arrays.stream(temp).filter(s -> !s.isEmpty()).toArray(String[]::new);
```

```
40    String satz = iterator.next();
41    String[] temp = satz.toLowerCase().replaceAll("[^a-z0-9]", " ").split(" ");
42    temp = Arrays.stream(temp).filter(s -> !s.isEmpty()).toArray(String[]::new);
43    int doc_id = Integer.parseInt(temp[0]);
44    for (int i = 1; i < temp.length; i++) {
45      String word = temp[i];
46
47
48      if(!doc.containsKey(word)) {
49
50        doc.put(word, new HashMap<Integer,Integer>());
51
52      }
53
54      HashMap<Integer,Integer> innerMap = doc.get(word);
55      innerMap.put(doc_id, innerMap.getOrDefault(doc_id, 0) + 1);
56
57    }
58
59
60
61 }
62 Comparator<HashMap.Entry<Integer, Integer>> valueComparator = new Comparator<HashMap.Entry<Integer, Integer>>() {
63   @Override
64   public int compare(Entry<Integer, Integer> o1, Entry<Integer, Integer> o2) {
65     int compare = o2.getValue().compareTo(o1.getValue()); // comparing values in descending order
66     if (compare == 0) {
67       return o1.getKey().compareTo(o2.getKey()); // comparing keys in ascending order
68     }
69     return compare;
70   }
71 }
72
73 HashMap<String, LinkedHashMap<Integer,Integer>> sortedDoc = new HashMap<>();
74 Set<String> keySet = doc.keySet();
75 List<String> keyList = new ArrayList<>(keySet);
76 Collections.sort(keyList);
77 for (String key : keyList) {
78   HashMap<Integer,Integer> innerMap = doc.get(key);
79   List<Map.Entry<Integer, Integer>> list = new ArrayList<>(innerMap.entrySet());
80   // comparator로 정렬
81   Collections.sort(list,valueComparator);
82   // comparator로 정렬된 결과를 새로운 해쉬맵에 저장
83   LinkedHashMap<Integer,Integer> innerMapSorted = new LinkedHashMap<>();
84   for(HashMap.Entry<Integer, Integer> innerEntry : list) {
```

```
81    Collections.sort(list,valueComparator);
82    // comparator로 정렬된 결과를 새로운 해쉬맵에 저장
83    LinkedHashMap<Integer,Integer> innerMapSorted = new LinkedHashMap<>();
84    for(HashMap.Entry<Integer, Integer> innerEntry : list) {
85      innerMapSorted.put(innerEntry.getKey(), innerEntry.getValue());
86    }
87    sortedDoc.put(key, innerMapSorted);
88  }
89  /*
90  Set<String> li = sortedDoc.keySet();
91  for (String w : li) {
92    LinkedHashMap<Integer, Integer> inn = sortedDoc.get(w);
93    Set<Integer> inner = inn.keySet();
94    for (Integer docId : inner) {
95      System.out.println(w+" "+ docId + " " + inn.get(docId) + " ");
96    }
97  }
98  */
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

'역색인'

5. 검색 엔진 연동

- 역색인 (Inverted index)

→ 매우 빠른 폴텍스트 검색

→ 문서에 나타나는 모든

→ java는 컴파일 언어이기

- 역색인 코드

```
loading input.big file
Complete! (0,176s)
Complete! (5,762s)
total Complete! (5,938s)
input : frowning
Doc: 11396, Frequency: 1
Doc: 82860, Frequency: 1
Doc: 104083, Frequency: 1
Doc: 249603, Frequency: 1
```

```
input : happy
Doc: 64017, Frequency: 3
Doc: 8929, Frequency: 2
Doc: 68177, Frequency: 2
Doc: 71409, Frequency: 2
Doc: 122598, Frequency: 2
Doc: 125899, Frequency: 2
Doc: 143714, Frequency: 2
Doc: 236782, Frequency: 2
Doc: 262076, Frequency: 2
Doc: 911, Frequency: 1
Doc: 1214, Frequency: 1
Doc: 1428, Frequency: 1
Doc: 4452, Frequency: 1
Doc: 6080, Frequency: 1
Doc: 6859, Frequency: 1
Doc: 7005, Frequency: 1
Doc: 7902, Frequency: 1
Doc: 8201, Frequency: 1
Doc: 8333, Frequency: 1
Doc: 9214, Frequency: 1
Doc: 9686, Frequency: 1
Doc: 11735, Frequency: 1
Doc: 12324, Frequency: 1
Doc: 14368, Frequency: 1
Doc: 14374, Frequency: 1
Doc: 14519, Frequency: 1
Doc: 15320, Frequency: 1
Doc: 16165, Frequency: 1
Doc: 16361, Frequency: 1
```

