ChatGPT

한국어 금융보안 QA 성능 향상을 위한 LLM 접근법 제안

예선 **baseline 대비 소폭 우수한 성능**을 목표로, 현실적으로 해볼 만한 **3~5가지 LLM 활용 접근법**을 제안합니다. 각 접근법마다 사용 가능한 **모델 종류**, **미세튜닝 전략**, **양자화 방법**, **필요 데이터**, **4090 GPU 환경 고려 사항**을 상세히 설명하고, **실현 가능성(난이도) 및 추천도**를 함께 평가합니다.

접근법 1: 한국어 도메인 특화 LLM 미세튜닝 (LoRA/QLoRA)

개요: 오픈소스 **대형 언어모델(LLM)**을 금융보안 QA 도메인에 맞게 **파인튜닝**하는 접근입니다. 예를 들어 Meta의 **Llama-2** (7B 또는 13B 파라미터)나 EleutherAI의 **Polyglot-Ko 12.8B** 모델 등을 베이스로 선택합니다 [1] [2]. 한국어 데이터에 강점이 있는 모델을 고르면 유리하며, **Vicuna**와 같이 사용자 지시에 잘 응답하도록 사전 튜닝된 계열도 고려할 수 있습니다. 이 베이스 모델에 대회 제공 **FSKU 금융보안 QA 문항 데이터** 및 추가 수집한 유사 도메인 데이터를 활용해 **지도학습 미세튜닝**을 수행합니다.

- **미세튜닝 전략:** 전체 파라미터를 업데이트하는 **Full Fine-Tuning**도 가능하나, 이는 **메모리와 계산 비용이 매우 큼** [3]. 대신 **LoRA**(Low-Rank Adaptation)와 최신 기법인 **QLoRA**를 활용하면 **파라미터 일부만 미세조정하여 메모리 사용을 크게 절약**할 수 있습니다 [4] [5]. LoRA는 모델 가중치를 고정한 채 작은 랭크의 보조 행렬(A, B)을 학습하여 효율적으로 튜닝하며, **QLoRA는 여기에 4비트 양자화를 접목**해 대폭적인 메모리 감소를 이룹니다 [6]. 예컨대 QLoRA 기법을 쓰면 원래 **780GB** 메모리가 필요한 모델도 **48GB 미만**으로 줄여 단일 GPU에서 튜닝 가능하고, 16비트 풀튜닝과 **동등한 성능을 유지**할 수 있습니다 [6]. 실제로 65억~130억 단어 이상의 거대 모델도 한 장의 48GB GPU로 미세튜닝 성공 사례들이 보고되었습니다. **RTX 4090 (24GB)** 환경에서는 7B~13B 모델을 LoRA/QLoRA로 튜닝하기에 적합합니다. 예를 들어 13B 모델도 4비트 양자화+LoRA로 **24GB 메모리에 수용 가능**하며 학습 시 **paged AdamW 8bit 옵티마이저** 등 최적화 기법을 적용해 자원 소모를 줄일 수 있습니다 [7].

- **양자화 방법:** 학습 단계에서는 QLoRA로 **4비트 NF4 양자화**된 가중치를 사용하고, 추론 시에는 최종 LoRA 어댑터를 기본모델에 합친 뒤 **8비트 혹은 4비트 양자화 모델**로 배포하면 메모리 여유를 더 확보할 수 있습니다. `bitsandbytes` 라이브러리를 이용하면 모델 로딩시 **INT8 혹은 INT4**로 변환 가능하고, 속도도 거의 저하되지 않습니다. 양자화로 인한 미세한 성능 손실는 LoRA 튜닝으로 상쇄되며, 사실상 **정밀도 저하 없이** 큰 모델을 활용하게 됩니다 [6].

- **데이터 요구사항:** 제공된 **금융보안 QA 데이터(FSKU 문항)**를 **모델 훈련 데이터**로 활용합니다. 객관식(Q)에는 질문+지문(있다면)+보기 -> 정답 번호, 주관식(Q)에 대해서는 질문 -> 해설형 답변 텍스트를 **지도 학습**합니다. 데이터량이 충분치 않을 경우, **유사 도메인 추가 데이터**를 확보하면 성능 향상에 도움됩니다. 예를 들어 과거 금융회사 내부 통제 QA, 정보보호 관리체계(ISMS) 관련 문제, 금융 IT 컴플라이언스 FAQ 등을 수집해 보강할 수 있습니다. 또한 **데이터 증강**으로 ChatGPT API 등을 활용해 기존 질문에 대한 다양한 표현 변형, 추가 유사 질문 생성 등을 시도할 수 있습니다 (이에 대해서는 접근법 5에서 상세히 언급) [8]. 다만 무분별한 증강보다는 **기존 정답의 정확성 유지**가 중요합니다. 학습 시 **객관식과 주관식 형태를 모두 한 모델이 다루도록** 프롬프트 포맷을 통일해야 합니다. 예를 들어 입력 텍스트에 "<질문>\n[보기: ...]" 같은 패턴으로 보기를 포함하고, 출력을 객관식이면 숫자만, 주관식이면 문장으로 생성하도록 **특수 토큰이나 지시 문구**를 두어 구분해 학습시키는 방법이 있습니다.

- **4090 GPU 환경 고려:** 7B 모델의 Full FT는 FP16 기준 약 14GB VRAM이 요구되지만 옵티마이저, 그래디언트 등으로 전체 2~3배 메모리가 필요합니다. 24GB로는 batch size를 매우 줄여 간신히 가능할 수 있으나 권장되지 않습니다. **LoRA** 적용 시는 메모리 부담이 크게 줄어 13B 모델도 학습 가능하며, **QLoRA 4bit**로는 30B급 모델도 시도해볼 법 하나, 24GB에서는 30B의 경우 아주 소량 batch로 겨우 가능하거나 offloading 필요하므

로 안전하게 13B 이하를 택합니다. **학습 속도**는 4090 한 장으로 7B 모델을 LoRA튜닝할 때 수 시간~하루 이내, 13B는 배 이상 시간이 걸릴 수 있습니다. 데이터량이 많지 않다면 에포크를 늘리지 말고 적절한 early stopping이나 검증 세트 모니터링을 통해 **과적합 방지**가 필요합니다. 최적 학습률 및 LoRA 하이퍼파라미터(`r`, `alpha` 등)는 PEFT 논문 권장값(예: r=8~16, alpha=16~32 등)을 따라가되 데이터 크기에 맞춰 조정합니다.

- **예상 효과:** 해당 접근은 모델 내부에 금융보안 도메인 지식을 **내재화**하여 **질문 맥락 이해도와 정답 생성 일관성**을 높여줍니다. Baseline이 사전튜닝되지 않은 LLM이었다면, 도메인 데이터로 SFT(Supervised Fine-tuning)를 거친 모델은 답변 형식과 내용 정확도가 개선될 것입니다. 특히 모델이 **학습 데이터에서 본 개념**은 바로 답할 수 있어 **hallucination 감소** 기대됩니다. 다만 한정된 데이터로 튜닝하므로 모델이 **훈련에서 보지 못한 신규 질문**에 대해서는 여전히 불확실성이 존재할 수 있고, 매우 세부적인 금융법 조항 같은 것은 잘못 답할 위험도 있습니다. 즉 완벽한 지식 보완은 어렵고, baseline 대비 **작지만 유의미한 성능 향상**을 노리는 현실적인 전략입니다.

