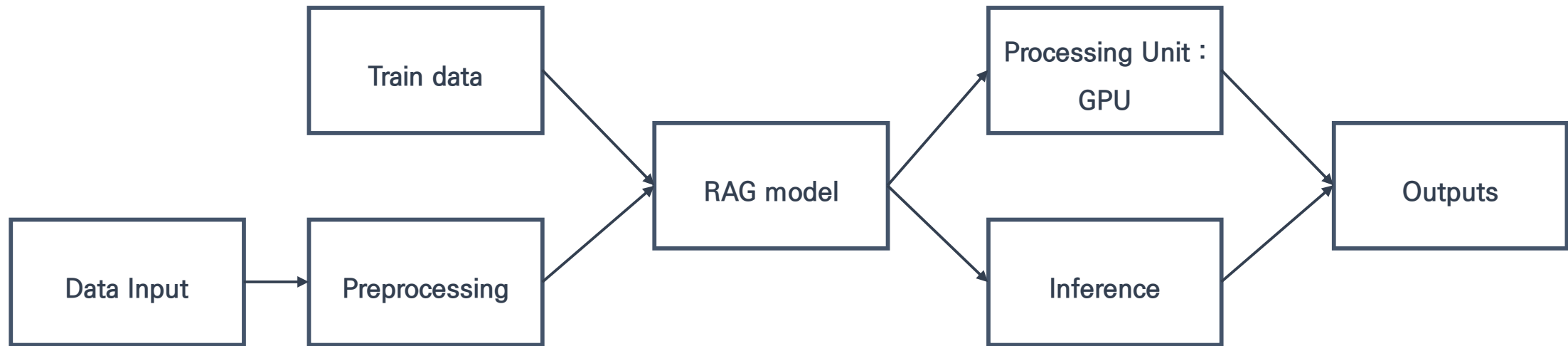
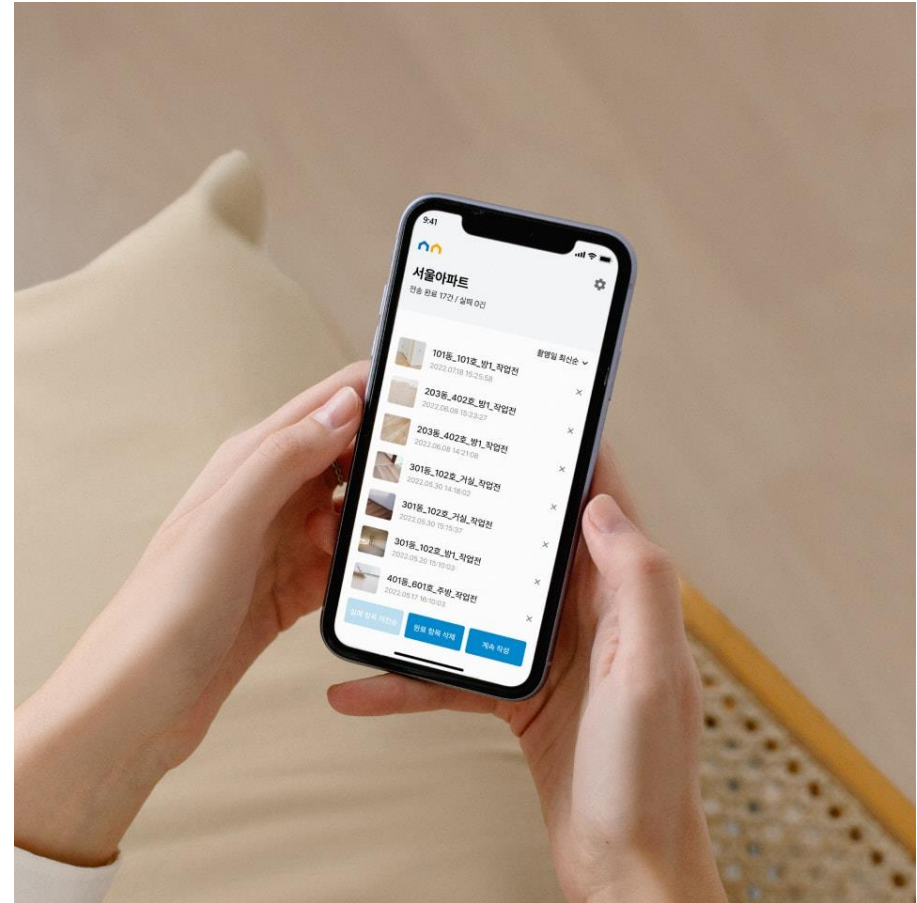

