250203 논문 피드백

1. 데이터셋 및 초기 설정

• 생성 및 검색 데이터셋 준비 완료:
– 실제 서비스에서는 방대한 데이터셋이 필요하지만,
논문에서는 형사 분야 하나에 집중하여 충분한 양의 데이터를 확보

.

```
테스트 케이스 생성 시작...
2025-02-01 00:40:35,547 - data_processor - INFO - 기존 Sparse Encoder 로드 완료: KiwiBM25_sparse_encoder.pkl
[load_sparse_encoder]
Loaded Sparse Encoder from: KiwiBM25_sparse_encoder.pkl
2025-02-01 00:40:36,824 - data_processor - INFO - FAISS 인덱스 로드 완료 (cached_vectors/retrievers/faiss)
카테고리별 분포 계산 중...
카테고리 분석: 100%|██████████| 53209/53209 [00:00<00:00, 924264.36it/s]

카테고리별 할당된 질문 수:
형사: 30개
전체 진행률: 100%|██████████| 30/30 [04:59<00:00,  9.97s/it]

테스트 케이스 생성 완료!
총 소요 시간: 301.30초
생성된 총 테스트 케이스: 30개

JSON 파일 저장 중...
```

– ground truth는 실제 법률 자문을 통해 보완할 필요가 있음
(현재는 향후 Scopus 수준의 논문 개발을 위해서는 추가 보완 필요)

```python
def generate_structured_query(ground_truth):
    """GPT를 사용하여 더 복잡하고 분석적인 질문 생성"""
    prompt = f"""
    다음 법률 텍스트를 바탕으로 법률적 분석이 필요한 질문을 생성해주세요:
    {ground_truth}

    다음 유형의 질문들 중 하나를 선택하여 생성하세요:
    1. 법리 분석: "어떤 법적 원칙이 적용되며, 그 근거는 무엇인가?"
    2. 요건 분석: "어떤 요건들이 충족되어야 하며, 각 요건의 의미는?"
    3. 비교 분석: "유사한 다른 법적 상황과 어떤 차이가 있는가?"
    4. 영향 분석: "이러한 판단이 향후 유사 사례에 미치는 영향은?"

    질문은 단순한 yes/no가 아닌 설명이 필요한 형태여야 합니다.
    """
    llm = ChatOpenAI(model="gpt-4o-mini-2024-07-18")
    response = llm.invoke(prompt)
    return response.content.strip()
```

• Sparse Encoder 관련 개선:

– 기존에 학습된 Sparse Encoder가 존재하면(예: ../KiwiBM25_sparse_encoder.pkl) 불필요한 학습 과정을 생략하고, 저장된 모델을 바로 로드하는 방식으로 개선하였습니다.

– 로드(load)와 학습(fit) 과정에 대해 “[LOAD]”, “[FIT]” 태그가 포함된 상세 로그를 남겨, 각 실험 케이스별로 어떤 방식이 사용되었는지 명확히 기록되고 있습니다.

```python
def fit_and_save_sparse_encoder(self, query: str, documents: List[Document]) -> str:
    """Sparse Encoder를 실시간으로 학습하고 저장합니다."""
    try:
        doc_contents = [doc.page_content for doc in documents]
        stop_words = []  # 필요 시 추가
        sparse_encoder = create_sparse_encoder(stop_words, mode="kiwi")
        save_path = self.sparse_encoders_dir / f"sparse_encoder_{query[:10]}_{len(documents)}.pkl"
        logger.info(f"[FIT] Sparse Encoder 학습 시작: query='{query[:10]}', 문서 수={len(documents)}. 저장 경로: {save_path}")
        fit_sparse_encoder(sparse_encoder, doc_contents, save_path=save_path)
        logger.info(f"[FIT] 새로운 Sparse Encoder 저장 완료: {save_path}")
        self.sparse_encoder = sparse_encoder
        return str(save_path)
    except Exception as e:
        logger.error(f"Sparse Encoder 학습 및 저장 중 오류 발생: {e}")
        raise
```

2. Context Relevance Check (검색 평가)

• 점수 측정:

– Query와 문서의 Sparse 벡터(키워드 기반) 및 Dense 벡터(의미 기반)를 각각 코사인 유사도로 산출하고, 이를 0~1 사이의 값으로 정규화

```python
def evaluate_sparse_method(self, query: str, documents: List[Document]) -> float:
    """실시간으로 Sparse Encoder를 학습한 뒤 쿼리-문서 유사도를 계산합니다."""
    try:
        if not documents:
            logger.warning("문서 리스트가 비어 있음")
            return 0.0

        vectors = self._get_or_create_sparse_vectors(query, documents)
        query_vec = vectors['query_vec']
        doc_vecs = vectors['doc_vecs']
        similarities = []
        # 각 문서와의 코사인 유사도 계산 (Sparse)
        for doc_vec in doc_vecs if isinstance(doc_vecs, list) else [doc_vecs]:
            common_indices = set(query_vec["indices"]) & set(doc_vec["indices"])
            if not common_indices:
                similarities.append(0.0)
                continue

            max_index = max(max(query_vec["indices"]), max(doc_vec["indices"]))
            q_vec = np.zeros(max_index + 1)
            d_vec = np.zeros(max_index + 1)
            for idx, val in zip(query_vec["indices"], query_vec["values"]):
                q_vec[idx] = val
            for idx, val in zip(doc_vec["indices"], doc_vec["values"]):
                d_vec[idx] = val

            norm_q = np.linalg.norm(q_vec)
            norm_d = np.linalg.norm(d_vec)
            if norm_q == 0 or norm_d == 0:
                similarities.append(0.0)
            else:
                similarity = np.dot(q_vec, d_vec) / (norm_q * norm_d)
                normalized_similarity = (similarity + 1) / 2  # [0,1] 범위로 정규화
                similarities.append(normalized_similarity)

        avg_similarity = np.mean(similarities) if similarities else 0.0
        logger.info(f"[Sparse 평가] Query: '{query[:15]}...', 문서 수: {len(documents)}, 평균 유사도: {avg_similarity:.4f}")
        return avg_similarity

    except Exception as e:
        logger.error(f"Sparse 평가 중 오류: {e}")
        return 0.0
```

```python
def evaluate_dense_method(self, query: str, documents: List[Document]) -> float:
    """Upstage solar-embedding-1-large-query 기반 점수 계산"""
    try:
        dense_val = self.search_engine._calculate_semantic_similarity(documents, query)
        normalized_score = max(min(dense_val, 1.0), 0.0)
        logger.info(f"[Dense 평가] Query: '{query[:15]}...', 문서 수: {len(documents)}, 점수: {normalized_score:.4f}")
        return normalized_score
    except Exception as e:
        logger.error(f"Dense 평가 중 오류: {str(e)}")
        return 0.0
```

– 두 점수를 가중치 (예: 0.3, 0.4, 0.5, 0.6, 0.7)를 적용하여 Hybrid Score로 결합하는 방법을 사용하였습니다.

• 정규화 전처리 및 평가 개선:

– RAGAS 평가 과정에서 ground truth와 문서(컨텍스트) 간의 형식 차이(대소문자, 불필요한 공백 등)로 인해 평가 점수가 0으로 산출되는 문제를 해결하기 위해, 두 텍스트를 소문자 변환 및 trim(공백 제거) 등의 정규화를 적용