- **난이도 및 추천도: 난이도: 중간**. HuggingFace Transformers의 PEFT 라이브러리로 LoRA 튜닝 파이프라인을 비교적 쉽게 구축할 수 있고 [9], GPU 한 장으로 구현 가능해 **개인/소팀도 도전할 만**합니다. 다만 학습 파라미터 튜닝, 디버깅 등에 시간이 소요되고, 데이터 준비가 성패를 좌우하므로 **데이터 전처리와 포맷 설계에 경험**이 필요합니다. **추천도: 높음.** 주어진 QA데이터를 최대한 활용하여 **모델 성능을 직접 끌어올릴 수 있는 핵심 방법**이므로, 리더보드 상위권을 노린다면 시도하는 것이 바람직합니다. 특히 baseline이 범용 한국어 LLM 그 자체였다면, 이 접근으로 도메인 특화 SFT를 거친 모델은 **baseline을 확실히 능가할 가능성**이 있습니다.

접근법 2: 도메인 지식 RAG 활용 (Retrieval-Augmented Generation)

개요: 모델 파라미터 자체를 변경하지 않고, **외부 지식소스**를 검색해 답변에 활용하는 **검색 증강 생성 (RAG)** 접근입니다 [10]. 금융보안 QA는 법령, 규정, 지침 등의 **구체적 사실 지식**을 요하는 경우가 많으므로, 관련 문서를 미리 **데이터베이스화(knowledge base)**하여 **질문에 맞는 문서를 검색**, 이를 LLM의 프롬프트 컨텍스트로 제공하면 정확도를 높일 수 있습니다 [11] [10]. RAG의 큰 장점은 **최신 정보나 도메인 특화 내용**을 모델이 몰라도 참고해 답을 생성할 수 있다는 점이며, **모델을 재학습하지 않아도** 성능 개선이 가능하다는 유연성입니다 [10]. Baseline이 만약 순수 LLM 추론이었다면, RAG를 접목한 시스템은 법 조항이나 가이드라인을 근거로 답변하여 **hallucination(환각)을 줄이고 사실적 정확성**을 높일 수 있습니다 [12].

- **모델 종류:** RAG에서는 **LLM은 주로 생성 역할**을 하고, 별도의 **임베딩 모델**과 **벡터 DB**가 검색 역할을 수행합니다. 생성 모델은 **크기가 아주 크지 않아도 됨**이 일반적입니다. 7B~13B급 LLM이면 충분하며, 한국어 대응을 위해 **다국어 임베딩에 능한 모델**이면 좋습니다. 예를 들어 **KoVicuna 7B** (한국어 데이터로 튜닝된 Vicuna 존재 시)나, 접근법 1에서 미세튜닝한 모델을 그대로 생성기에 써도 됩니다. 또는 GPT-3.5급 API를 생성 전용으로 쓰고 자체 검색만 구현하는 방법도 있으나, 본 대회 규정상 **단일 자체 모델** 사용을 권장하므로 여기서는 **로컬 LLM**을 가정합니다. 임베딩 모델은 SBERT 계열의 **한국어 문장 임베딩 모델** (예: KLUE-STB, KorCos 등)이나 다국어 **MPNet 기반 모델** 등을 활용합니다. 최신 Multilingual MiniLM이나 all-mpnet-v2 등도 한국어 의미 비교에 쓸만합니다. **선택지**: 만약 하나의 LLM으로 생성+임베딩 모두 처리하고자 한다면, LLM의 내부 임베딩을 추출하거나, LLM에게 문서들을 읽히며 관련도를 판단하게 하는 **Re-rank 방식**도 생각해볼 수 있으나 구현 복잡성이 높아지므로 일반적인 **임베딩 추출 모델 + 벡터 검색** 조합이 현실적입니다.

- **지식 문서 구축 전략:** 금융보안 QA에 적합한 **지식자료**를 선정하여 **로컬 데이터베이스**로 구축해야 합니다. 특히 참고할 만한 문서 유형과 수집처는 다음과 같습니다:

문서 유형	예시 및 주요 출처	활용 방안
금융보안 관련 법령	전자금융거래법, 신용정보의 이용 및 보호에 관한 법률, 정보통신망법 등 **법률 조문** 금융위원회 금융규정 (예: 전자금융감독규정), 금융보안 관련 고시	**질문이 법적 기준을 묻는 경우** 법 조항 원문을 인용하거나 요약하여 정확한 근거 제시

문서 유형	예시 및 주요 출처	활용 방안
금융보안 지침 및 실무 매뉴얼	금융보안원 발간 가이드라인 (예: 금융권 정보보호 업무 가이드), 금융감독원 IT·보안 지침, 각 금융기관 내부통제 매뉴얼 중 공개 가능한 부분	**실무 상황 기반 질문**(예: 보안사고 대응 절차)에 대해 권고되는 조치나 절차를 문헌 근거로 제시
정책보고서·발표자료	금융위원회 보고서 (예: 생성형 AI 활용 지원방안, 금융권 AI 보안위협 대응 발표자료), 금융보안원 연구보고서 (디지털 금융 보안 동향 등)	**최신 동향이나 사례 기반 질문**에 대해 공식 기관의 입장이나 통계를 인용하여 신뢰도 있는 답변 제공

위 자료들은 주로 **금융위원회(FSC)**나 **금융보안원(FSec)** 공식 웹사이트의 자료실, 법령 정보센터(law.go.kr), 각 기관 보도자료 등을 통해 입수할 수 있습니다. 대회 규정상 **인터넷 접속 없이 모델을 구동**해야 할 가능성이 높으므로, **사전에 크롤링/다운로드**하여 로컬에 데이터베이스로 만들어두는 것이 중요합니다.

- **전처리 방식:** 수집된 문서는 **텍스트로 변환 및 정규화**가 필요합니다. HTML이나 PDF로 제공되면 **OCR/파싱**을 거쳐 **순수 텍스트 추출**을 진행합니다. 그 후 문서당 너무 방대한 내용이 한 임베딩으로 묶이지 않도록, **내용을 청크(chunk) 단위로 분할**합니다 [13]. 일반적으로 **한 청크당 300~500자 내외**로 하되, 의미 단위를 최대한 보존하도록 **문단/항목 기준**으로 자르는 것이 좋습니다. 각 청크에는 "전자금융거래법 제21조 ..."처럼 **원 출처 식별자**(문서명 및 조항 등)을 메타데이터로 달아놓으면, 추후 모델 답변에 근거 출처를 밝히는 것도 가능해집니다. 텍스트 정규화 단계에서는 한글 문장의 **형태소 분석은 불필요**하지만, **불필요한 개행, 공백, 특수문자**를 제거하여 임베딩의 잡음을 줄입니다. 또한 **표, 목록** 등이 있다면 일렬의 문장 형태로 풀어서 저장하고, **표제어 통일**(예: "개인정보" vs "개인 정보") 등도 검색 효율을 위해 적용할 수 있습니다.

