FSKE Track1 (금융보안 QA) QLoRA 파인튜닝 개선 브리프

0) 한 줄 요약

- **목표**: QLoRA로 `ax.4.0-light-7b`를 금융·보안 MCQA/주관식 혼합 QA에 맞게 미세조정.
- **이슈**: 초기 300 step 구간에서 CE loss가 1.77~1.85 사이로 정체.
- **전략**: 에폭 1은 기준선으로 끝까지 진행. 이후 **짧은 민감도 테스트(0.3 epoch)**로 반응 좋은 설정을 찾고, **조기중단** 달고 2→3→(필요시)5 epoch 확장.
- **핵심 변경**: (1) 대회식 내부 메트릭 추가(객관식 Accuracy, 주관식 0.6*Cos* + *0.4*Keyword Recall), (2) 스케줄러/러닝레이트·LoRA 드롭아웃 민감도 테스트, (3) 주관식 오버샘플링, (4) 출력 규율(숫자 강제/길이 제한) 주입.

---

1) 리포지토리 컨텍스트 (가정)

- 데이터: `/mnt/data/mcqa.csv` (Korean), 컬럼: `Question`, `Answer`
  - 객관식: `Answer`는 `"1"~"5"` 등 숫자 1토큰
  - 주관식: 짧은 서술(한두 문장)
- 노트북: `/mnt/data/finetune_qlora.ipynb`
- 베이스 모델: `ax.4.0-light-7b` (로컬 가중치), QLoRA(4bit, NF4, double quant)
- 환경: Python 3.10, CUDA 11.8, PyTorch 2.1, RTX 4090 24GB

---

2) 현재 상태 진단

- **손실 지표**: Trainer 기본 **토큰 단위 Cross-Entropy(CE)**
  - 객관식은 한두 토큰, 주관식은 수십 토큰 → 평균 CE는 주관식에 민감
  - 대회 점수(정확도/코사인/키워드재현율)와 **직결되지 않음**
- **프롬프트**:
  - 객관식: “정답 선택지 번호만 출력” + `답변:`
  - 주관식: “두 문장 이내 간결 답변” + `답변:`

→ 구조는 양호, 형식 규율을 더 강화 가능

- **QLoRA 설정**: r=16, alpha=32, dropout=0.05, bf16, grad checkpointing ON → 합리적

---

3) 목표 지표(내부 메트릭)

검증 단계에서 다음을 **compute_metrics**로 기록:

- **MC Accuracy**: `pred.strip() == label.strip() and label.isdigit()`
- **주관식 점수**: `0.6 * CosineSim(Emb(pred), Emb(label)) + 0.4 * KeywordRecall(pred,label)`
  - **임베딩**: 로컬 Sentence-BERT/KoSimCSE (멀티링구얼 가능)
  - **키워드**: 형태소 기반 명사/고유명사 추출 후 불용어 제거, recall = |∩|/|label_keywords|
- **Final**: `0.5*MC + 0.5*주관식`

> 검증 속도 위해 고정 서브셋(예: 2,000 샘플) 사용 권장.

---

4) 실행 계획 (개발 지시)

A. 에폭 1 마무리 (기준선 확정)

- [ ] `eval_strategy="steps"` (예: 250 step)로 내부 메트릭과 CE-loss 동시 로깅
- [ ] 체크포인트 **중간 2~3개**(예: step 800, 1200)와 **에폭 말** 저장
- [ ] `print_trainable_parameters()`로 LoRA 학습 파라미터 수 확인(실수 방지)

B. 0.3 epoch 민감도 테스트 (세트 3개)

세트별로 **0.3 epoch**만 돌려 final_score 반응을 비교. 기준 대비 **+1.5~2.0%p**면 합격.

