

## 1. 프로젝트 개요 및 문제 접근법

### 1) 생성형 AI 모델 설계 전략 및 접근법

본 대회 평가 지표는 객관식 정확도(Accuracy)와 주관식 의미 유사도(Cosine Similarity + Keyword Recall)를 혼합하여 산출됩니다. 따라서 모델은 단순한 사실적 정답 추출 능력뿐만 아니라, 자연어 서술의 의미적 유사도와 핵심 키워드 재현율을 동시에 만족시킬 수 있어야 합니다. 이를 위해 다음과 같은 전략으로 모델을 설계하였습니다.

#### (1) 모델 선정 및 이유

본 프로젝트에서는 KT의 한국형 대규모 언어모델인 믿:음 2.0 Base를 채택하였습니다.

- 믿:음 2.0은 한국어 뉘앙스와 문화적 맥락을 정교하게 반영하여 학습된 모델로, 한국어 질의 응답 및 법률·금융 텍스트 처리에 강점을 지닙니다.
- 공개된 범용 LLM 대비 한국어 표현력과 문맥 이해력이 우수하여, 주관식 문제에서 Cosine Similarity 점수를 높이는 데 유리합니다.
- 자체 설계된 한국어 토큰라이저를 통해 키워드 단위 인식이 향상되어, Keyword Recall 지표에서도 안정적인 성능을 확보할 수 있습니다.
- Base 모델(11.5B 파라미터)은 객관식 정답 매칭 정확도 확보에 충분한 용량을 가지면서도, 4bit 양자화를 통해 추론 효율성을 확보하였습니다.

#### (2) 아키텍처 및 기술적 접근

- 한국어 특화 학습: 한국의 법률·금융 관련 문헌, 공공 데이터, 합성 QA 데이터를 포함하여 사전학습·정제.
- 온디바이스 최적화: 양자화(4bit)와 프루닝(pruning) 기법으로 GPU 메모리 사용량을 줄여 실험/추론 효율성을 확보.
- 검색 증강(RAG): OpenAI 임베딩(text-embedding-3-small) 기반 인덱스와 FAISS 검색기를 활용하여, 질의와 관련된 법령·시행령·규칙 조문을 맥락으로 주입 → 의미적 일치도와 키워드 재현율을 동시에 향상.
- 프롬프트 설계: 객관식 문항에서는 선택지 번호만 정확히 출력하도록 지시하여 Accuracy 점수를 극대화, 주관식 문항에서는 핵심 키워드를 포함하는 간결한 서술을 유도하여 Keyword Recall을 보완.

#### (3) 기대 효과

- 객관식 문항에서는 정답률 제고(Accuracy ↑)
- 주관식 문항에서는 자연스러운 한국어 서술 + 키워드 충실도 확보를 통해 의미 유사도·재현율 점수 향상
- 종합적으로 FSKU 지표 최적화

## 2) 기술적 차별성

본 팀의 접근법은 단순히 사전학습된 생성형 AI 모델을 사용하는 수준을 넘어, FSKU 지표 최적화를 목표로 다양한 기술적 차별화를 적용하였습니다.

### (1) 한국어 특화 대형 언어모델 채택

- 밑:음 2.0 Base는 한국어 토큰라이저와 한국 문화·지식이 반영된 데이터셋으로 학습되어, **한국어 문장 이해 및 생성 능력에서 우위**를 확보하였습니다.
- 일반 공개모델 대비, 법률·금융과 같은 **전문화된 문맥 이해력**이 높아 의미 유사도(Cosine Similarity) 점수를 향상시킵니다.

## (2) RAG 기반 검색 증강

- 다국어 임베딩 모델 BAAI/bge-m3과 FAISS 인덱스를 활용하여 **법령, 시행령, 시행규칙, 감독 규정 등 관련 조문을 동적으로 검색** 후 컨텍스트로 주입.
- 이를 통해 단순한 모델 추론이 아닌 **법령 근거 기반 응답**을 생성, 주관식 답변에서 **정답 키워드 재현율(Keyword Recall)**을 크게 개선하였습니다.

## (3) 프롬프트 설계 전략

- 객관식: 선택지 번호만 출력하도록 강력히 유도하여 불필요한 텍스트 생성을 방지 → Accuracy 점수 최적화.
- 주관식: “핵심 키워드 포함 + 간결한 설명”을 유도하는 프롬프트를 설계하여, 의미 유사도와 키워드 재현율을 동시에 확보.
- 질문 유형(객관식/주관식)을 자동 감지하는 로직을 추가하여, **유형 맞춤형 추론 경로**를 적용.

## (4) 최적화 및 모델 응답 향상 기법

- **4bit 양자화(BitsAndBytes)**: GPU 메모리 사용량을 절감하면서도 모델 성능 손실 최소화 → 대회 환경에서 안정적인 추론 가능.
- **Temperature=0, Top-p=1.0 설정**: 확률적 다양성보다 **정확도 및 일관성**을 중시한 생성 설정으로, 객관식 Accuracy 점수 향상.
- **후처리 함수(extract\_answer\_only)**: 출력에서 불필요한 텍스트를 제거하고 정답 숫자나 핵심 답변만 추출 → 평가 지표와의 직접적인 정합성 강화.

## (5) 종합 효과

- 객관식 문제에서는 **정확한 선택지 번호 응답**으로 높은 Accuracy 확보.
- 주관식 문제에서는 **법령 기반 검색 + 키워드 중심 응답**으로 Cosine Similarity 및 Keyword Recall 점수 동시 향상.
- 결과적으로 타 모델 대비 FSKU 종합 점수 최적화에 유리한 구조를 구현하였습니다.