- **벡터 DB 구축:** 전처리된 청크들을 **벡터화(embedding)**하여 **벡터 데이터베이스**에 저장합니다 [13]. 임베딩에는 앞서 언급한 **문장 임베딩 모델**을 활용하여 각 청크당 임베딩 벡터(예: 768차원)를 얻습니다 [13]. 그런 다음 **유사도 검색이 가능한 벡터DB**에 적재합니다. 간편한 선택으로 **Faiss** 라이브러리를 이용해 **메모리 내에 인덱스**를 만들 수 있습니다. 데이터 규모가 크지 않다면 (수만개 이하 벡터), **Flat L2 또는 Cosine 인덱스**로도 충분하며, 규모가 크면 **IVF**나 **HNSW** 그래프 구조를 쓸 수 있습니다. 또는 오픈소스 **Milvus**, **Qdrant** 등을 도커로 띄워 사용해도 되나, 대회 환경 세팅을 고려하여 **Faiss + 파이썬 Pickle** 등으로 간단히 구축해두는 편이 안전합니다. 임베딩 저장시 **메모리**도 고려해야 하는데, 24GB GPU VRAM은 모델에 쓰이고 임베딩 검색은 CPU RAM에서 해도 무방합니다 (Faiss CPU사용). 임베딩 차원이 768이고 1만개 벡터라면 수 MB 수준이므로 큰 부담은 없습니다.

- **질문 시 동작:** 최종 시스템은 **사용자 질문을 입력**으로 받으면, 먼저 **질문 문장**을 같은 임베딩 공간으로 변환하여 **벡터 DB에서 질의 벡터와 가장 유사한 상위 k개 청크**를 검색합니다 [14]. 예를 들어 코사인 유사도 top-5를 선택하고, 해당 청크의 원문 내용을 LLM에 제공하여 답변 생성을 유도합니다 [14]. 프롬프트 형식은 예를 들어:

```
"<사용자 질문>\n[검색 결과]\n1. <청크1 내용>\n2. <청크2 내용>\n...\n위 정보를 바탕으로 질문에 답하시오."
```

와 같이 구성할 수 있습니다. LLM은 컨텍스트로 제공된 자료를 참고하며 답변을 생성하게 되고, 이때 답변에 **근거가 있는 내용만 포함**하도록 프롬프트에 명시하여 환각을 억제합니다. 또한 **객관식 질문의 경우**, 검색된 청크에 정답 근거가 있다면 LLM이 그 근거를 토대로 정확한 선택지를 고르도록 유도합니다. 필요시 LLM에게 "위 법령 발췌를 근거로 정답 번호만 출력하세요." 등의 추가 지시를 할 수 있습니다.

- **4090 GPU 환경 고려:** RAG 접근에서는 **LLM의 추론 메모리** + **임베딩 연산 자원**이 고려됩니다. 임베딩 추출은 미리 off-line으로 해두므로 실시간 추론 시에는 **질문 임베딩 계산** 정도만 필요합니다. 이는 CPU로 해도 수십 ms 이내로 가능하지만, 4090의 Tensor Core를 활용해 임베딩 모델을 올리면 더 빠르게 처리할 수 있습니다. 다만 GPU에는 이미 LLM이 올라가 있을 테니, 임베딩 계산은 **멀티스레드 CPU 처리**로 충분합니다. LLM 추론 시에는 검색된 상위 문맥을 넣어야 하므로 **프롬프트 토큰 길이**가 늘어나는데, 7B 모델 기준 2048~4096 토큰

max에서 문맥 300~500자 * 5개 ≒ 1500자(750 토큰 수준) + 질문 길이를 합쳐도 무리 없습니다. 13B 모델도 비슷합니다. **응답 속도**는 약간 느려질 수 있으나 (검색+LLM생성), 4090이면 7B 모델 생성 속도가 매우 빠르므로 충분히 실용적입니다.

- **예상 효과:** 적절한 문서만 잘 구축된다면, LLM이 몰랐던 법 조항이나 지식을 검색 결과로 제공함으로써 **사실적 질문에 대한 정확도가 큰 폭으로 향상**될 수 있습니다 [12]. 예를 들어 "개인정보를 제3자에게 제공할 때 법적으로 필요한 조치는?" 같은 질문이 주어지면, 모델 스스로 기억하는 지식이 부족해도 **개인정보보호법 관련 조항**을 찾아보고 그 내용을 요약하여 답하는 식입니다. 이 경우 근거가 명확하니 **정답의 신뢰성**도 높고, 답변에 출처도 달 수 있습니다. RAG는 **특정 도메인 지식 격차를 메우고 환각을 줄이는** 데 매우 효과적이며 [11] [10], 모델을 새로 튜닝하는 것보다 **노력 대비 효율**이 높을 수 있습니다 [12]. 특히 금융보안처럼 **최신 동향**(예: 최근 발생한 금융 해킹 사례 등)을 묻는 질문에는 사전학습된 LLM으로는 답하기 어려운데, RAG로 미리 관련 보고서를 포함해두면 최신 정보도 답변 가능합니다.

- **한계:** RAG의 성능은 **지식 베이스의 포괄성**에 달려 있습니다. 만약 구축한 문서에 답이 없다면 모델은 여전히 추측해야 하므로 효과가 없습니다. 또 **질문-문서 매핑**이 어려운 경우, 엉뚱한 검색 결과가 들어와 오히려 혼란을 줄 수도 있습니다. 객관식의 경우, 관련 단서가 여러 문서에 흩어져 있으면 모델이 통합 추론해야 하므로 난이도가 높습니다. 따라서 **자료선정의 적절성**과 **벡터 검색 튜닝(임베딩 모델 선택, top-k 조정)**이 필요합니다.

- **난이도 및 추천도: 난이도: 중~상**. 구현 측면에서 **LangChain** 등의 프레임워크를 활용하면 비교적 쉽게 파이프라인을 만들 수 있으나, **도메인 자료 수집/정제**에 시간이 꽤 필요합니다. 팀원 중 한 명이 크롤링과 전처리에 전념해야 할 수도 있습니다. 또한 LLM에 비해 검색/임베딩 모듈은 **튜닝 관련 정보가 적어 시행착오**가 있을 수 있습니다. 예를 들어 임베딩 모델에 따른 검색 성능 비교, 한글 형태소 처리 여부 등에 따른 영향 등을 실험해야 할 수도 있습니다. **추천도: 매우 높음 (상위권 지향 시)**. 비록 준비가 번거롭지만, **baseline 대비 정확도 향상의 폭이 가장 클 수 있는 접근**입니다. 특히 법/규정 기반 문제가 많은 경우 RAG 없이는 완벽히 맞히기 어려운데, RAG를 도입하면 **적중률을 상당히 끌어올릴 수 있습니다**. **실현 가능성**도 높은 편인데, 왜냐하면 LLM 튜닝보다 **GPU 연산을 덜 쓰고도** 성능 보강이 가능하므로 **리소스가 한정된 해커톤** 환경에 잘 맞습니다 [12]. 단시간에 결과를 내야 한다면, **기존 모델 + RAG 조합이 빠른 개선책**이 될 수 있어 적극 권장합니다.