-> 현재까지 0으로 나오는 문제 발생

– 정규화된 텍스트의 일부(예: 앞 100자)를 디버그 로그로 출력하여, 평가 데이터의 신뢰성을 확인

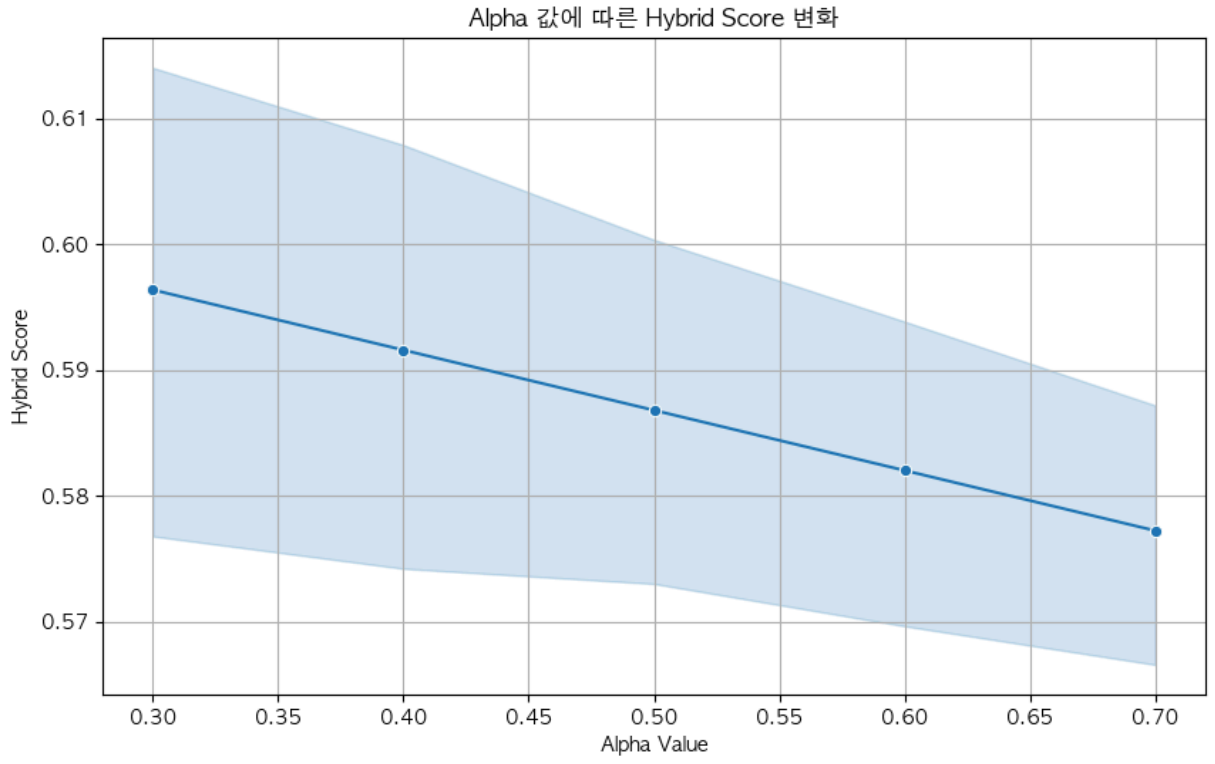
• RAGAS 평가:

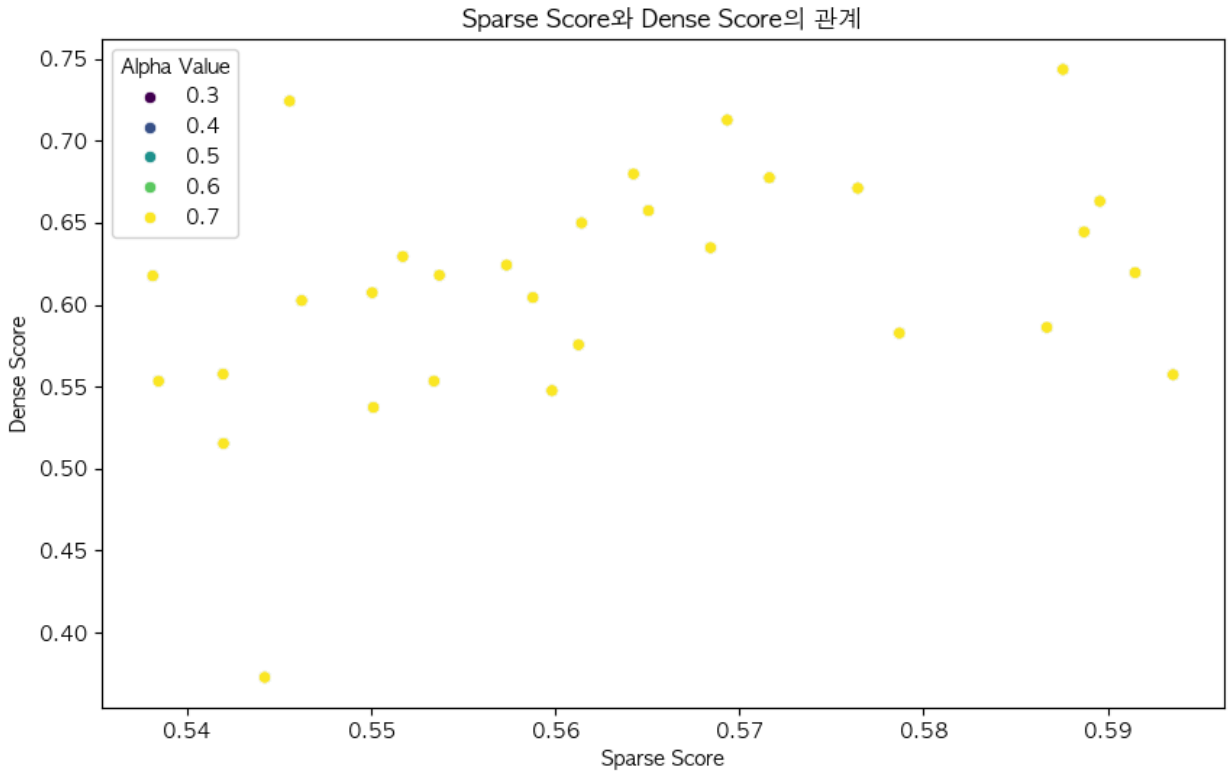
– 정규화된 데이터를 기반으로 EvaluationDataset과 SingleTurnSample을 생성하고, RAGAS 평가 지표(문맥 Precision, Recall) 및 F1 Score도 계산를 산출