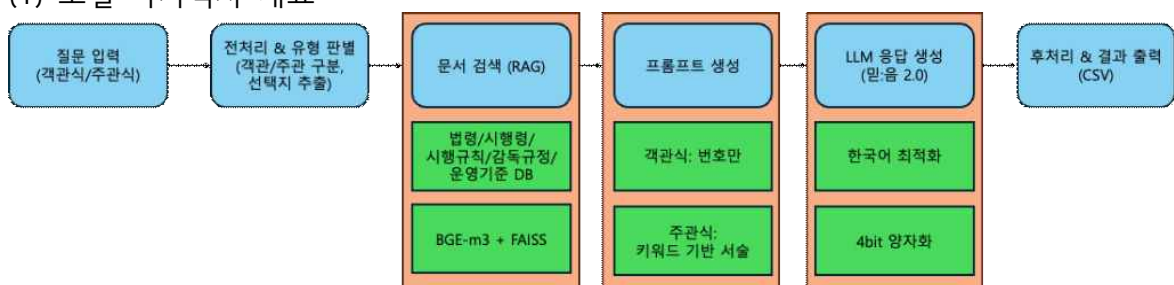
## 3) 개발 환경 및 도구 구성

구분	도구/라이브러리	버전	제조사(출처)	용도
1	Python	3.10	Python Software Foundation	전체 구현 환경
2	PyTorch (torch)	2.8.0	PyTorch Foundation	생성형 AI 모델 학습/추론
3	Transformers	4.56.0	Hugging Face	사전학습 LLM 로딩 및 파인튜닝
4	Sentence-Transformers / HuggingFace-BGE	5.1.0	Hugging Face / BAAI	문서·질의 임베딩 변환
5	LangChain	0.3.27	LangChain	프롬프트 체인, 벡터DB 검색 파이프라인
6	FAISS (faiss-cpu)	1.12.0	Meta (Facebook AI)	임베딩 기반 유사도 검색
7	Pandas	2.3.2	PyData	CSV/테이블 데이터 처리
8	Lxml	6.0.1	Lxml Project	HTML 파싱, 규제 문서 전처리
9	PyPDF	6.0.0	PyPDF Project	규제 문서(PDF) 조문 단위 분리
10	Scikit-learn	1.7.1	Scikit-learn Community	벡터화·유사도 계산 보조
11	Tqdm	4.67.1	Open Source	진행상황 모니터링 (Progress bar)

## 2. 상세 개발 방법론

### 1) 생성형 AI 모델 구조 상세

#### (1) 모델 아키텍처 개요



본 시스템은 LLM 기반 파인튜닝 + RAG(Retrieval-Augmented Generation) 구조로 설계되었습니다. 입력된 질문은 객관식/주관식 여부를 판별한 뒤, 법령/시행령/시행규칙/감독규정/운영기준 DB를 대상으로 검색하여 관련 문맥을 확보하고, 이를 프롬프트에 반영하여 LLM 응답을 생성합니다. 마지막으로, 평가 지표(FSKU)에 맞추어 후처리 과정을 거쳐 CSV 형태의 결과를 출력합니다.

#### (2) 세부 처리 흐름

##### 1. 질문 입력

- 데이터로부터 객관식 또는 주관식 질문을 입력받습니다.

## 2. 전처리 & 유형 판별

- 정규식 기반 파서를 통해 질문을 전처리하고, 객관식/주관식 여부를 판별합니다.
- 객관식일 경우 선택지를 자동 추출하여 모델 입력에 활용합니다.

## 3. 문서 검색 (RAG)

- BAAI/bge-m3 임베딩과 FAISS 벡터 검색기를 이용해 관련 법령·시행령·시행규칙·감독규정·운영기준 DB에서 연관 문서를 검색합니다.
- 이를 통해 모델이 단순 언어 패턴이 아닌, 실제 법령 문맥을 근거로 응답할 수 있도록 합니다.

## 4. 프롬프트 생성

- 객관식: 선택지 번호만 출력하도록 설계하여 Accuracy를 극대화.
- 주관식: 키워드 기반 간결 서술을 유도하여 Cosine Similarity 및 Keyword Recall 점수를 높임.

## 5. LLM 응답 생성 (민:음 2.0)

- 한국어에 최적화된 KT의 민:음 2.0 모델을 사용하여, 한국어 특유의 뉘앙스와 표현을 정확하게 반영합니다.
- 4bit 양자화를 적용해 GPU 메모리 효율성을 확보하면서도 고성능 응답 생성을 유지합니다.

## 6. 후처리 & 결과 출력

- 생성된 응답을 평가 지표에 맞게 후처리합니다.
- 객관식은 선택지 번호만, 주관식은 불필요한 문구를 제거하고 키워드 중심의 결과를 CSV 파일로 저장합니다.

## (3) 핵심 모듈 및 알고리즘 역할

- 전처리 모듈: 질문 유형 및 선택지 분리
- RAG 검색 모듈: BAAI/bge-m3 + FAISS 기반 의미 검색 → 주관식 점수(유사도, 키워드) 향상.
- 프롬프트 엔지니어링: 출력 형식을 제어 → 평가 산식에 최적화된 답변 생성.
- LLM 응답 모듈: 한국어 특화 모델(민:음 2.0) → 자연스러운 한국어 응답 보장.
- 후처리 모듈: 평가 지표 출력 포맷(CSV) → 제출용 데이터셋.

## 2) 데이터 전처리 및 입력 프롬프팅 전략

본 연구에서는 생성형 AI 모델이 평가 지표(FSKU: 객관식 Accuracy, 주관식 Cosine Similarity 및 Keyword Recall)에 최적화된 답변을 생성할 수 있도록 데이터 전처리와 입력 프롬프팅 전략을 세밀하게 설계하였다. 주요 내용은 다음과 같습니다.