접근법 3: 프롬프트 튜닝 및 고도화된 Prompt Engineering

개요: 모델을 재학습하지 않고 **입력 프롬프트를 최적화**하거나, 아주 소량의 학습가능 파라미터(soft prompt)만 추가로 훈련하는 **프롬프트 튜닝** 기법입니다. 또한 별도의 학습 없이 **Prompt Engineering**만으로 성능 향상을 도모할 수도 있습니다. 이 접근법은 데이터나 시간 자원이 부족한 상황에서 **모델의 응답 형태와 방향을 개선**하는데 유용합니다 [15] [16].

- **모델 종류:** 기존 **baseline 모델 자체**를 변경 없이 사용합니다. (원본 LLM이 그대로 기반이 됩니다.) 이 접근에서는 모델 선택보다는 **프롬프트 설계 전략**이 핵심이므로, baseline이 만약 KoGPT든 Llama2든 그대로 둡니다. 만약 소규모 파인튜닝이 가능하다면, **Prefix-Tuning** 같은 기법으로 모델 앞부분에 학습된 벡터를 추가하는 형태를 고려할 수 있습니다. 이는 **Parameter-Efficient Fine-Tuning (PEFT)**의 하나로, 모델 파라미터는 그대로 두고 **입력 토큰에 대한 embedding**만 조금 학습하는 방식입니다 [17]. 일종의 **가상 토큰**을 만들어 내재된 지식을 끌어내는 역할을 하는 것이죠.

- **프롬프트 엔지니어링:** 우선 **수동 프롬프트 디자인**으로 모델의 성능을 끌어올립니다. 한국어 금융보안 QA에 특화되도록, **시스템 메시지**나 **역할 플레이** 지시를 활용합니다. 예를 들어 **시스템 프롬프트**로 “당신은 금융보안 전문가입니다. 질문에 대해 정확하고 간결한 답변을 제공합니다.”라는 지침을 넣어 **모델의 톤과 정확성**을 유도할 수 있습니다. 또한 **출력 형식**을 엄격히 규정하여 객관식이면 숫자만, 주관식이면 한두 문장으로 답하도록 지시할 수 있습니다. 프롬프트 예시:

```
시스템: 당신은 금융보안 전문가 AI입니다. 사용자 질문에 대해 정확한 법적/기술적 근거에 기반한 답변을 제공합니다.
사용자: [여기에 실제 질문 삽입]
assistant:
```

위와 같이 시스템 프롬프트를 늘려주는 것만으로도 baseline 대비 일관성이 좋아질 수 있습니다. **Few-shot 예제**를 포함하는 것도 강력한 방법입니다 [18]. 모델 입력에 **유사 형식의 Q&A 예제 2~3개**를 함께 제공하면, 모델이 **문제 유형을 더 잘 파악**하게 됩니다. 예를 들어 객관식 예시 한 개 (질문-보기-정답번호)와 주관식 예시 한 개(질문-해설형 답변)를 넣고, 이어서 실제 질문을 물으면, 모델이 예시를 참고해 **정답 형식**을 맞추고 내용을 유추하는 능력이 향상됩니다 [19]. 이러한 **Few-shot prompting**은 모델 파라미터를 변경하지 않으면서도 성능을 올리는 대표적인 기법입니다 [18].

- **프롬프트 튜닝 (Soft Prompt 학습):** 추가로, 개발 기간이 조금 더 있다면 **프롬프트 튜닝**을 적용할 수 있습니다. 이는 모델이 해석하는 입력의 맨 앞에 위치하는 몇 개의 토큰 임베딩을 **학습 가능한 벡터로 설정**하여, 마치 짧은 **지식 혹은 지시사항을 모델이 내부적으로 항상 참고**하도록 만드는 것입니다 [15] [20]. 이 접근은 **전체 파라미터는 그대로 두고** 일부 **프롬프트 관련 가중치만 업데이트**하므로, **학습 시간과 메모리가 매우 적게** 듭니다 [17]. 예컨대 20개 토큰 길이의 soft prompt라면 수만 개 파라미터 수준이므로, 수백~수천만개 파라미터를 가진 모델에 비하면 미미합니다. 한국어 QA 데이터에 대해 이 soft prompt를 **LoRA 학습과 유사한 방식으로 학습**시키면, 모델이 그 프롬프트 벡터를 통해 **도메인 힌트**를 얻게 됩니다 [21]. 결과적으로, 별도의 지시문을 매번 넣지 않아도 모델이 **금융보안 문맥**에 맞춰 답변하는 효과를 냅니다. 기술적으로는 Huggingface P-Tuning v2 같은 기법으로 구현 가능합니다.

- **양자화:** 프롬프트 튜닝은 모델 자체를 미세튜닝하지 않으므로, **baseline 모델을 8-bit/4-bit로 양자화**해서 사용해도 무방합니다. 양자화 모델에도 soft prompt는 적용 가능합니다(임베딩 계산만 약간 달라질 뿐). 따라서 4090 메모리가 부족한 경우 baseline 모델을 4-bit GPTQ 등으로 올리고, 프롬프트 벡터를 결합하여 추론해도 성능에 큰 지장 없습니다.

- **데이터 요구사항: 모델 학습을 하지 않는 경우**, 별도 데이터는 필요 없고, **기존 개발 세트**를 이용해 프롬프트 실험을 거듭하면 됩니다. 예를 들어 개발 데이터의 일부 질문에 대해 다양한 프롬프트 버전을 시험하여 채점해보고, 가장 성능이 좋거나 일관성이 높은 프롬프트 템플릿을 선정하는 방식입니다. **프롬프트 튜닝(soft prompt 학습)**을 하는 경우에는 baseline 모델에 대해 **기존 QA 데이터를 그대로 학습**시킵니다. 이때 정답을 맞혔는지보다는 **모델 출력이 원하는 형식으로 나오는지**를 로스 함수로 삼아 학습시키게 됩니다. 데이터량이 적다면 에포크 수를 늘리기보다 **학습률을 작게 하고 반복**하면서 validation 기준으로 early stop을 거는 게 좋습니다.

- **4090 환경 고려:** 프롬프트 튜닝의 학습은 **파라미터 수가 매우 작기 때문**에 4090에서는 부담이 거의 없습니다. 수천 개 QA로 몇 에포크 돌리는데 몇 분~한두 시간 내로 끝날 수 있습니다. 추론 시에도 모델 크기만큼 메모리가 필요할 뿐, 추가 연산은 미미합니다. Few-shot 프롬프트의 경우, 예시를 몇 개 넣으면 **컨텍스트 길이가 늘어나 VRAM 사용 증가**와 **응답 속도 저하**가 있을 수 있습니다. 그러나 2~3개 예시는 큰 문제는 아니며, 4090으로 2048토큰 정도까지는 무난합니다.