<역색인 결과>

```
Collections.sort(list,valueComparator);
// comparator로 정렬한 결과를 새로운 해시맵에 저장
LinkedHashMap<Integer,Integer> innerMapSorted = new LinkedHashMap<>();
for(HashMap.Entry<Integer,Integer> innerEntry : list) {
```

```

        Collections.sort(L, valueComparator);
        Comparator < Integer> 词频值 倒置 = 词频值 倒置以列 方式
        LinkedHashMap<Integer, Integer> innerMapSorted = new LinkedHashMap<>();
        HashMap.Entry<Integer, Integer> innerEntry : list) {
            innerMapSorted.put(innerEntry.getKey(), innerEntry.getValue());
        }
        idDoc.put(key, innerMapSorted);
    }

    // 3. 遍历 doc 的 key 集合
    for (Li = sortedDoc.keySet();
        kw = Li) {
        LinkedHashMap<Integer, Integer> inn = sortedDoc.get(w);
        Integer> inner = inn.keySet();
        Integer docId : inner) {
            System.out.println(w+" "+ docId + " " + inn.get(docId) + " ");
        }
    }

    // 4. 写入文件
    Writer bw = null;

    new BufferedWriter(new FileWriter("sortedDoc.txt"),1024);
    String> sortedDocKeySet = keyList;

    String word : sortedDocKeySet) {
        LinkedHashMap<Integer, Integer> innerMap = sortedDoc.get(word);
        Integer> innerMapKeySet = innerMap.keySet();
        for (Integer docId : innerMapKeySet) {
            bw.write(word+" "+ docId + " " + innerMap.get(docId) + " ");
        }
    }

    bw.newLine();
    bw.flush();

    // 5. 异常处理
    try {
        IOException e) {
            printStackTrace();
        }
    }

    if (bw != null)
        bw.close();

    catch (IOException e) {
        e.printStackTrace();
    }
}