1. **세트 A (LR↑)**
   - `lr=2e-4`, `scheduler=linear`(또는 constant), `warmup_ratio=0.03`
2. **세트 B (신호 강화)**
   - `lora_dropout=0.0` (기타 동일)
3. **세트 C (데이터 균형화)**
   - 주관식 샘플 **가중치 3×**(WeightedRandomSampler)로 오버샘플

- 로더에서 타입 구분해 가중치 부여

C. 본 학습 확장(2→3 epoch, 조기중단)

☐ 민감도 테스트 **최고 조합**으로 2~3 epoch

☐ **조기중단**: 검증 2회 연속 개선 없음 → 중단

D. 출력 규율 주입(형식 안정화)

☐ **MC 디코딩 강제**: 생성 결과가 `^[1-5]$`(또는 `[1-4]`) 아니면, 로짓에서 해당 숫자 중 argmax로 강제 매핑

☐ **주관식 길이 제한**: 프롬프트에 "두 문장 이내" 명시 + 생성시 max_new_tokens 제한 (예: 64)

☐ 검증 메트릭에 **형식 위반률**(MC 숫자 외 출력, 주관식 과다 길이) 기록

E. 미니-오버핏 체크(디버깅)

☐ 객관식 5k + 주관식 2k **서브셋만**으로 0.2 epoch

- CE가 확실히 ↓면 세팅 정상
- **안 내려가면**: 레이블 정제(공백/제어문자), 토크나이저 매핑, label_smoothing=0 확인

F. (선택) 커리큘럼

☐ 0.5 epoch(객관식만) → 0.5 epoch(주관식만) → 혼합 1~2 epoch

G. (최후) LoRA 용량 상향

☐ `r=32`, `alpha=64`, `dropout=0` (메모리 허용 시)

☐ 반응(기울기) 보일 때만 적용

---

5) 코드 변경 스니펫 (핵심만)

5.1 내부 메트릭 (compute_metrics)

```
# prerequisites:
# tokenizer: same as training tokenizer
# emb_model: local sentence embedding model (e.g., sentence-transformers multi-lingual)
```

```
# extract_keywords: returns set of keywords from a Korean sentence

from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import re

digit_pat = re.compile(r"^\s*[1-9]\s*$")  # adjust range to [1-4] or [1-5] as n
eeded

def compute_metrics(eval_preds):
    preds, labels = eval_preds
    # Some trainers pass logits; adapt if needed:
    if isinstance(preds, tuple):
        preds = preds[0]
    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=Tr
ue)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=Tr
ue)

    mc_acc_list, subj_cos_list, subj_kw_list = [], [], []
    mc_format_violation, subj_len_violation = 0, 0

    for pred, gold in zip(decoded_preds, decoded_labels):
        p = pred.strip()
        g = gold.strip()

        # MC vs Subjective split (gold numeric → MC)
        if g.isdigit():
            # metric
            mc_acc_list.append(int(p == g))
            # format check (only a single digit)
            if not digit_pat.match(p):
                mc_format_violation += 1
        else:
            # embeddings
            ep = emb_model.encode([p])[0]
            eg = emb_model.encode([g])[0]
            cos = float(cosine_similarity([ep], [eg])[0][0])
```

```
        subj_cos_list.append(cos)

        # keyword recall
        k_g = extract_keywords(g)
        k_p = extract_keywords(p)
        if len(k_g) > 0:
            subj_kw_list.append(len(k_g & k_p) / len(k_g))
        else:
            subj_kw_list.append(0.0)
        # length rule (rough check)
        if p.count("다.") + p.count(".") > 2 or len(p.split()) > 80:
            subj_len_violation += 1

mc_acc = float(np.mean(mc_acc_list)) if mc_acc_list else 0.0
subj_cos = float(np.mean(subj_cos_list)) if subj_cos_list else 0.0
subj_kw  = float(np.mean(subj_kw_list)) if subj_kw_list else 0.0
subj_score = 0.6 * subj_cos + 0.4 * subj_kw
final = 0.5 * mc_acc + 0.5 * subj_score

# format violations as rates
n_mc = len(mc_acc_list)
n_subj = len(subj_cos_list)
mc_fmt_rate = mc_format_violation / n_mc if n_mc else 0.0
subj_len_rate = subj_len_violation / n_subj if n_subj else 0.0

return {
    "mc_accuracy": mc_acc,
    "subj_cos": subj_cos,
    "subj_kw": subj_kw,
    "subj_score": subj_score,
    "final_score": final,
    "mc_format_violation_rate": mc_fmt_rate,
    "subj_len_violation_rate": subj_len_rate,
}
```

> 속도 최적화: 검증 서브셋 고정(eval_dataset = eval_subset) 또는 임베딩 캐시.