- **예상 효과:** 이 접근은 **출력 양식 개선 및 약간의 성능 향상**을 기대할 수 있습니다. 특히 baseline이 정답을 알아도 **형식이 틀리거나 불완전하게 답하는** 경우가 있는데, 프롬프트 최적화를 통해 **정답 포맷 준수율**이 올라갑니다. 예를 들어 baseline이 객관식 질문에 불필요한 설명을 달거나 헷갈리게 답했다면, few-shot 예제로 숫자만 답하는 것을 보여주면 그런 실수가 줄어듭니다. 또한 **모델의 잠재지식 활용 극대화**라는 측면에서, 잘 설계된 지시문은 모델로 하여금 애매한 질문도 **스스로 논리적 추론**을 하도록 유도할 수 있습니다. (예: "각 보기 내용을 하나씩 검토하고 가장 알맞은 답을 고르세요"라고 프롬프트에 쓰면, 모델이 체계적으로 옵션을 평가하여 답을 내는 체인이 유도될 수도 있습니다.)

- **한계:** 프롬프트만으로는 **새로운 지식을 주입할 수 없기 때문에**, **도메인 지식 자체의 한계**는 여전히 존재합니다 [2]. 모델이 모르던 금융보안 법 세부내용을 갑자기 알게 되진 않으므로, 이 접근은 **지식 강화보다는 출력 통제**

에 가깝습니다. 또한 few-shot 예제를 많이 넣으면 모델이 **예시에 과적합**해서 특이한 질문에 다소 **획일적인 답변**을 내놓을 우려도 있습니다. 프롬프트 튜닝의 경우, 작은 데이터에 과적합되면 오히려 **편향된 답변**이 나올 수 있으므로 주의해야 합니다.

- **난이도 및 추천도: 난이도: 낮음~중간**. 모델을 다시 훈련하지 않는 범위에서는 누구나 시도할 수 있어 난이도가 낮습니다. 프롬프트 실험은 **창의성과 도메인 이해**가 요구될 뿐, 코딩보다 **문구 작성 능력**이 좌우합니다. 다만 soft prompt 학습은 약간의 튜닝 코드 작성이 필요하나, 이는 LoRA보다 간단한 수준입니다. **추천도: 중간**. 이 접근은 **baseline에 비해 큰 비용 없이 성능을 약간 개선**시키는 안전망으로 추천됩니다. 특히 **대규모 파인튜닝이 어려운 참가자**의 경우 프롬프트 개선만으로도 부분 점수를 얻을 수 있습니다. 하지만 **획기적인 성능 향상은 제한적**이므로, 가능하면 다른 접근(튜닝, RAG)과 **병행**하여 보조 수단으로 사용하는 것이 좋습니다. 예를 들어 접근법 1이나 2를 적용한 후에도 추가로 프롬프트 엔지니어링을 가미하면 더 좋은 결과를 얻을 수 있으므로, **부가적 최적화 수단**으로 활용할 것을 권장합니다.

접근법 4: Encoder-Decoder 모델 활용 (예: Flan-T5 계열 Fine-tuning)

개요: 일반적인 LLM은 GPT류의 **Decoder-Only** 모델이지만, **Enc-Dec 구조**의 **T5 계열** 모델을 활용하는 방법입니다. Google의 **Flan-T5**는 다양한 언어작업에 Instruction 튜닝되어 있어, 특히 **QA 및 분류 태스크 성능이 우수**한 것으로 알려져 있습니다 [22]. 실제 연구에서도 **Flan-T5 기반 모델이 Vicuna 등의 Decoder Only 모델보다 객관식 QA에 강점**을 보인다고 보고되었습니다 [22]. 따라서 **한국어**에 적절한 **다국어 T5** 모델 (예: mT5, 또는 한국어로 추가학습된 KoT5가 있다면)이나 공개된 **Flan-T5 Large/XL/XXL** 모델을 가져와 **금융보안 QA 데이터로 미세튜닝**하는 접근입니다.

- **모델 종류:** Flan-T5에는 크기별로 Base(0.25B), Large(0.78B), XL(3B), XXL(11B) 등이 있습니다. **4090 24GB**에서는 **XL(3B)** 모델은 full fine-tuning도 무리 없고, **XXL(11B)**도 16-bit로는 약간 어려우나 8-bit로는 가능할 수 있습니다. 인코더-디코더 모델은 디코더만 있는 모델보다 같은 파라미터에서 **메모리 요구량이 높지만**, T5는 구조상 **Feed-Forward 감소 등의 최적화**가 있어 파라미터 대비 효율이 좋습니다. 우선 3B 모델을 튜닝해보고, 여력이 되면 11B까지 시도해볼 수 있습니다. **한국어 대응** 측면에서 Flan-T5는 멀티링구얼 데이터도 일부 포함하지만 주로 영문에 강합니다. 따라서 **다국어 버전인 mT5**에 Flan식 instruction tuning을 거친 **mT0** 모델이나, HuggingFace에 공개된 **Flan-T5의 multilingual fine-tuned 버전**이 있다면 그것을 사용하는 것이 좋습니다. (예를 들어 "google/mt5-xl" 등 기본 checkpoint 후 자체 instruct tuning 필요.) 만약 **국내 공개 KoT5** 모델이 존재한다면 (KT Brainko의 KoGPT-일반언어이해 버전 등), 검토 가능합니다.

- **미세튜닝 전략: Full fine-tuning**이 비교적 저비용으로 가능합니다. 3B 모델은 FP16으로 약 6GB VRAM 정도이므로, 배치 사이즈 여유롭게 잡아도 24GB에서 학습이 빠르게 이뤄집니다. 11B 모델은 FP16 시 약 22GB로 간당하지만, **bfloat16** 또는 **8-bit Adam** 등을 쓰면 충분합니다. 또한 **LoRA**를 Enc-Dec 모델에도 적용할 수 있습니다. Huggingface PEFT에서 `task_type="SEQ_2_SEQ_LM"`으로 LoRA 설정하여, 주로 **디코더 쪽 MHA와 FFN에 LoRA**를 넣고 튜닝하면 메모리 소모를 크게 줄일 수 있습니다. LoRA로 하면 11B 모델도 24GB에서 넉넉히 미세튜닝 가능할 것입니다. Encoder 부분까지 LoRA할 필요는 없고, 디코더에 주로 적용하면 되므로 연산량 감소 효과도 있습니다.

- **양자화:** 학습시에는 8-bit Adam 옵티마이저 등을 쓰거나, QLoRA만큼은 아니어도 **부분 양자화**하여 학습할 수 있습니다. 추론 시에는 **INT8 우선** 권장합니다. T5계열은 Transformers 라이브러리에서 `load_in_8bit=True` 옵션으로 메모리 절약이 용이합니다. 11B 모델을 8bit로 로드하면 VRAM 12GB 안팎으로 구동 가능하므로, 4090에서 **하나의 프로세스에 두 모델(ensemble)** 올리는 등의 여유도 생깁니다. (하지만 단일 모델 사용 조건이므로 ensemble보다는 여유 메모리로 컨텍스트 늘리는 데 쓰면 됩니다.)