```



Elastic Search

- 텍스트, 숫자, 위치 기반 정보, 정형 및 비정형 데이터 등 모든 유형의 데이터를 위한 무료 검색 및 분석 엔진

실시간 검색 플랫폼

문서가 색인될 때부터
검색 가능해질 때까지의
대기 시간이 매우 짧음

분산적

Elastic search에 저장된
문서는 '샤드'
(여러 다른 컨테이너) 에
걸쳐 분산되며,

샤드는 복제되어 하드웨어
장애 시에 중복되는 데이터
사본을 제공함

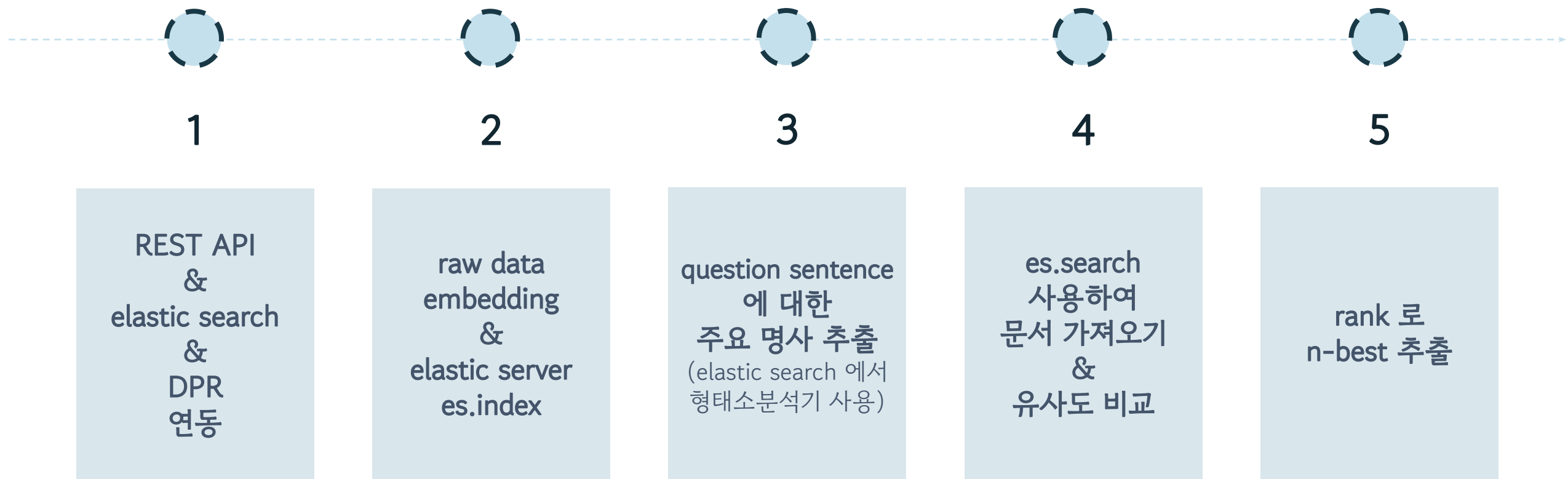
강력한 기본기능

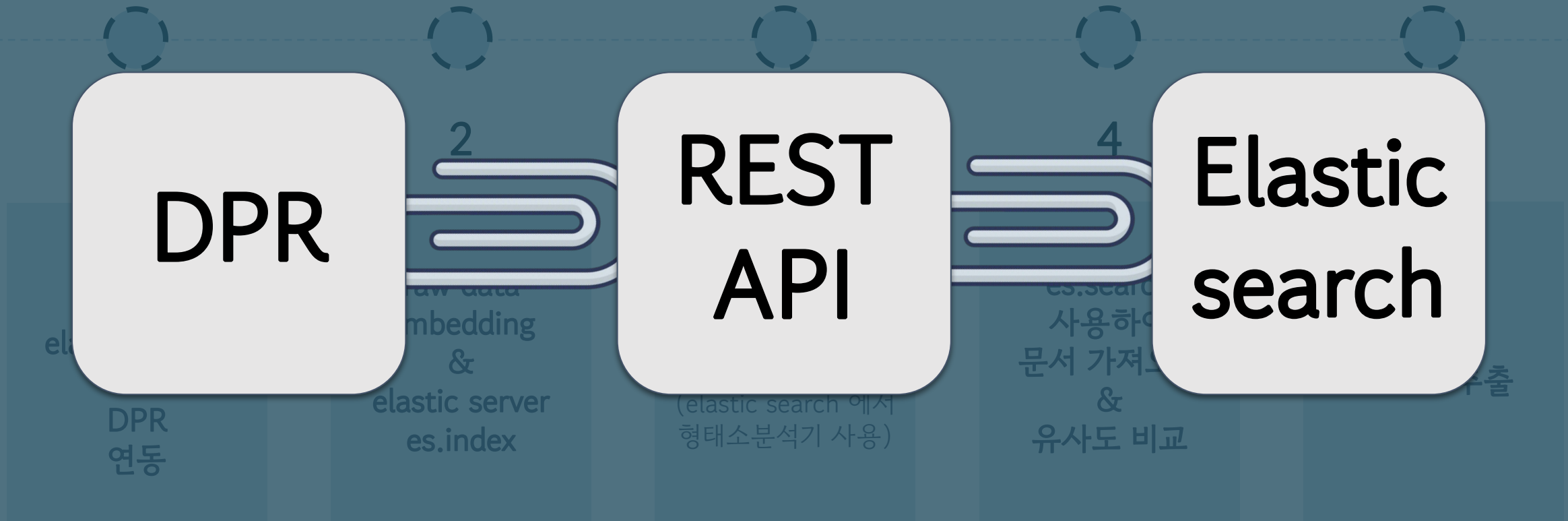
속도, 확장성, 복원력
뿐만 아니라
'데이터 롤업, 인덱스 수명
주기 관리' 등과 같이
데이터를 더욱 효율적으로
저장 및 검색할 수 있는
강력한 기본기능 탑재

데이터 처리 간소화

데이터 수집, 시각화,
보고를 간소화

Beats 와 Logstash 의
통합은 Elastic search 로
색인하기 전 데이터를 쉽게
처리할 수 있게 함





품목

브랜드

인기 Best

니트/스웨터 (19,093)