평가 결과를 정량적으로 비교 분석

• 상관관계 분석 및 시각화:

– Sparse, Dense, Hybrid 점수와 RAGAS 평가 지표 간의 상관관계를 계산하고, 이를 히트맵과 2차원 그래프(예: Sparse-Dense 관계 시각화)를 통해 분석하였습니다.



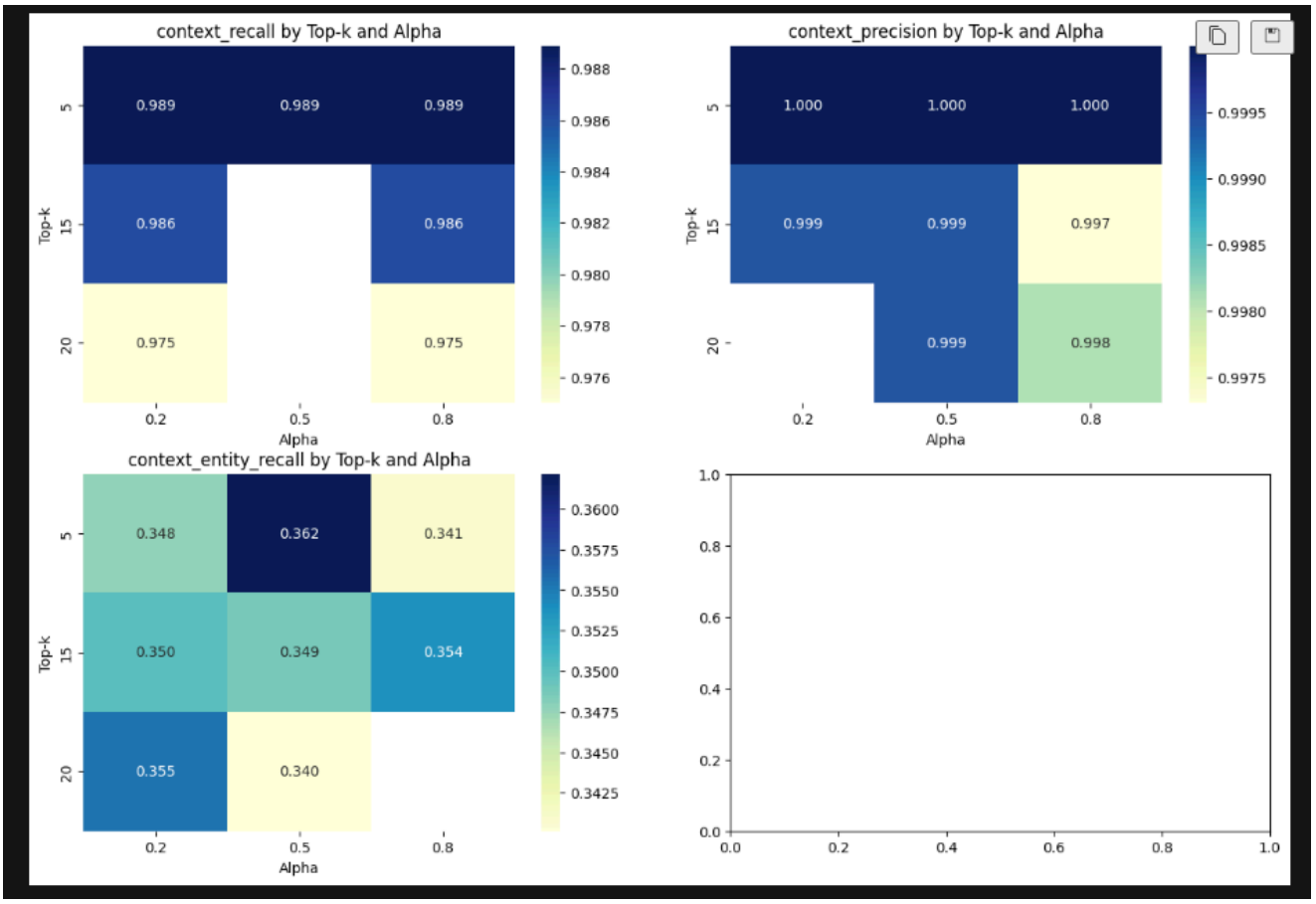


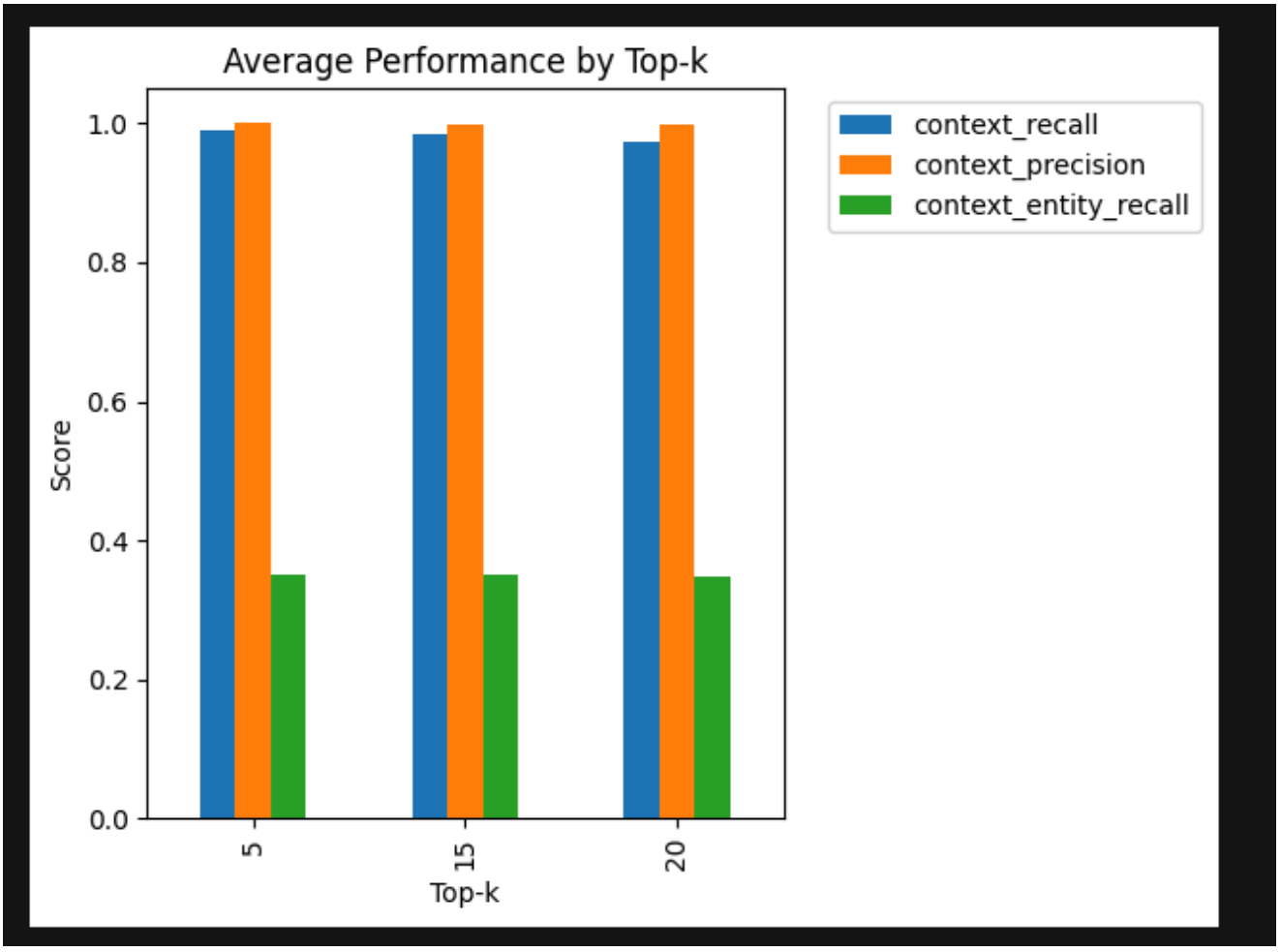
– 이를 통해 Hybrid 방식이 단일 방식 대비 검색 정확도 및 관련 문서 선정에 있어 상호보완적 효과를 지니는지 확인