- **데이터 요구사항:** 접근법 1과 거의 동일하게, **FSKU QA 데이터**를 supervision signal로 사용합니다. Enc-Dec 모델의 입력은 **질문 (및 보기)**, 출력은 **정답**으로 구성합니다. 객관식의 경우 정답을 "1", "2" 등 **문자 토큰** 하나로 생성하도록 하고, 주관식은 자연어 문장으로 생성하도록 합니다. 이때 **멀티태스크 학습**으로 객관식/주관식을 같이 다루게 되므로, 출력 포맷이 혼재하는 것을 모델이 학습해야 합니다. 한 가지 기법은 **출력 앞에 특정 토큰**

을 두어 유형을 구분하는 것입니다. 예를 들어 객관식 정답은 출력 시작을 "<MC>" 토큰으로, 주관식은 "<SA>" (short answer) 토큰으로 시작하도록 데이터에 넣고, 훈련 후에는 해당 토큰을 무시하고 후처리하거나 또는 그 토큰이 나오면 분기 처리하는 방법입니다. 다만 이 복잡도를 피하려면, **모델에게 규칙을 익히도록** 하는 수밖에 없습니다. Flan-T5는 원래 다양한 형식의 출력에 익숙하므로, 큰 문제없이 두 형태 모두 생성해내리라 기대합니다.

- **4090 환경 고려:** T5 모델 학습은 **Flax/JAX**로도 가능하나, 여기서는 PyTorch + Accelerate로 하는 것이 일반적일 것입니다. 4090 한 장으로 11B 모델도 LoRA로 충분히 돌릴 수 있으나, **학습 속도**는 7B LLaMA보다 느릴 수 있습니다(모델이 크므로). 예를 들어 3B는 수 시간, 11B는 하루 가까이 걸릴 수 있습니다(데이터양과 epoch 설정에 따라). **메모리 관리** 측면에서, 11B full FT는 optimizer 상태까지 합치면 48GB 요구될 수 있으므로 반드시 8bit optimizer나 DeepSpeed ZeRO Offload 등을 써야 합니다. LoRA 사용 시엔 훨씬 적게 들겠죠. 추론 시 8bit 로드하면 4090에 충분하며, **응답 생성 속도**는 11B T5면 7B LLaMA와 비슷하거나 조금 빠를 수 있습니다(T5는 병렬 생성 가능성이 약간 있음).

- **예상 효과:** Encoder-Decoder 구조의 장점은 **입력 질문을 한 번에 인코딩**하여 문맥을 잘 파악하고, **디코더가 답만 생성**하므로 **정확한 매핑**에 유리하다는 점입니다. 특히 **객관식처럼 입력과 출력이 뚜렷이 구분**되는 경우, Enc-Dec 모델은 입력을 완전히 읽고 나서 답을 내놓기에, **보기 중 정답 선택**을 더 체계적으로 할 수 있습니다 [22]. 이는 사람으로 치면 질문지를 다 읽고 답안만 작성하는 방식과 비슷합니다. 반면 GPT류는 질문을 읽으며 곧바로 답을 생성하기에 (물론 내부적으로는 다 읽겠지만) 형식 제약이 덜 엄격할 수 있습니다. 따라서 Flan-T5 튜닝 모델은 객관식 정답을 정확히 한 글자로 내보내는 **포맷 측면 강점**이 기대됩니다. 또한 Flan-T5는 **사전 학습된 멀티태스크 데이터**에 QA, NLI 등이 많아, **추론 능력**이 준수합니다 [23]. 주관식도 짧은 텍스트 생성의 형태이므로, Flan-T5의 장점 (간결하고 사실적인 답변)을 살릴 수 있습니다. 결과적으로 baseline이 GPT-계열 6B 수준이라면, 동일 크기라도 Flan-T5 Large정도의 모델이 특정 QA에서 **더 나은 성능을 보일 가능성**이 있습니다.

- **한계:** 다만, **한국어**에 완벽히 최적화된 모델은 아니므로, 순수 Decoder인 KoGPT-6B같은 모델보다 초기에 한국어 능력이 떨어질 수 있습니다 [2]. 이를 미세튜닝 데이터로 보완한다고 해도, 완벽한 문법이나 맥락 이해는 Llama2같은 최신 모델보다 부족할 여지가 있습니다. 또한 Enc-Dec 모델은 답을 한두 문장으로 생성하는 데는 좋지만, **장문 생성**이 필요하면 디코더가 금방 멈추거나 하는 경우도 있습니다. 금융보안 QA는 긴 답변까지는 요구되지 않을 것으로 보입니다만, 혹시 해설형으로 길게 써야 할 때 부족할 수 있습니다. 마지막으로, T5류는 사전훈련 corpus에 코드나 수학, 논리 추론 부분이 GPT계열보다 약하다는 지적이 있습니다. 특정 보안상황에서 **논리적 추론**이 필요할 때 성능이 제한될 수 있습니다.

- **난이도 및 추천도: 난이도: 중간**. 익숙하지 않다면 Decoder-only 모델보다 T5 fine-tuning이 절차상 복잡해 보일 수 있습니다. 하지만 HuggingFace **Seq2SeqTrainingArguments**를 쓰면 크게 다르지 않으며, 오히려 generation을 `generate()`로 쉽게 할 수 있는 장점도 있습니다. 3B급은 가볍게 다룰 수 있어 **개발 난이도가 높지 않고**, 데이터세트도 크지 않아 학습이 빨리 끝납니다. **추천도: 중간**. 이 접근은 **baseline 대비 "소폭" 성능 개선** 목표에 부합할 수 있으나, 큰 혁신을 보장하진 않습니다. 다만 **객관식이 많은 평가**라면 적극 고려할 만합니다. Vicuna류 대비 Flan-T5가 객관식 정답 정확도가 좋았다는 보고가 있는 만큼 [22], 만약 baseline이 객관식 처리에 약했다면 이 대안을 통해 **몇 점 차이**라도 벌릴 수 있을 것입니다. 한편으로 주관식 생성능력은 LLM 크기에 좌우되므로, 3B나 11B T5가 13B Llama수준의 언어능력을 낼지는 미지수입니다. **시간과 자원이 충분하면 부차적 실험**으로 권장하되, **핵심 접근**으로 삼을지는 팀의 경험에 따라 판단해야 합니다. 이전에 T5류 모델 튜닝 경험이 있다면 안정적으로 결과를 낼 수 있으므로 유리하고, 없으면 시행착오가 있을 수 있습니다.

접근법 5: 상위 LLM 활용 데이터 증강 및 지식 증류

개요: 직접 모델을 튜닝하기 전에, **더 강력한 LLM**(예: GPT-4, ChatGPT)을 활용하여 **훈련 데이터를 확장하거나** 모델이 학습할 **정답의 질을 향상**시키는 방법입니다. 일종의 **Knowledge Distillation(지식 증류)** 또는 **Self-Instruct** 데이터 생성 접근법으로 볼 수 있습니다. 예를 들어 OpenAI GPT-4가 금융보안 질문에 꽤 정확한 답을 줄 수 있다면, 이를 활용

해 **기존 QA 쌍을 보완**하거나 **새로운 유사 질문에 대한 모범 답안을 생성**할 수 있습니다. 그렇게 생성한 추가 데이터로 우리 LLM을 튜닝하면, **상위 모델의 지식과 표현력을 작은 모델이 모방**하게 되어 성능이 올라갈 수 있습니다 [24] [8] .