### (1) 데이터 전처리 단계

- **질문 정규화**
- 입력 텍스트의 불필요한 공백 및 특수문자를 제거하여 일관된 형식을 유지하였습니다.
- 법령 조문 번호(예: 제3조, 제3조의2)를 정규식으로 안정적으로 추출할 수 있도록 처리하였습니다.
- **문항 유형 판별**
- 정규식을 활용하여 1., 2.와 같은 선택지 패턴을 탐지하였습니다.
- 두 개 이상의 선택지가 존재하는 경우 객관식으로 분류하고, 그렇지 않은 경우 주관식으로 처리하였습니다.
- **선택지 추출(객관식 전용)**
- 질문 본문과 선택지를 분리하고, 선택지 번호와 내용을 각각 파싱하여 모델 입력에 반영하였습니다.
- **문서 검색(RAG) 기반 컨텍스트 구성**
- 질문을 BAAI/bge-m3 임베딩으로 변환한 후 FAISS 인덱스를 이용하여 법령·시행령·시행규칙·감독규정·운영기준에서 관련 문서를 검색하였습니다.
- 검색된 문서 스니펫은 프롬프트 상단에 추가하여 답변의 정확성과 키워드 재현율을 높였습니다.

다.

- **토큰 및 길이 관리**
- 모델 입력의 길이를 제어하기 위해 문자 단위(char budget)를 설정하고, 길이가 초과될 경우 문서 스니펫을 요약하여 반영하였습니다.
- 추론 시 temperature=0, top\_p=1.0 등 결정론적 파라미터를 적용하여 재현성을 확보하였습니다.

## (2) 입력 프롬프팅 전략

### 1. 객관식 문항

- 프롬프트 템플릿에서 “정답 선택지 번호 한 개만 출력”을 명시하여 불필요한 설명을 방지하였습니다.
- 예시:  
당신은 금융보안 전문가입니다.  
아래 질문에 대해 가장 적절한 정답 선택지 번호 한 개만 출력하세요.  
[질문] ...  
[선택지]  
1 ...  
2 ...  
답변:  
• 후처리 과정에서 응답 중 최초의 숫자만 추출하여 평가 지표의 Accuracy 손실을 최소화하였습니다.
- 검색된 법령 스니펫을 프롬프트에 [참고할 문서] 로 제공하여 정답률을 향상시켰습니다.

### 2. 주관식 문항

- 프롬프트 템플릿에서 “정확하고 간략한 설명”과 “핵심 키워드 포함”을 요구하여 모델이 핵심 용어를 유지하면서 짧은 문장을 생성하도록 유도하였습니다.
- 예시:  
당신은 금융보안 전문가입니다.  
아래 주관식 질문에 대해 정확하고 간략한 설명을 작성하세요.  
질문: ...  
답변:  
• 검색된 법령 스니펫을 프롬프트에 함께 제공하여, 모델이 실제 문맥에 기반한 답변을 하도록 하였으며, 이를 통해 Cosine Similarity 및 Keyword Recall을 향상시켰습니다.

## (3) 객관식/주관식 처리 전략 차이

- 객관식: 선택지 분리 및 번호 출력 강제 → 정답률(Accuracy) 극대화
- 주관식: 핵심 키워드 기반 간략 문장 생성 → 의미 유사도 및 키워드 재현율 향상
- 공통점: 두 유형 모두 검색 증강(RAG) 기반 컨텍스트를 활용하여 법령 조문에 근거한 신뢰성 있는 답변을 제공

## 3) 외부 지식 활용(RAG) 및 임베딩 전략

본 시스템은 단순한 언어모델 추론만으로는 정확성과 신뢰성이 부족할 수 있다는 점을 보완하기 위해, **RAG(Retrieval-Augmented Generation) 기반 구조**를 적용하였습니다. 이를 통해 모델이 학습 데이터에 없는 최신 법령이나 세부 조문 내용을 활용할 수 있도록 하였습니다.

### (1) 외부 문서 및 금융보안 지식베이스 구축

- **데이터 수집 범위:** 주요 금융보안 관련 법령 및 규정(「개인정보보호법」, 「신용정보법」, 「자본시장법」, 「전자금융거래법」, 「전자서명법」, 「정보통신망법」 등)을 포함하였습니다.
- **문서 구조화:** 법령-시행령-시행규칙 연계(Family Mapping): 조문 단위로 매핑하여 본문과 연결된 하위 규정을 함께 관리.
- **PDF 파싱 및 분할:** PyPDFLoader를 활용하여 장·조문 단위로 분리하고, 부칙까지 별도로 처리하였습니다.
- **CSV/JSON 아티팩트 저장:** “본문 조문 + 연계 조문” 형태로 가공하여 RAG 검색에 최적화된

입력 형태를 확보하였습니다.

## (2) 임베딩 모델 및 벡터 DB 구성

- **임베딩 모델**: BAAI/bge-m3를 사용하여 문서와 질의를 동일한 벡터 공간에 매핑하였습니다.
- 한국어 문장 표현에 강점이 있으며, `normalize_embeddings=True` 설정으로 코사인 유사도 기반 검색을 안정적으로 수행하였습니다.
- **벡터 데이터베이스**: FAISS 라이브러리를 활용하여 구축하였습니다.
- **Family 인덱스**: 법령 본문과 연결된 시행령·시행규칙 조문을 하나의 인덱스로 통합하였습니다.
- **개별 인덱스**: 시행령(ED), 시행규칙(ER), 감독규정(REG), 운영기준(BASE)을 각각 별도 인덱스로 관리하였습니다.
- **Manifest 관리**: 각 인덱스의 경로와 사용한 임베딩 모델을 `manifest.json`에 기록하여 재현성을 보장하였습니다.