5.2 스케줄러/러닝레이트(민감도 세트 A)

```
# When building training args (SFTConfig/TrainingArguments)
learning_rate = 2e-4
lr_scheduler_type = "linear"  # or "constant"
warmup_ratio = 0.03
```

5.3 LoRA 드롭아웃 0 (민감도 세트 B)

```
lora_config = LoraConfig(
    r=16, lora_alpha=32, lora_dropout=0.0, target_modules=..., bias="none",
task_type="CAUSAL_LM"
)
```

5.4 주관식 오버샘플링(민감도 세트 C)

```
# assume dataset has column "type" in {"mc", "subj"}; if not, derive from g
old label.isdigit()
from torch.utils.data import WeightedRandomSampler

weights = []
for ex in train_dataset:  # HF arrow dataset: map to list or create PyTorch da
taset wrapper
    is_subj = not ex["Answer"].strip().isdigit()
    weights.append(3.0 if is_subj else 1.0)

sampler = WeightedRandomSampler(weights, num_samples=len(weights),
replacement=True)
trainer = Trainer(..., train_dataset=train_dataset, data_collator=..., sampler=
sampler)
```

5.5 MC 디코딩 강제(인퍼런스 후처리)

```
def force_mc_digit(generated_text: str, allowed=("1","2","3","4","5")) → str:
    p = generated_text.strip()
    if p in allowed and len(p) == 1:
        return p
```

```
    # fallback: pick most probable among allowed via logits if available; else
regex extract first digit
    m = re.search(r"[1-5]", p)
    return m.group(0) if m else allowed[0]
```

5.6 주관식 길이 제한

- 프롬프트: **"두 문장 이내로 간결하게 답변"**
- 생성 파라미터: `max_new_tokens=64`, `no_repeat_ngram_size=3`, `temperature=0.7`(또는 0.2~0.7 그리드)

---

6) 품질 게이트(Go/No-Go)

- **Go(확장)**: 기준선 대비
  - MC Accuracy **+1.5~2.0%p↑** 또는
  - 주관식 `0.6*cos + 0.4*recall` **+1.5~2.0%p↑** 또는
  - 형식 위반률(MC/주관식) **유의미 감소**
- **No-Go**: 위 중 아무것도 충족 못하면, 민감도 세트 조합 변경 또는 커리큘럼 시도 → 그래도 무반응이면 LoRA 용량 상향 검토

---

7) 제출 대비(추론 파이프라인 권장)

- 오프라인 환경/단일 LLM 제약 준수
- **inference.py** 분리: 데이터 로드→전처리→모델 로드→추론→후처리(MC 강제/길이 제한)→CSV 출력
- 재현성: requirements.txt, 상대경로, UTF-8, 고정 시드, 270분 내 완주

---

8) 체크리스트 (Codex 액션 아이템)

- [ ] `compute_metrics` 구현 및 검증 서브셋 구성(2,000 샘플 고정)
- [ ] 민감도 세트 A/B/C 런 스크립트 3종 (`run_setA.sh`, `run_setB.sh`, `run_setC.sh`)
- [ ] Weighted sampler 적용 데이터로더(주관식 3×)
- [ ] MC 강제 후처리/주관식 길이 제한 유틸
- [ ] 미니-오버핏 스크립트(객관식5k+주관식2k)

- [ ] 로깅: loss·final_score·형식위반률, 중간 체크포인트 비교 리포트 생성
- [ ] (선택) 커리큘럼 학습 파이프라인
- [ ] (선택) LoRA r=32 실험 스크립트(메모리 여유 시)

---

9) 기대 비용/시간 (대략)

- 에폭 1: ~$5 (기준선)
- 민감도 3×0.3 epoch: ~$3–$6
- 본학습 2–3 epoch: ~$10–$15

  → 총 ~$18–$26 범위
