Part 1. Wrap Up

1-1. 개요

- 개발 개요

이미지와 4지선다로 결합되어 있는 데이터를 활용하여 지도학습을 진행하고, 새로운 데이터에서의 성능을 최대화하는 것을 목표로 함. ChatGPT를 활용하여, 성능을 최대화할 수 있는 하이퍼파라미터 수치 와 모델을 추천 받고, 개선한 코드로 Submission 을 진행하여 모델의 학습 성능을 최대화하고자 함. 최종적으로 0.94135 라는 높은 성능으로 마무리함.

1-2. 수행 절차 및 방법

- 개발 기회

기간 : 2025.10.23.목 14:00 ~ 2025.10.27.월 11:00
팀원 : 한상민(1441020), 김민수(1445638), 황가연(1440914), 양새하(1444879)
절차 :
- 2025.10.23.목 : AI 챌린지 OT, 팀 정하기
- 2025.10.24.금 : AI 챌린지 시작, Colab / SSH 적응 — 개인기록 0.86625, 팀기록 0.87860 달성
- 2025.10.25.토 : (11:00~19:00) 모델 7B로 변경 — 팀기록 0.94135 갱신
- 2025.10.26.일 : (11:00~19:00) 프롬프트 추가 변경 — 팀기록 갱신 실패
- 2025.10.27.월 : AI 챌린지 종료, 발표

submission12.csv Complete · A044_한상민_1441020 · 4d ago	**0.93106**	☐
12-submission.csv Complete · A044_황가연_1440914 · 4d ago	**0.26183**	☐
11-submission.csv Complete · A044_황가연_1440914 · 5d ago	**0.94135**	☐
10-submission.csv Complete · A044_양새하_1444879 · 6d ago	**0.86625**	☐
submission.csv Complete · A044_황가연_1440914 · 6d ago	**0.87860**	☐
8 - submission.csv Complete · A044_한상민_1441020 · 6d ago	**0.81224**	☐
7 - submission.csv Complete · A044_김민수_1445638 · 6d ago	**0.86162**	☐
5-submission(5e-3).csv Complete · A044_양새하_1444879 · 6d ago · 5e-3	**0.24176**	☐
5- submission.csv Complete · A044_한상민_1441020 · 6d ago	**0.79938**	☐
4 - submission.csv Complete · A044_김민수_1445638 · 6d ago	**0.86162**	☐
submission.csv Complete · A044_한상민_1441020 · 6d ago	**0.81224**	☐
submission.csv Complete · A044_김민수_1445638 · 6d ago	**0.76234**	☐
submission.csv Complete · A044_한상민_1441020 · 6d ago	**0.76028**	☐

1-3. 개발 수행 결과

- 사용한 ChatGPT 프롬프트

> AI 모델을 학습시켜서 성능을 향상시키고 싶어. 바꿀 수 있는 파라미터를 알려주고, 어떤 파라미터를 어떻게 바꿔야 할지 알려줘. 더 좋은 결과를 위해 프롬프트도 수정해도 돼. 대신 변경 전과 후를 주석으로 표시해줘

- 첫번째 수정사항

1) 프롬프트(시스템/유저) 강화

```
# BEFORE
SYSTEM_INSTRUCT = (
    "You are a helpful visual question answering assistant. "
    "Answer using exactly one letter among a, b, c, or d. No explanation."
)

def build_mc_prompt(question, a, b, c, d):
    return (
        f"{question}\n"
        f"(a) {a}\n(b) {b}\n(c) {c}\n(d) {d}\n\n"
        "정답을 반드시 a, b, c, d 중 하나의 소문자 한 글자로만 출력하세요."
    )

# AFTER
SYSTEM_INSTRUCT = (
    "You are a helpful visual QA assistant for multiple-choice. "
    "Reply with exactly ONE lowercase letter: a, b, c, or d. "
    "No punctuation, no newline, no extra text. If unsure, guess."
)

def build_mc_prompt(question, a, b, c, d):
    return (
        f"{question}\n"
        f"(a) {a}\n(b) {b}\n(c) {c}\n(d) {d}\n\n"
        "정답은 반드시 a/b/c/d 중 소문자 1글자만, 마침표/개행 없이 출력하세요."
    )
```

2) 이미지 해상도(선택)

```
# BEFORE
IMAGE_SIZE = 384

# AFTER  (GPU 여유 있으면 권장, 부족하면 384 유지)
IMAGE_SIZE = 448
```

3) LoRA 하이퍼파라미터 튜

```
# BEFORE
lora_config = LoraConfig(
    r=32,           # 8
    lora_alpha=64,    # 16
    lora_dropout=0.05,# 0.05
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM",
)

# AFTER (과적합/발산 줄이면서 안정적으로 수렴)
lora_config = LoraConfig(
    r=16,
    lora_alpha=64,
    lora_dropout=0.10,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM",
)
```

4) 라벨 마스킹: assistant 정답만 학습

```
# BEFORE (prompt 전체를 따라치도록 학습되어 성능 저하)
if self.train:
    enc["labels"] = enc["input_ids"].clone()

# AFTER (assistant 구간만 손실 계산)
if self.train:
    # full(assistant 포함) 렌더
    full_texts = texts
    # no-assistant(정답 제거) 렌더: 마지막 assistant turn 제거
    no_assist_texts = []
    for msgs in [s["messages"] for s in batch]:
        no_assist = msgs[:-1]  # 마지막 assistant gold 제거
        t = self.processor.apply_chat_template(no_assist, tokenize=False,
add_generation_prompt=False)
        no_assist_texts.append(t)

    full_ids = self.processor(text=full_texts, images=images, padding=True,
return_tensors="pt")
    no_assist_ids = self.processor(text=no_assist_texts, images=images, padding=True,
return_tensors="pt")

    enc = full_ids  # enc를 full로 교체
    enc["labels"] = enc["input_ids"].clone()

    # no-assistant 길이 이전은 모두 마스킹(-100)
    for i in range(enc["labels"].size(0)):
        cutoff = (no_assist_ids["input_ids"][i] != self.processor.tokenizer.pad_token_id).sum()
        enc["labels"][i, :cutoff] = -100
```