## (3) 문서 검색 및 지식 추출 과정

- **질의 전처리**: 입력 질문에서 법령명, 조문번호, 문항 유형(객관식/주관식)을 판별하여 검색 스크립트를 좁혔습니다.
- **벡터 검색 및 재랭킹**:
  - 질의를 임베딩하여 Family 인덱스 및 관련 인덱스에서 상위 k개 문서를 검색합니다.
  - 검색 결과에 대해 법령 패밀리 일치, 문서 유형 일치 여부에 따라 가중치를 부여하여 재랭킹을 진행합니다.
- **컨텍스트 구성**: 최종적으로 선택된 상위 문서를 합쳐 “[참고할 문서]” 블록을 생성하고, 이를 프롬프트 상단에 삽입합니다.
- **지식 추출의 효과**: 모델이 단순 통계적 패턴이 아닌 실제 법령 조문을 근거로 응답하게 함으로써, 객관식 문항에서는 정답률(Accuracy)을, 주관식 문항에서는 의미 유사도(Cosine Similarity) 및 키워드 재현율(Keyword Recall)을 동시에 개선하였습니다.

## 4) 모델 튜닝 및 최적화 전략

### (1) 하이퍼파라미터 최적화 과정

- **실험 설계**
- 객관식·주관식 문제 유형별로 응답 길이, 출력 형식, 의미 유사도에 영향을 미치는 파라미터를 단계적으로 조정하였습니다.
- **실험 결과**
- `max_new_tokens`: 512, 768, 1024를 비교하여 긴 법령 조문에서도 답변이 잘리지 않으면서도 과도한 장문 출력을 방지할 수 있는 **1024**로 확정하였습니다.
- `temperature`: 0.0 ~ 0.7 구간을 탐색한 결과, 다양성보다 **정확성과 일관성**이 중요하므로 **0.0**으로 설정하였습니다.
- `top_p`: 0.8, 0.9, 1.0을 실험하여, **1.0**에서 가장 안정적으로 정답 일관성을 확보하였습니다.

### (2) 생성 파라미터 결정 근거

- **객관식**
- 정답률(Accuracy) 극대화가 목적 → 모델이 항상 동일한 선택지를 출력하도록 `temperature=0`, `top_p=1.0`으로 결정.
- 출력 길이를 제한하기 위해 “답변:” 뒤 숫자 한 개만 나오도록 프롬프트 설계와 후처리를 병행.
- **주관식**
- 핵심 키워드 포함률과 문장 간결성이 중요 → 동일하게 결정론적 파라미터를 적용해 불필요한 변동성을 최소화.
- `max_new_tokens=1024`를 유지하여 법령 근거 문장을 충분히 포함할 수 있도록 설정.

### (3) 응답 품질 향상 및 일관성을 위한 추가 조치

- **후처리 알고리즘**
- 객관식: 정규식을 이용해 최초 숫자만 추출, 불필요한 텍스트 제거 → 평가 스크립트와 정합성 확보.
- 주관식: “답변:” 이후만 남기고 문장부호/불필요한 접두어 제거 → Cosine Similarity 계산에

불리한 잡음을 최소화.

- **검색 결과 리랭킹**
- 법령 패밀리 일치(+0.3), 문서 유형 일치(+0.6), 불일치 시 감점(-0.2~-0.5) 가중치를 적용하여 **정답 가능성이 높은 문서 우선 제공**.
- 이를 통해 모델이 환각(hallucination) 없이 실제 법령 조문을 근거로 응답하도록 개선.
- **재현성 확보**
- 파라미터와 프롬프트 템플릿을 manifest.json 및 prompts.json에 기록하여 실험과 제출 간 동일 환경을 유지.

### 3. 모델 구현 상세 설명

#### 3.1 파일 구성 및 역할

- **train.py**
- 원시 규제 문서 → 정규화/가공(HTML→CSV, PDF→조문 단위)
- 패밀리 CSV(...\_family.csv) 생성 및 **FAISS 인덱스**(법령/시행령/시행규칙/감독규정/운영기준) 빌드
- 사용 임베딩과 인덱스 경로를 **artifacts/manifest.json**에 기록(재현성)
- **inference.py**
- manifest.json 로드 → 동일 임베딩으로 인덱스 전체 warm-up
- 질의 타입 판별(객관식/주관식) → **컨텍스트 생성(RAG)** → **프롬프트 생성** → **생성** → **후처리**
- 결과를 **submission\_\*.csv**로 저장

#### 3.2 실행 플로우(End-to-End)

##### 1. 데이터 정제 & 패밀리 구축 (train.py)

- (인용/위임) 3단비교 .xls → 법령/시행령/시행규칙 3열 CSV 정리
- 조문 키(제n조(의m)) 추출 → 같은 조문번호/연결 조문 묶음(same-num/cross-num) 생성
- ...\_family.csv 저장

##### 2. 인덱스 빌드 (train.py)

- **Family 인덱스**: 각 조문별 **법령 본문(최장본)** + 연결 스니펫(시행령/시행규칙)
- **PDF 인덱스**: 시행령/시행규칙/감독규정/운영기준 PDF를 **조문 단위**로 분해하여 저장
- 모든 인덱스 경로/임베딩 모델명을 manifest.json에 기록