건설공사 사고 예방 및 대응책 생성

: 한솔데코 시즌3 AI 경진대회



- 건설공사 안전사고 대응 및 재발방지 대책 AI 모델 개발
- 목표
 - 건설공사 사고 데이터를 학습하여 사고 원인을 분석
 - 재발방지 대책 및 향후 조치 계획을 자동 생성
 - AI를 활용한 건설 현장 사고 예방 및 대응 혁신



출처 : 한솔데코

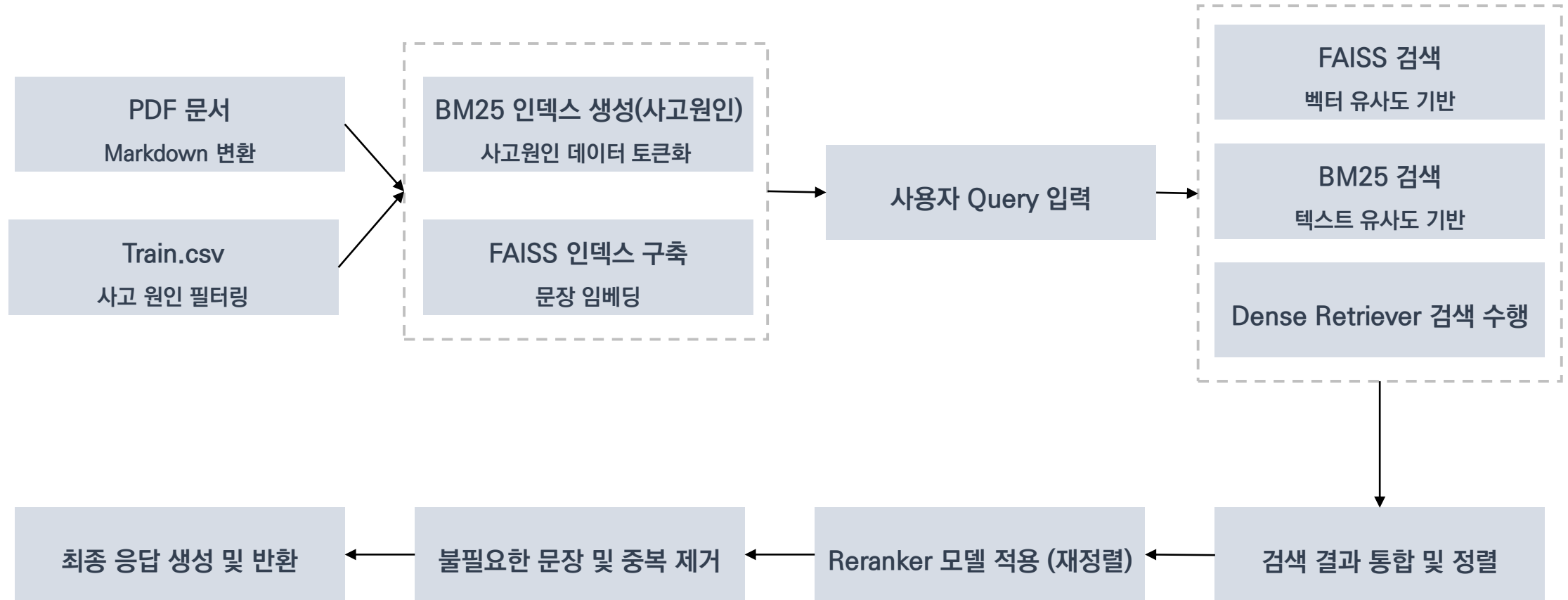
- PDF 문서에서 텍스트 추출 (PYPDF2 활용)
- Markdown 변환 후 불필요한 개행 및 특수문자 제거
- BM25kapi를 위해 텍스트를 토큰화하여 저장

```
pdf_folder = 'DAT/건설안전지침'
pdf_texts = []
for file in os.listdir(pdf_folder):
    if file.endswith(".pdf"):
        with open(os.path.join(pdf_folder, file), "rb") as f:
            reader = PyPDF2.PdfReader(f)
            text = "\n".join([page.extract_text() for page in reader.pages if page.extract_text()])
            markdown_text = markdown.markdown(text)
            pdf_texts.append(markdown_text)
```

```
[ ] bm25_corpus = df_train["사고원인"].astype(str).tolist() + pdf_texts
    tokenized_corpus = [text.split() for text in bm25_corpus]
    bm25 = BM25kapi(tokenized_corpus)
```

```
[ ] embedding_model = SentenceTransformer("jhgan/ko-sbert-sts")
    embeddings = embedding_model.encode(bm25_corpus, convert_to_numpy=True)
    embeddings = normalize(embeddings)
    faiss_index = faiss.IndexFlatL2(768)
    faiss_index.add(embeddings)
```