- **모델 종류:** 실제 서비스에는 우리의 파인튜닝된 모델이 쓰이지만, 이 접근의 준비과정에서 **외부 API나 거대모델**을 사용합니다. 예컨대 **GPT-4**나 **ChatGPT(GPT-3.5)** API, 또는 오픈소스 중 더 대규모인 **Llama2-70B** 등을 지식 원천으로 삼습니다. 이러한 모델들은 금융보안 도메인에 대해 사전학습된 지식이 풍부할 가능성이 높고, 최소한 **일반 상식이나 논리 면에서 작은 모델보다 뛰어납니다**. 우리의 7~13B 모델이 직접 답하기 어려운 질문도 GPT-4에게 물어보면 상당히 정확한 답을 얻을 수 있을 겁니다. 다만 **도메인 특화 정확도는 담보되지 않을 수 있어**, 생성된 답을 검증하는 과정이 필요합니다.

- **데이터 증강 전략:** 먼저, **기존 FSKU 문항**에 대해 상위 LLM으로 **정답 생성/교정**을 합니다. 예를 들어 다소 모호한 주관식 질문에 대해 GPT-4에게 물어 **모범 해설**을 생성하게 하고, 이를 우리의 훈련 답안으로 채택합니다. baseline이 제공한 정답도 있지만, 종종 **간략하거나 충분한 설명이 없는 경우**가 있으므로, GPT-4의 답변을 참고해 **보다 풍부한 정답 데이터셋**으로 확장할 수 있습니다. 둘째, **유사 질문 생성**: 금융보안 법령이나 지침서 내용을 GPT-4에 요약해주면서 "여기에 관련된 예상 질문 3개 만들어줘" 식으로 프롬프트를 주면, 실제 데이터와 다른 새로운 질문들을 얻을 수 있습니다. 이렇게 생성된 질문에 대해서도 GPT-4에게 답을 달라고 하면 **새로운 QA 페어**가 생깁니다. 예를 들어 개인정보보호법 주요 조항들로 수십 개의 문답쌍을 만들 수도 있습니다. Stanford Alpaca 사례에서는 GPT-3를 사용해 52k개의 instruction 데이터를 생성했고, 7B Llama를 튜닝해 OpenAI 모델과 유사한 성능을 달성했습니다 [24] [8] . 우리도 이와 비슷하게, **수백~수천 개 수준의 추가 QA**를 생성해볼 수 있습니다. 마지막으로, **모델 반응 교정**: 우리 모델이 틀리기 쉬운 문제 유형을 선별하여 GPT-4로 하여금 그에 대한 **해설 및 정답 근거**를 작성하게 한 뒤, 이를 훈련시켜 모델이 오답을 고치도록 하는 방법도 있습니다. 예컨대 baseline이 잘못 답한 질문들을 분석해 GPT-4에게 “이 질문에 대한 정답과 이유를 설명해줘” 하면 상세한 이유와 함께 정답을 주는데, 이 데이터를 모델에 추가 학습시키면 나중에 유사 문제에 더 잘 대응할 수 있습니다.

- **미세튜닝 전략:** 증강된 데이터를 가지고는 **접근법 1 혹은 4의 방식**으로 미세튜닝을 진행합니다. 이미 상위 모델이 포맷을 잘 지켜 답변했을 것이므로, 우리 모델도 **그 포맷을 모방**하게 됩니다. 예를 들어 GPT-4가 객관식은 그냥 번호만 내놓았다면, 우리 모델도 자연스레 그럴 확률이 높아집니다. Distillation을 더 엄격히 하려면, GPT-4의 Chain-of-Thought(사고과정)까지 생성하게 하여 모델에게 **해설 단계**를 함께 학습시키고, 최종 출력만 답으로 내도록 훈련하는 것도 생각해볼 수 있습니다. 하지만 이번 과제에서는 **최종 정답 산출**이 목표이므로, 단순히 질문->정답의 지도학습으로 충분합니다. 가능하다면 **cross-entropy loss** 대신 **KL-divergence**로 두 모델 분포를 맞추는 식으로 distillation loss를 쓸 수도 있으나, 복잡하게 가지 않아도 됩니다. 또한 학습시 **원본 데이터와 생성 데이터의 비율**을 조정해야 합니다. 생성 데이터는 품질이 들쭉날쭉할 수 있으니, **원본 실제 데이터 비중을 높게**(예: 70% 실제 + 30% synthetic) 두는 것이 안전합니다.

- **양자화:** 이 접근 자체는 데이터 준비에 관한 것이므로, 최종 모델 추론에는 기존와 동일하게 양자화 가능합니다. 상위 LLM(API)은 외부 클라우드에서 동작하므로 우리 GPU와 무관합니다.

- **데이터 요구사항:** 상위 모델 접근에는 **API 호출 비용/제한**을 염두에 둬야 합니다. GPT-4는 유료이며 응답당 토큰 비용이 있습니다. 다행히 생성해야 하는 데이터의 양은 수백~수천 쌍 정도면 충분할 것이므로, 1000쌍 생성에 $0.0x$/1K tokens 의 비용 합쳐 수십 달러~100달러 내로 가능할 수 있습니다 (명확한 계산은 필요). 시간이 부족하면 **가장 필요한 부분**에 집중합니다. 예를 들어 baseline 성능을 검토했을 때 **특정 챕터 (예: 망분리, 인증)에 오답이 많았다**면 그 부분에 대해 집중적으로 Q&A를 생성해 보강합니다.

- **4090 환경 고려:** 증강 데이터 생성은 **모델 외부**에서 이뤄지므로, 4090에는 영향 없습니다. 다만 새로운 데이터로 학습을 더 시켜야 하니, **추가 학습시간**이 필요합니다. 예선 종료까지 시간 여유가 있다면 수행하고, 촉박하다면 차라리 RAG처럼 다른 접근에 시간 쓰는 게 나을 수 있습니다. **병렬**로 진행할 수 있다면 최상이겠지요. 또한 증강데이터로 모델 튜닝 시 **에포크 조절**이 중요합니다. 데이터가 2배로 늘었다면 epoch을 줄이거나, learning rate 스케줄을 달리해 과적합을 방지합니다.

- **예상 효과:** 상위 LLM을 교사로 활용하면 **작은 학생 모델의 성능 향상**이 여러 연구에서 입증되었습니다 [24] [8]. 우리 시나리오에서도 GPT-4 정도면 금융보안 지식을 꽤 포함하고 있어, **정확하고 풍부한 정답**을 생산해 줄 것입니다. 학생 모델은 이를 학습하며 **자신이 원래 학습한 범위를 넘어서는 추가 정보와 표현습관**을 얻게 됩니다. 특히 GPT-4는 답변을 조리 있게 하므로, 학생 모델도 **논리적인 답변 구조**를 모방할 수 있습니다. 또한 baseline 대비 성능이 크게 도약하기 위해선 **새로운 양질의 데이터**가 필수인데, 상위 LLM이 이를 비교적 쉽게 공급해줄 수 있습니다. 요약하면, 적절히 활용 시 **baseline 대비 확실한 성능 개선**을 이끌 **잠재력이 큰 접근**입니다.