3. Top-k (n_docs) 파라미터 시험

• 실험 설계:
– Top-k 문서 수(n_docs)를 5, 15, 20으로 변경
각 설정에 따른 Hybrid Score 및 RAGAS 지표, 사용자 평가 점수를 측정
– K-Means나 K-selection 방법을 참고하여, 최적의 n_docs 값을 도출하는 실험을 진행 중
특히 n_docs가 15일 때 가장 우수한 결과를 보이는 경향을 확인

=> 앞선 실험 이후에 구체화할 예정





4. Response Verifier 및 재작성(Re-RAG) 실험

• **재작성 프로세스:**
– GPT-4o 모델을 사용하여 생성된 답변에 대해, UpstageGroundCheck 및 임베딩 유사도 기반의 RAGAS 평가를 수행합니다.
– 만약 평가 점수가 사전 정의된 임계값(예: 0.7)에 미달할 경우, 질문 또는 답변을 재작성(Re-RAG)하여 최대 3회까지 반복하는 루프를 적용하여 답변 품질을 개선하는 실험을 진행 중

• **현재 진행도:**
– Response Verifier 및 Re-RAG 실험은 코드 완료도가 약 30% 수준으로,
기존 코드를 재활용하여 진행할 계획이며, 후속 실험을 통해 보완.

5. Boosting 방법 및 향후 계획
• **Boosting RAG:**
– Sparse와 Dense 검색의 점수를 선형 결합(가중치 적용)하여 Hybrid Score를 산출하는 방법을 적용
– 추후 선형 SVM 등 다른 결합 기법을 도입하여, 두 점수를 최적의 weight로 결합하는 방법도 고려

• **논문 작성 방향:**
– 현재 진행 중인 Retrieval(검색) 부분과 Generation(생성) 부분을
각각 독립된 논문으로 작성할 수 있는 가능성을 열어둔 상태

종합 및 향후 연구 방향

• **현재 성과:**
– 법률 도메인에서 Sparse와 Dense 검색 방식을 결합한 Hybrid 검색 평가 시스템을 구현
저장된 Sparse Encoder의 효율적 활용, 정규화 전처리를 통한 RAGAS 평가 개선,
그리고 상관관계 분석 및 시각화를 통해 하이브리드 접근법의 우수성을 정량적으로 입증

• **남은 과제:**
– Dense와 Sparse 점수를 합산하는 최적의 가중치 조합 및 Linear-SVM 등 추가 결합 기법 도입.
– Top-k 파라미터 최적화 및 Re-RAG(재작성) 단계의 코드 완성도 향상.
– ground truth의 법률 자문 보완과 데이터셋 확장을 통한 연구 범위 확대.

• **향후 계획(논문 결론 부분):**
– 사용자 평가 및 전문가 피드백을 반영한 동적 튜닝 기법 도입.
– 다양한 LLM 적용과 더 대규모의 법률 전문 임베딩 모델 결합을 통해 시스템 성능을 추가적으로 개선