- 건설공사 안전사고 대응 및 재발방지 대책 AI 모델 프로세스



- BM25(Best Matching 25) : 전통적인 정보 검색 모델, 쿼리와 문서 간의 텍스트 유사도를 계산해서 관련 문서를 찾는 방법.
- FAISS (Facebook AI Similarity Search) : 고차원 벡터 검색 라이브러리, 텍스트 의미를 반영한 검색을 수행하는 방식.
- DenseRetriever : 하이브리드 검색 방식

검색 기법	방식	특징	장점	단점
BM25	키워드 기반	전통적인 정보 검색	속도 빠름, 간단함	의미 파악 어려움
FAISS	벡터 기반	의미적 유사성 검색	단어가 달라도 검색 가능	메모리 사용 큼
Dense Retriever	하이브리드	BM25 + FAISS 조합	정확도 향상	속도 저하 가능

- 다중 검색 기법 적용
 - BM25 + FAISS + SBERT 조합
 - 단순 키워드 검색이 아닌, 의미 유사성 기반 검색을 추가하여 성능 개선

```
# 1 FAISS 검색
query_embedding = embedding_model.encode([query], convert_to_numpy=True)
query_embedding = query_embedding / np.linalg.norm(query_embedding) # 정규화
_, faiss_top_idx = faiss_index.search(query_embedding, 3)

# 2 Dense Retriever 검색
dense_query_embedding = embedding_model.encode([query])
dense_scores = np.dot(train_dense_embeddings, dense_query_embedding.T).flatten()
dense_top_idx = np.argsort(dense_scores)[-3:][::-1]

# 3 BM25 검색
tokenized_query = query.split()
bm25 = BM25Okapi([text.split() for text in bm25_corpus])
bm25_scores = bm25.get_scores(tokenized_query)
bm25_top_idx = np.argsort(bm25_scores)[-3:][::-1]

# 4 검색 결과 통합
combined_top_idx = set(faiss_top_idx.flatten()).union(set(dense_top_idx)).union(set(bm25_top_idx))

# 5 검색된 문서 추출
retrieved_texts = []
for idx in combined_top_idx:
    if idx < len(df_train):
        text = df_train.iloc[idx]["재발방지대책 및 향후조치계획"]
    elif idx - len(df_train) < len(pdf_texts):
        text = pdf_texts[idx - len(df_train)]
    else:
        continue

    retrieved_texts.append(text)
```