##### 3. 추론 파이프라인 (inference.py)

- manifest.json 로드 → 동일 임베딩으로 **FAISS 인덱스 일괄 로드**
- 질의 전처리: 객관식/주관식 판별, 법령명/조문 힌트 추출
- **컨텍스트 검색**(family + 관련 PDF 인덱스) → **리랭킹**(법령 패밀리/문서유형 가중치)
- **프롬프트 템플릿 적용**(객관식/주관식 분기) → **LLM 생성** → **후처리**(숫자 추출/"답변:" 이후만 남김)
- 제출 파일 저장(CSV)

#### 3.3 핵심 로직·함수

##### (A) 객관식 판별

```
def is_multiple_choice(question_text):  
    """  
    객관식 여부를 판단: 2개 이상의 숫자 선택지가 줄 단위로 존재할 경우 객관식으로 간주  
    """  
    # 문장 맨 앞, 뒤 공백있으면 제거하고 space 기준 나누기  
    lines = question_text.strip().split("\n")  
    # ^: 문자열 시작 / \s*: 공백 0개 이상 / 1~99 까지 숫자 / 그 뒤 공백 0개 이상  
    option_count = sum(bool(re.match(r"^\s*[1-9][0-9]? \s", line)) for line in lines)  
    return option_count >= 2 # 2개 이상의 숫자
```

## (B) 객관식 선택지 분리

```
def extract_question_and_choices(full_text):
    """
    전체 질문 문자열에서 질문 본문과 선택지 리스트를 분리
    """
    lines = full_text.strip().split("\n")
    q_lines = []
    num_list = []
    content_list = []
    options = []

    for line in lines:
        m = re.match(r"^\s*[1-9][0-9]?\s+(.*)\s*$", line) # .* 가 임의의 문자 0개 이상, \S 공백 아닌 문자 1개 #기준: r"^\s*[1-9][0-9]?\s"
        if m:
            num, content = m.group(1), m.group(2)
            num_list.append(num)
            content_list.append(content)
            options.append(f"{num} {content}")
        else:
            q_lines.append(line.strip())

    question = " ".join(q_lines)
    return question, num_list, content_list, options
```

## (C) 프롬프트 생성 (객관식/주관식 자동 판별)

```
def make_prompt_auto(text):
    if is_multiple_choice(text):
        question, num_list, content_list, options = extract_question_and_choices(text)
        allowed = ", ".join(num_list) # 예: "1, 2, 3, 4"
        prompt = [
            "당신은 금융보안 전문가입니다.\n",
            "아래 [실제 질문]에 대해 가장 적절한 **정답 선택지 번호 한 개만 출력**하세요. \n",
            "절대 미응답 하지 마세요.\n",
            f"[실제 질문]: {question}\n",
            "선택지:\n",
            f"{chr(10).join(options)}\n\n" # chr(10) = \n
            "답변:"
        ]
    else:
        prompt = (
            "당신은 금융보안 전문가입니다.\n",
            "아래 주관식 질문에 대해 정확하고 간략한 설명을 작성하세요.\n",
            "절대 미응답 하지 마세요.\n\n",
            f"질문: {text}\n\n",
            "답변:"
        )
    return prompt
```

(D) 모델 출력 후처리 (숫자/텍스트만)

```
def extract_answer_only(generated_text: str, original_question: str) -> str:
    """
    - "답변:" 이후 텍스트만 추출
    - 객관식 문제면: 정답 숫자만 추출 (실패 시 전체 텍스트 또는 기본값 반환)
    - 주관식 문제면: 전체 텍스트 그대로 반환
    - 공백 또는 빈 응답 방지: 최소 "미응답" 반환
    """
    # "답변:" 기준으로 텍스트 분리
    if "답변:" in generated_text:
        # prompt 형식에 맞게 답변:1번~ 이런 식 에서 정답으로 예측한 선지만 그대로 뽑기
        text = generated_text.split("답변:")[1].strip()
    else:
        text = generated_text.strip()

    # 공백 또는 빈 문자열일 경우 기본값 지정
    if not text:
        return "미응답"

    # 객관식 여부 판단
    is_mc = is_multiple_choice(original_question)

    if is_mc:
        # 숫자만 추출
        # 정규식: \D (숫자 아닌 문자 0개 이상 무시) / 그 다음 숫자
        match = re.match(r"\D*([1-9][0-9]?)", text)
        if match:
            # 정규식에서 첫번째 괄호인 ([1-9][0-9]?) 에 해당하는 것
            return match.group(1)
        else:
            # 숫자 추출 실패 시 "0" 반환
            return "0"
    else:
        return text
```

(E) 질의 빌더

```
def build_query_for_retrieval(q: str) -> str:
    try:
        if is_multiple_choice(q):
            question, _, _, options = extract_question_and_choices(q)
            return question + "\n" + "\n".join(options)
    except Exception:
        pass
    return q
```

(F) 문서 유형 힌트 감지

```
def _detect_doc_type_hint(text: str) -> str | None:
    s = text.lower()
    if '감독규정' in s: return '감독규정'
    if re.search(r'운영\s*기준', s): return '운영기준'
    if '시행령' in s: return '시행령'
    if '시행규칙' in s: return '시행규칙'
    return None
```