후드 집업 (4,490)

데님 팬츠 (12,544)

백팩 (5,321)

후드 티셔츠 (21,500)

메신저/크로스백 (8,395)

패션스니커즈화 (2,305)

맨투맨/스웨트셔츠 (35,510)

슈트 팬츠/슬랙스 (7,986)

카디건 (9,639)

스타디움 재킷 (1,082)

숏패딩/숏에비 아우터 (7,445)

상의 Top

아우터 Outer

바지 Pants

원피스 Onepiece

스커트 Skirt

스니커즈 Sneakers

신발 Shoes

가방 Bag

여성 가방 Women's bag

스포츠/용품 Sports/Goods

모자 Headwear

양말/레그웨어 Socks/Legwear

속옷 Underwear

선글라스/안경테 Eyewear

액세서리 Accessory

시계 Watch

주얼리 Jewelry

뷰티 Beauty

무신사 스토어 > 상품 랭킹

all

Ranking Shop

무신사 랭킹은 상품 매출, 판매 수량, 상품 조회 수, 작성 후기 수를 반영한 공식에 의해 선정됩니다. 무신사 스토어는 광고 목적으로 랭킹을 절대 임의 조작하지 않으므로 믿고 구매하셔도 됩니다.

상품

브랜드

검색어

무신사 랭킹을 알려드립니다.

자세히 보기

기간 구분

실시간

일간

주간

월간

3개월

대분류

전체

상의

스포츠/용품

디지털/테크

아우터

모자

리빙

바지

양말/레그웨어

책/음악/디켓

원피스

속옷

번려동물

스커트

선글라스/안경테

액세서리

신발

시계

가방

주얼리

여성 가방

뷰티

가격

전체

5만원 이하

5~10만원

10~20만원

20~30만원

30만원 이상

원 ~ 원

검색

NEW X

5분 전 갱신

<< < 1 2 3 4 5 6 7 8 9 10 > >>

100 페이지 중 1 페이지

전체

여성

남성

1위

2위

3위

4위

5위

6위

7위

8위

9위

10위

11위

12위

13위

14위

15위

16위

17위

18위

분크

Occam Doux Shoulder XL (오리 두 숏드 엑스라)

475,000원

쿠폰 -47,500원

MEMBERSHIP PRICE

★★★★★ 8

685

29

2/28 배송

PURPLE TWEED BLOUSON

268,000원

MEMBERSHIP PRICE

★★★★★ 328

5,997

아식스

젤 1090 V2 SMU - 폴라

99,000원

MEMBERSHIP PRICE

★★★★★ 342

1,976

디스커버리 익스페디션

라이크 에어 시프트 백팩 (BLACK)