- Reranker 적용
 - Reranker_model을 사용하여 검색된 문서의 중요도를 다시 평가
 - 문맥적으로 가장 관련성이 높은 문서를 최상위로 정렬

```
reranker_model_name = 'cross-encoder/ms-marco-MiniLM-L-6-v2'  
reranker_tokenizer = AutoTokenizer.from_pretrained(reranker_model_name)  
reranker_model = AutoModelForSequenceClassification.from_pretrained(reranker_model_name)
```

```
def rerank_documents(query, documents, reranker_model, reranker_tokenizer):  
    # 쿼리와 문서 간의 유사도를 재평가  
    inputs = []  
    for doc in documents:  
        inputs.append(reranker_tokenizer(query, doc, return_tensors="pt", padding=True, truncation=True))  
  
    with torch.no_grad():  
        reranker_scores = []  
        for input_ids in inputs:  
            outputs = reranker_model(**input_ids)  
            score = outputs.logits.item() # 모델의 예측 점수 추출  
            reranker_scores.append(score)  
  
    # 점수를 기준으로 문서 재정렬  
    ranked_docs = [doc for _, doc in sorted(zip(reranker_scores, documents), reverse=True)]  
  
    return ranked_docs
```


- ```
def clean_response(text):
 """반복 문장 제거 및 불필요한 기호, 시간 표현, 공백 정리"""
 if not isinstance(text, str) or not text.strip():
 return "" # 빈 값 처리

 # 1 "년 월 일 시 분 초 경" 같은 시간 관련 표현 삭제 (공백 고려하여 정리)
 text = re.sub(r"\b\d{4}년\b\d{1,2}월\b\d{1,2}일\b\d{1,2}시\b\d{1,2}분\b\d{1,2}초\b경", "", text)

 # 2 불필요한 개행 문자 및 특수문자 삭제 (공백 정리)
 text = re.sub(r"#+\n+#+", " ", text) # 연속 개행 -> 공백 변환
 text = re.sub(r"[\.\!\@\#\%\&*\>\<]+", "", text) # 추가 특수문자 포함

 # 3 중복 문장 제거 (완전 일치하는 문장만 제거)
 sentences = re.split(r"(\.?|!|?)#+", text) # 문장 단위 분할
 unique_sentences = list(dict.fromkeys(sentences)) # 중복 제거 후 순서 유지
 text = " ".join(unique_sentences)
 unwanted_phrases = [
 "이번에 새로 나온 신제품을 광고하고 싶어요.",
 "Ich möchte für ein neues Produkt werben.",
 "이번에 새로 나온 신제품을 수입하고 싶어요.",
 "Ich möchte dieses Mal ein neues Produkt importieren.",
 "Ich möchte dieses Mal ein neues Produkt import",
]

 for phrase in unwanted_phrases:
 text = text.replace(phrase, "") # 불필요한 문구 제거

 # 4 양쪽 공백 정리 후 반환
 return text.strip()
```




### 개발환경

- Python 3.11.11
- Numpy 2.0.2
- Pandas 2.2.2
- Torch 2.6.0+cu124



### GPU 사양

- L4 -22.5GB



### 추론 최적화

- “beomi/Llama-3-Open-Ko-8B”  
기본모델로 설정
- 최대 토큰 길이 설정
- Float16 데이터타입
- autocast() 활용
- 예외 처리 및 기본 응답 설정

- 현재 코드의 서빙 가능성
  - FAISS 및 BM25를 활용한 검색 기반 모델
  - PyTorch 기반 모델을 사용하여 최종 응답 생성
  - Reranker를 적용해 검색된 문서를 정제
- 개선 및 제안 사항
  - FastAPI/Flask를 적용하여 REST API로 서비스화
    - 현재는 로컬에서 실행되지만, API 형태로 배포하면 실시간 질의 응답 가능
  - FAISS & BM25 인덱스를 메모리에 로드하여 검색 속도 최적화
    - 현재 매번 검색을 수행하지만, API 배포 시 인덱스를 메모리에 로드하면 처리 속도 향상
  - Pytorch Inference 최적화
    - 현재 autocast()를 사용하지만, 모델 배포 시 추가적인 최적화(ONNX 변환, FP16 변환 등) 고려 가능

---

감사합니다.

---