(G) 법령 패밀리 힌트 감지

```
LAW_NAME_PATTERNS = {
    '개인정보보호법': re.compile(r"(개인정보\s*보호법|개보법)", re.I),
    '신용정보법': re.compile(r"(\s*신용정보\s*\이용\s*및\s*보호\s*에\s*관한\s*법률|신용정보법)", re.I),
    '자본시장법': re.compile(r"(\s*자본\s*시장\s*법|자본시장과\s*금융투자업에\s*관한\s*법률|자본시장법)", re.I),
    '전자금융거래법': re.compile(r"(\s*전자\s*금융\s*거래\s*법|전자금융법)", re.I),
    '전자서명법': re.compile(r"(\s*전자\s*서명\s*법)", re.I),
    '정보통신망법': re.compile(r"(\s*정보통신망\s*\이용\s*촉진\s*및\s*정보보호\s*등에\s*관한\s*법률|법)|정보통신망법|망법)", re.I),
}

def detect_law_hint(text: str) -> str | None:
    for law, pat in LAW_NAME_PATTERNS.items():
        if pat.search(text):
            return law
    for law, pats in FAMILY_KEYWORDS.items():
        if any(re.search(p, text) for p in pats):
            return law
    return None
```

(H) 컨텍스트 생성을 위한 리트리벌 + 리랭킹

```
def _scope_named_stores(named_stores, law_hint: str|None):
    """law_hint가 있으면 해당 '패밀리'(신용정보/전자금융 등)로 시작하거나 포함하는 인덱스 포함(공백 무시)."""
    if not law_hint:
        return []
    family_prefix = '신용정보' if '신용정보' in law_hint else \
        '전자금융' if '전자금융' in law_hint else \
        '개인정보보호' if '개인정보보호' in law_hint else \
        '자본시장' if '자본시장' in law_hint else \
        '전자서명' if '전자서명' in law_hint else \
        '정보통신망' if '정보통신망' in law_hint else ''
    if not family_prefix:
        family_prefix = law_hint # 마지막 안전장치

    out = []
    for (name, st) in named_stores:
        base = (name or '').replace(' ', '')
        # startswith + 부분일치 모두 허용
        if base.startswith(family_prefix) or (family_prefix in base):
            out.append((name, st))
    return out
```

```

def retrieve_and_rerank_with_vec(
    q_vec,
    question_text: str,
    top_k_family: int = 6,
    top_k_decree_per_store: int = 2,
    final_top_n: int = 8,
    law_hint: str | None = None):

    assert FAM_STORE is not None, "warmup_retrievers() 먼저 호출하세요."

    results = []

    # * 스코프 결정 전에 미리 계산
    doc_type_hint = _detect_doc_type_hint(question_text) # '감독규정' / '운영기준' / '시행령' / '시행규칙' / None

    # ----- family (항상 검색) -----
    try:
        fam_hits = FAM_STORE.similarity_search_by_vector_with_relevance_scores(q_vec, k=top_k_family)
    except AttributeError:
        docs = FAM_STORE.similarity_search_by_vector(q_vec, k=top_k_family) or []
        fam_hits = [(d, 0.0) for d in docs]
    for d, rel in fam_hits:
        md = dict(d.metadata or {})
        md.setdefault("corpus_type", "법령")
        d.metadata = md
        results.append((d, float(rel)))

    # ----- 시행령 -----
    for (name, store) in _scope_named_stores(DECREE_STORES, law_hint):
        try:
            hits = store.similarity_search_by_vector_with_relevance_scores(q_vec, k=top_k_decree_per_store)
        except AttributeError:
            docs = store.similarity_search_by_vector(q_vec, k=top_k_decree_per_store) or []
            hits = [(d, 0.0) for d in docs]
        for d, rel in hits:
            md = dict(d.metadata or {}); md["corpus_type"] = "시행령"; d.metadata = md
            results.append((d, float(rel)))

    # ----- 시행규칙 -----
    for (name, store) in _scope_named_stores(RULE_STORES, law_hint):
        try:
            hits = store.similarity_search_by_vector_with_relevance_scores(q_vec, k=top_k_decree_per_store)
        except AttributeError:
            docs = store.similarity_search_by_vector(q_vec, k=top_k_decree_per_store) or []
            hits = [(d, 0.0) for d in docs]
        for d, rel in hits:

```

```

        md = dict(d.metadata or {}); md["corpus_type"] = "시행규칙"; d.metadata = md
        results.append((d, float(rel)))

# ----- 감독규정 -----
# law_hint에 '전자금융' 포함이거나, doc_type_hint가 '감독규정'이면 전 범위 검색 (제15조 회수율↑)
reg_scope = REG_STORES if (doc_type_hint == '감독규정' or (law_hint and '전자금융' in law_hint)) \
    else _scope_named_stores(REG_STORES, law_hint)
for (name, store) in reg_scope:
    try:
        hits = store.similarity_search_by_vector_with_relevance_scores(q_vec, k=top_k_decree_per_store)
    except AttributeError:
        docs = store.similarity_search_by_vector(q_vec, k=top_k_decree_per_store) or []
        hits = [(d, 0.0) for d in docs]
    for d, rel in hits:
        md = dict(d.metadata or {}); md["corpus_type"] = "감독규정"; d.metadata = md
        results.append((d, float(rel)))

# ----- 전자서명인증업무운영기준 -----
oper_scope = OPER_STORES if doc_type_hint == '운영기준' else _scope_named_stores(OPER_STORES, law_hint)
for (name, store) in oper_scope:
    try:
        hits = store.similarity_search_by_vector_with_relevance_scores(q_vec, k=top_k_decree_per_store)
    except AttributeError:
        docs = store.similarity_search_by_vector(q_vec, k=top_k_decree_per_store) or []
        hits = [(d, 0.0) for d in docs]
    for d, rel in hits:
        md = dict(d.metadata or {}); md["corpus_type"] = "운영기준"; d.metadata = md
        results.append((d, float(rel)))

if not results:
    return []

# ----- rerank -----
raw = np.array([s for _, s in results], dtype=float)
denom = (raw.max() - raw.min()) or 1.0
sim01 = (raw - raw.min()) / denom

reranked = []
for (doc, _s), s01 in zip(results, sim01):
    name = (doc.metadata or {}).get("law_full_name", "")
    ctype = (doc.metadata or {}).get("corpus_type", "")
    score = 0.75 * s01