169,000원

MEMBERSHIP PRICE

★★★★★ 13

13

잘 샌디

새벽배송 여성 스몰 카슬

1,036,000원

MEMBERSHIP PRICE

★★★★★ 169

18,398

아디다스

센테니얼 85 로우 - 화이트/블루 / IF5419

139,000원

쿠폰 -13,900원

MEMBERSHIP PRICE

★★★★★ 3

1,012

인사일런스

커브드 라인 울 플레이저

188,000원

쿠폰 -9,400원

MEMBERSHIP PRICE

★★★★★ 269

에프씨엔엠

2/24 배송 [오징어]에프씨엔엠) 헤비웨이트 후디 -

75,000원

쿠폰 -7,500원

MEMBERSHIP PRICE

★★★★★ 19

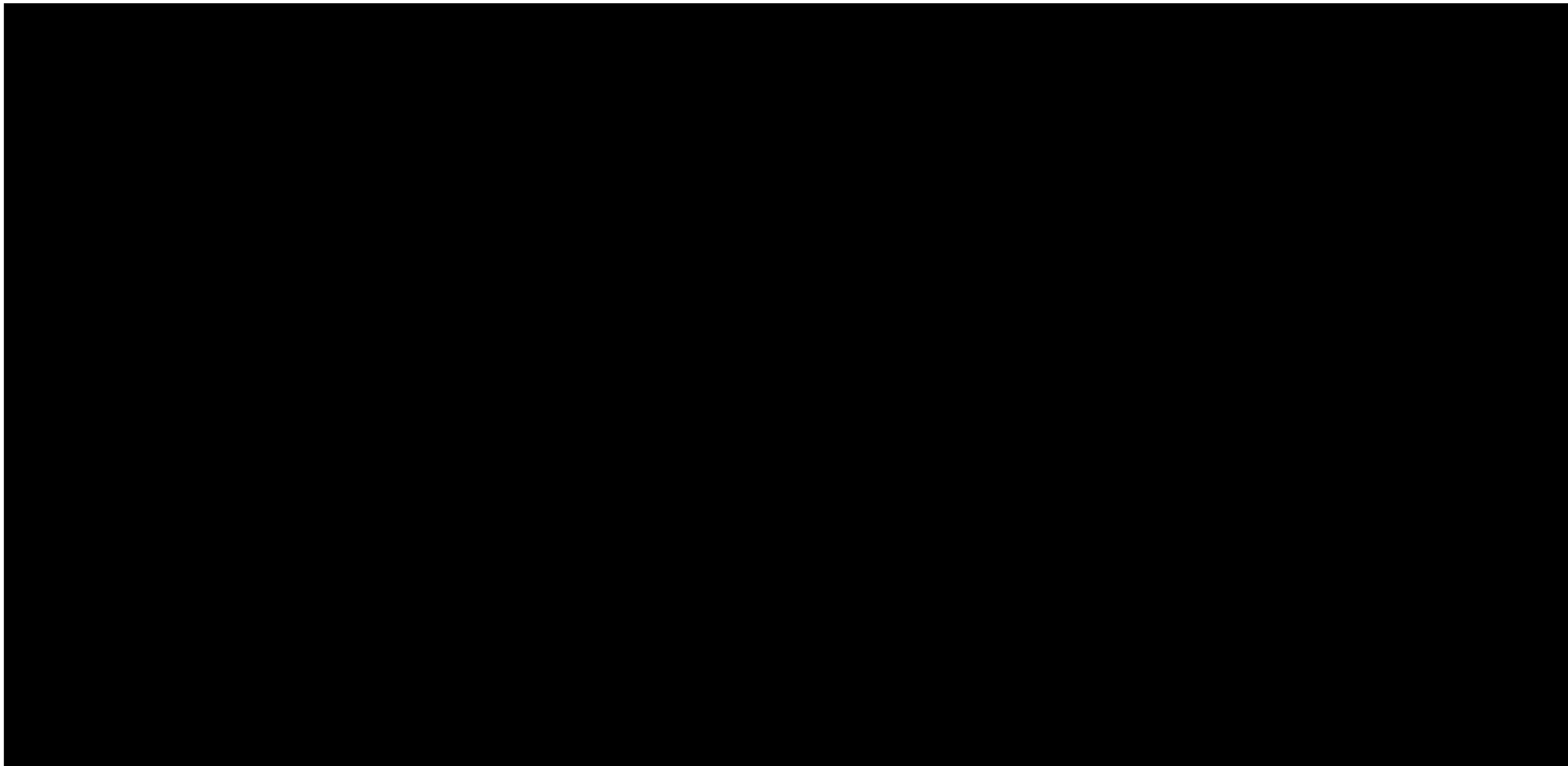
가민

인스틸트 2 솔라 텍티컬

599,000원

MEMBERSHIP PRICE

★★★★★ 19



6

기대효과 및 한계점

기대효과

- 검색 품질의 개선 : dense retrieval 모델을 이용한 품질개선
- 검색 효율성 향상 : 여러개의 문서를 병렬로 처리 가능
- 기존 시스템 개선 : 기존 시스템의 검색 품질 및 효율성 개선
- 새로운 비즈니스 모델 개발 : 검색 엔진 서비스 제공

Conclusion

한계점

- gpu 서버가 없어 모델 훈련에 많은시간이 소모됨
- 한국어 데이터셋 부족
- 인터프리터 언어인 파이썬은 컴파일 언어에 비해 느린 속도
- gpu 사용시에도 문장 임베딩 시 오랜시간 소요

개선점

- 로그인 기능을 구현 및 클라이언트 세션 구현
- 모델 하이퍼 파라미터 튜닝
- 엘라스틱 서치/플라스크 서버 보안설정
- 유사도 함수를 종합해서 사용하여 엔진 성능 개선

Q & A

Thank You