- **한계:** 생성된 데이터의 **품질 통제**가 어렵습니다. 상위 모델이 항상 옳지 않을 수 있고, 때로는 **그럴듯한 오류(hallucination)**를 포함한 답을 내놓기도 합니다. 만약 그런 잘못된 데이터를 학습하면 오히려 모델을 망칠 위험이 있습니다. 따라서 **인간 검수**가 중요하며, 리소스가 허락한다면 생성된 Q&A를 팀원이 빠르게 리뷰해야 합니다. 특히 법률 관련 답변은 GPT-4라도 틀릴 수 있으므로, 법 조문과 대조 검증이 필요합니다. 또한 distillation은 모델이 **교사 모델의 편향까지 따라배울 수 있는** 부작용이 있습니다. 예컨대 GPT-4가 "~인 것 같습니다"처럼 애매한 표현을 쓰면 학생도 그렇게 답할 수 있습니다. 이런 부분은 프롬프트로 교사 답변 스타일을 통제하여 (“간결히 단정적으로 답하라” 등) 완화해야 합니다. 시간 대비 효과 측면에서도, 준비 시간이 많이 필요하면 오히려 본말전도일 수 있습니다.

- **난이도 및 추천도: 난이도: 중간**. 기술적으로 어렵진 않으나 **준비 작업량이 많아** 중간으로 평점했습니다. API 활용과 데이터 가공, 추가 학습 등이 필요하고, 잘못하면 시간만 소모할 수 있습니다. **추천도: 상황별 권장**. 만약 **baseline 대비 큰 향상**을 노려야 하는데 우리 모델/데이터만으로 한계가 뚜렷하다면 이 방법이 돌파구가 될 수 있습니다. 특히 **다른 팀들도 유사한 LLM 튜닝을 할 것으로 예상**된다면, 우리만의 추가 데이터로 우위를 점할 수 있으므로 유의미합니다. 예선 통과를 위해 1~2%포인트의 정확도 차이라도 만들어야 한다면, **투자 가치가 있습니다**. 반면 시간이 빠듯하고 다른 접근(위의 1~4)이 아직 제대로 완성되지 않았다면, 이 부분은 **후순위**로 두는 것이 좋겠습니다. 증강 데이터 없이도 baseline을 약간 이길 수 있다면 굳이 risk를 늘리지 말라는 판단입니다.

<hr/>

마지막으로, 각 접근법의 **특징 비교**와 **실현 가능성**을 표로 요약합니다:

접근법	모델/튜닝 방식	장점 (기대효과)	난이도	추천도 및 비고
1. 도메인 미세튜닝 (LoRA)	Llama-2 7B/13B 등 + LoRA/QLoRA 튜닝 (4090 24GB에서 4bit로 13B까지 학습 가능) [6]	- 도메인 지식을 모델 내재화 - 출력 형식/일관성 향상 - baseline 대비 직접적 성능 향상 기대	중	**높음:** 주요 접근. 데이터 충분시 효과 확실.
2. RAG (검색 증강)	미튜닝 LLM + 벡터DB (Faiss 등) 구축 (외부 금융보안 문서 다수 필요)	- 최신/세부 지식 보완하여 정확도 향상 [11] - 환각 감소, 근거있는 답변 가능 [12] - 모델 튜닝 없이도 개선 (비용 효율)	중~상	**매우 높음:** 사실성 확보에 효과적. 준비노력 필요.
3. 프롬프트 튜닝/엔지니어링	Baseline LLM 그대로 활용 (시스템프롬프트, few-shot, soft prompt 등)	- 출력 포맷 개선, 실수 감소 - 데이터/자원 없이도 적용 가능 - 모델 성능 약간 향상 (특히 형식 면)	하~중	**중간:** 손쉬운 개선책. 단, 획기적 향상은 어려움.

접근법	모델/튜닝 방식	장점 (기대효과)	난이도	추천도 및 비고
4. Flan-T5 등 Enc-Dec 모델	Flan-T5 (Large~XXL) 등 + Full FT 또는 LoRA (객관식 강점 모델 활용) [22]	- 객관식 응답 정확도 및 형식 우수 [22] - 다양한 QA/NLI 학습으로 논리적 응답 - 비교적 작은 모델로도 효율적 튜닝	중	**중간:** 특정 상황에 유용. 기존 GPT계열 보완책.
5. 상위모델 활용 증강	GPT-4/API로 데이터 생성 → 우리 모델 튜닝 (지식 증류) [24] [8]	- 추가 데이터로 성능 점프 가능 - 교사 모델의 지식/스타일 학습 - baseline 넘는 **비밀무기** 될 수도	중	**조건부 높음:** 여유 자원/시간 있을 때 고려. 품질검수 병행 필수.

以上의 접근법들은 **상호 배타적이지 않으며**, 필요에 따라 **복합적으로 활용**할 수도 있습니다. 예를 들어 **접근법 1+2 결합**: 도메인 튜닝된 LLM에 RAG를 접목하면, 각각의 장점이 더해져 시너지 효과를 낼 수 있습니다. 또한 접근법 3 (프롬프트 개선)은 어떤 전략과도 함께 적용 가능하므로, **모델 튜닝 후에도 반드시 정교한 프롬프트 설계**를 통해 최종 성능을 끌어올리는 것이 좋습니다.

결론적으로, **"baseline보다 소폭 우수한 성능"**이라는 현실적 목표를 이루기 위해서는, 위 제안한 접근들 중 **리소스와 역량에 맞는 방법**을 선택하고 착실히 구현하는 것이 중요합니다. **도메인 미세튜닝**과/또는 **RAG**는 상당한 개선을 가져올 것으로 예상되며, **프롬프트 최적화**와 **데이터 증강** 기법이 이를 뒷받침하여 몇 점이라도 더 얻도록 해줄 것입니다. 각 방법의 구현 난이도를 고려해 **우선순위를 정하고 단계적으로 도전**한다면, 해커톤 본선 진출이라는 목표에 한걸음 더 가까워질 수 있을 것입니다.

---

[1] [2] CLIcK: A Benchmark Dataset of Cultural and Linguistic Intelligence in Korean
https://arxiv.org/html/2403.06412v4

[3] [4] [5] [7] [9] LLM의 다양한 SFT 기법: Full Fine-Tuning, PEFT (LoRA, QLoRA) — 코딩의 숲
https://ariz1623.tistory.com/348

[6] Topic #14: 암호같은 DoRA, QLoRA, QDoRA, 이건 뭘까요?
https://turingpost.co.kr/p/dora-qlora-qdora

[8] [24] Stanford CRFM
https://crfm.stanford.edu/2023/03/13/alpaca.html

[10] [11] [13] [14] 한국어 Advanced RAG 구현: Hugging Face와 LangChain 활용한 쿡북 - Hugging Face Open-Source AI Cookbook
https://huggingface.co/learn/cookbook/ko/advanced_ko_rag

[12] RAG vs. 파인튜닝 :: 기업용 맞춤 LLM을 위한 선택 가이드
https://www.skelterlabs.com/blog/rag-vs-finetuning

[15] [16] [17] [18] [19] [20] [21] Prompt Engineering vs Prompt Tuning vs Fine Tuning
https://yumdata.tistory.com/406

[22] [23] Baseline 모델에 대한 분석 - DACON
https://dacon.io/forum/414284?page=1&dtype=tag&ptype&fType&category=forum