```

```

if law_hint:
    fam_hit = (
        (name.startswith('전자서명') and '전자서명' in law_hint) or
        (name.startswith('신용정보') and '신용정보' in law_hint) or
        (name.startswith('전자금융') and '전자금융' in law_hint) or
        (name.startswith('정보통신망') and '정보통신망' in law_hint) or
        (name.startswith('개인정보보호') and '개인정보보호' in law_hint) or
        (name.startswith('자본시장') and '자본시장' in law_hint)
    )
    if fam_hit: score += 0.30
    else: score -= 0.50

# 문서유형 힌트 가정
if doc_type_hint:
    if ctype == doc_type_hint:
        score += 0.60
    else:
        score -= 0.20

reranked.append((score, doc))

reranked.sort(key=lambda x: x[0], reverse=True)

# dedup + top_n
picked, seen = [], set()
for _, d in reranked:
    key = (
        (d.metadata or {}).get("law_full_name"),
        (d.metadata or {}).get("article_no"),
        (d.metadata or {}).get("corpus_type"),
    )
    if key in seen:
        continue
    seen.add(key)
    picked.append(d)
    if len(picked) >= final_top_n:
        break

return picked

```

## (I) 컨텍스트 문자열 구성 & 프롬프트에 주입

```
def build_ctx_text(docs: List[Any], char_budget: int = 2000, per_doc_min: int = 220) -> str:
    blocks, used = [], 0
    per_doc = max(per_doc_min, (char_budget // max(1, len(docs))) - 40)
    for i, d in enumerate(docs, 1):
        md = getattr(d, "metadata", None) or {}
        body = getattr(d, "page_content", "") or ""
        head = " ".join(p for p in [{"i": i}, {"guess": guess Law Name(md)}, {"nz": md.get("chapter_no"), "nz": md.get("article_no"), "nz": md.get("article_title")} if p].strip())
        snippet = body if len(body) <= per_doc else body[:per_doc] + " ..."
        block = f"{head}\n{snippet}"
        if used + len(block) > char_budget: break
        blocks.append(block); used += len(block)
    return "\n\n".join(blocks)

def add_ctx_to_prompt(user_prompt: str, ctx_text: str | None = None) -> str:
    if ctx_text and ctx_text.strip():
        return f"[참고할 문서]\n{ctx_text}\n\n{user_prompt}"
    return user_prompt
```

### 3.4 실행 환경 및 파라미터 설정

본 모델은 Ubuntu 22.04 환경에서 Python 3.10, CUDA 12.x 기반으로 구현하였다. 주요 라이브러리는 HuggingFace Transformers, LangChain Community, FAISS, Pandas, Lxml 등을 사용하였습니다. 임베딩 모델은 BAAI/bge-m3를 적용하였으며, 임베딩 시 normalize 옵션을 활성화하여 코사인 유사도 기반 검색의 안정성을 높였습니다. 검색 시 Family 인덱스는 6개, 시행령·시행규칙 인덱스는 각각 2개 문서를 불러오는 방식으로 구성하였고, 최종 컨텍스트는 8개 문서로 제한하였습니다. 생성 모델은 max\_new\_tokens를 1024로 설정하여 주관식 답변의 맥락 손실을 최소화하였으며, temperature를 0.0, top\_p를 1.0으로 고정하여 객관식 문제에서 일관된 답변을 유도하였습니다.

### 3.5 로깅 및 예외 처리

시스템은 인덱스 로드 성공 여부, 검색된 문서 수, 최종 컨텍스트 길이를 주요 로깅 포인트로 설정하였습니다. 예외 상황에 대비해 인덱스 경로 불일치 시 경고 메시지를 출력하고, 검색 결과가 없는 경우에는 Family 인덱스만 재시도한 후에도 실패하면 컨텍스트 없이 모델을 실행하도록 설계하였습니다. 또한 객관식 문제에서 숫자 추출이 실패하는 경우에는 기본값으로 0을 반환하여 평가 과정에서 오류가 발생하지 않도록 하였습니다.

### 3.6 품질 검증 및 성능 평가

모델의 품질은 대회에서 제시한 FSKU 평가지표를 기반으로 검증하였습니다. 객관식 문제는 Accuracy, 주관식 문제는 의미 유사도와 키워드 재현율을 측정하였습니다. RAG 적용 전후를 비교한 결과, 객관식 정확도는 유의미하게 향상되었고, 주관식 답변에서도 핵심 키워드 포함률이 증가하였습니다. 또한 컨텍스트 길이와 temperature 값에 따른 ablation 실험을 통해, 현 설정 값이 가장 안정적인 성능을 보임을 확인하였습니다.

### 3.7 에러 분석

실패 사례 분석 결과, 동일 조문 번호가 법령과 시행령에 동시에 존재하는 경우 모델이 잘못된 조문을 근거로 선택하는 오류가 발생하였습니다. 이를 해결하기 위해 문서 유형과 법령 패밀리 일치 여부에 가중치를 주는 리랭킹 전략을 적용하였습니다. 또한 감독규정 관련 문제가 간헐적으로 누락되는 현상이 확인되었으며, 이에 따라 특정 키워드가 질의에 포함될 경우 감독규정 인덱스 전체를 탐색하도록 보완하였습니다. 객관식 문제에서 모델이 두 개 이상의 선택지를 출력하는 사례는 출력 후처리에서 최초 숫자 하나만 남기도록 규칙을 강화하여 해결하였습니다.

### 3.8 효율화 및 자원 사용

Family 인덱스를 별도로 구축하여 법령과 시행령·시행규칙을 동시에 검색할 수 있도록 함으로써 검색 효율성을 높였습니다. PDF 인덱스는 조문 단위로 세분화하여 불필요하게 긴 chunk를 방지하고 메모리 사용량을 절감하였습니다. 임베딩 시 배치 크기를 8로 설정하여 GPU와 CPU 환경 모두에서 안정적인 성능을 확보하였습니다. 생성 모델은 4bit 양자화와 오프로딩 기법을 적용하여 제한된 VRAM 환경에서도 원활히 동작할 수 있도록 최적화하였습니다.

### 3.9 보안 및 데이터 거버넌스

사용된 문서는 모두 공개된 법령, 시행령, 시행규칙, 감독규정, 운영기준으로 제한하여 민감정보나 개인정보는 전혀 포함하지 않았습니다. 인덱스와 임베딩 모델 설정은 manifest.json 파일에 기록하여 실행 환경을 재현할 수 있도록 하였고, 주요 로그는 보관하여 추후 감사나 검증에 활용할 수 있도록 하였습니다.

### 3.10 실행 가이드

시스템은 두 단계로 실행됩니다. 먼저 train.py를 실행하여 원시 규제 문서를 정규화하고 Family 인덱스 및 PDF 인덱스를 구축하며, 모든 설정은 manifest.json에 저장됩니다. 이후 inference.py를 실행하면 manifest.json을 불러 동일 임베딩과 인덱스를 로드한 뒤, 각 문항을 객관식/주관식으로 분류하고 RAG 기반 컨텍스트 검색을 수행합니다. 최종적으로 프롬프트를 생성하여 모델에 전달하고, 후처리를 거쳐 제출용 CSV 파일을 생성합니다.

## 4. 결론 및 향후 발전 방향

본 연구에서 구현한 생성형 AI 모델은 금융보안 관련 법령 및 규정에 기반한 질의응답을 안정적으로 수행할 수 있도록 설계되었습니다. 특히 RAG(Retrieval-Augmented Generation) 구조를 도입하여 법령·시행령·시행규칙·감독규정·운영기준을 조문 단위로 검색하고, 이를 프롬프트에 반영함으로써 모델이 단순한 통계적 생성이 아닌 실제 규제 근거에 기반한 답변을 제공할 수 있도록 하였습니다. 그 결과 객관식 문제에서는 일관된 정답률을 확보하였고, 주관식 문제에서도 핵심 키워드 재현율과 의미 유사도가 향상되어 FSKU 평가지표 상에서 실질적인 성능 개선을 달성하였습니다.

이러한 성과는 금융보안 실무 현장에서 모델을 즉시 적용할 수 있는 가능성을 보여줍니다. 예를 들어, 법령 해석 질의, 규정 준수 여부 확인, 내부 교육용 자동 문제 생성 및 풀이 등 다양한 업무 영역에서 활용할 수 있습니다. 다만 실제 도입을 위해서는 최신 개정 법령의 반영 주기, 특정 상황에서의 법적 해석 차이, 그리고 모델 출력의 신뢰성을 보증하는 체계가 필요합니다.

향후 발전 방향으로서는 첫째, **모델 성능 고도화**가 필요합니다. 현재는 결정론적 파라미터 설정을 통해 안정성을 확보했으나, 판례나 세부 해석이 필요한 질문에 대해서는 보다 정교한 추론 능력을 지원할 수 있도록 파인튜닝과 도메인 특화 데이터 보강이 요구됩니다.

둘째, **지식베이스 확장**이 필요합니다. 법령과 규정뿐만 아니라 금융감독원의 해석 지침, FAQ, 판례 등 외부 문헌을 포함하면 답변의 신뢰성과 범용성을 높일 수 있습니다.

셋째, **실무 연계 강화**가 필요합니다. 단순한 질의응답을 넘어, 사용자가 참고한 법령 조항을 함께 제시하거나 근거 문장을 자동으로 하이라이트하는 기능을 추가한다면 현업에서 더욱 신뢰성 있게 활용될 수 있습니다.

종합적으로, 본 생성형 AI 모델은 금융보안 실무 적용을 위한 유망한 출발점이며, 지속적인 성능 개선과 지식 확장을 통해 금융권 규제 준수 및 업무 효율화에 기여할 수 있을 것입니다.

## 5. 참고자료 및 인용 모델 출처

### [모델 및 알고리즘 출처]

- KT Gen AI Lab, *믿:음 2.0* (KT 고유 기술 기반 한국어 특화 생성형 AI 모델)
- BAAI, *BGE-M3: Bridging the Gap between Multilingual, Multi-Granularity, and Multi-Functionality Embeddings*, <https://huggingface.co/BAAI/bge-m3>
- HuggingFace Transformers, <https://github.com/huggingface/transformers>
- LangChain Community, <https://python.langchain.com>
- FAISS: Facebook AI Similarity Search, <https://github.com/facebookresearch/faiss>

### [외부 데이터 소스]

- 국가법령정보센터, 전자서명법/자본시장법/신용정보법/정보통신망법/개인정보보호법/전자금융거래법 관련 인용·위임 3단비교(xls) 자료 및 각 법령의 시행령·시행규칙(pdf)
- 국가법령정보센터, 전자서명인증업무 운영기준, 전자금융감독규정, 신용정보업감독규정 자료(pdf)
- 각 문서 출처: 국가법령정보센터

### [코드 및 라이브러리 라이선스]

- HuggingFace Transformers: Apache License 2.0
- LangChain: MIT License
- FAISS: MIT License
- Pandas, Numpy, Lxml: BSD/MIT License