5) 데이터로더/학습 파라미터

```
# BEFORE
train_loader = DataLoader(train_ds, batch_size=1, shuffle=True,
collate_fn=DataCollator(processor, True), num_workers=0)
valid_loader = DataLoader(valid_ds, batch_size=1, shuffle=False,
collate_fn=DataCollator(processor, True), num_workers=0)
GRAD_ACCUM = 4
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-4)  # 1e-4
num_training_steps = 1 * math.ceil(len(train_loader)/GRAD_ACCUM)
scheduler = get_linear_schedule_with_warmup(optimizer, int(num_training_steps*0.03),
num_training_steps)
scaler = torch.cuda.amp.GradScaler(enabled=True)

# AFTER
train_loader = DataLoader(
    train_ds, batch_size=2, shuffle=True,
    collate_fn=DataCollator(processor, True),
    num_workers=2, pin_memory=True
)
valid_loader = DataLoader(
    valid_ds, batch_size=2, shuffle=False,
    collate_fn=DataCollator(processor, True),
    num_workers=2, pin_memory=True
)
GRAD_ACCUM = 8  # 유효 배치 사이즈 ↑ (2*8=16)

# LoRA 학습은 보통 작은 LR 선호
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-4, weight_decay=0.01,
betas=(0.9, 0.999))

EPOCHS = 3
steps_per_epoch = math.ceil(len(train_loader)/GRAD_ACCUM)
num_training_steps = EPOCHS * steps_per_epoch
warmup_steps = int(num_training_steps * 0.06)
scheduler = get_linear_schedule_with_warmup(optimizer, warmup_steps,
num_training_steps)

# bfloat16 사용 시 GradScaler 무의미 → 끔
scaler = torch.cuda.amp.GradScaler(enabled=False)

# 메모리/속도 최적
model.config.use_cache = False  # 학습 시 비활성 권장
```

6) 학습 루프(클립·로깅·평가 마스크 유지)

```
# BEFORE (핵심만)
with torch.cuda.amp.autocast(dtype=torch.bfloat16):
    outputs = model(**batch)
    loss = outputs.loss / GRAD_ACCUM
scaler.scale(loss).backward()
...
scaler.step(optimizer)
scaler.update()
```

```
# AFTER
for epoch in range(EPOCHS):
    running = 0.0
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1} [train]", unit="batch")
    for step, batch in enumerate(progress_bar, start=1):
        batch = {k:v.to(device) for k,v in batch.items()}
        with torch.cuda.amp.autocast(dtype=torch.bfloat16):
            outputs = model(**batch)
            loss = outputs.loss / GRAD_ACCUM

        loss.backward()
        running += loss.item()

        if step % GRAD_ACCUM == 0:
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)  # 그라디언트 클립
            optimizer.step()
            optimizer.zero_grad(set_to_none=True)
            scheduler.step()

            avg_loss = running
            progress_bar.set_postfix({"loss": f"{avg_loss:.3f}"})
            running = 0.0

    # VALID (라벨 마스킹 유지)
    model.eval()
    val_loss = 0.0
    with torch.no_grad(), torch.cuda.amp.autocast(dtype=torch.bfloat16):
        for vb in tqdm(valid_loader, desc=f"Epoch {epoch+1} [valid]", unit="batch"):
            vb = {k:v.to(device) for k,v in vb.items()}
            out = model(**vb)
            val_loss += out.loss.item()
    print(f"[Epoch {epoch+1}] valid loss {val_loss/len(valid_loader):.4f}")
    model.train()
```

7) 추론: 한 글자만 생성

```
# BEFORE
out_ids = model.generate(**inputs, max_new_tokens=2, do_sample=False,
                         eos_token_id=processor.tokenizer.eos_token_id)

# AFTER (한 글자만, 탐욕 생성)
out_ids = model.generate(
    **inputs,
    max_new_tokens=1,        # 한 글자
    do_sample=False,
    eos_token_id=processor.tokenizer.eos_token_id,
    pad_token_id=processor.tokenizer.pad_token_id
)
```

- 두번째 수정

1) 토크나이저 PAD/EOS 정리 (생성 안정화)

```
# BEFORE (없음)

# AFTER
# 토크나이저/모델의 pad_token 을 eos 로 맞춰서 패딩/생성 안정화
processor.tokenizer.pad_token = processor.tokenizer.eos_token
base_model.config.pad_token_id = processor.tokenizer.pad_token_id
base_model.config.eos_token_id = processor.tokenizer.eos_token_id
```

2) Stratified split (라벨 분포 유지)

```
# BEFORE
split = int(len(train_df)*0.9)
train_subset, valid_subset = train_df.iloc[:split], train_df.iloc[split:]

# AFTER
from sklearn.model_selection import train_test_split
train_subset, valid_subset = train_test_split(
    train_df, test_size=0.1, random_state=SEED, stratify=train_df["answer"]
)
train_subset = train_subset.reset_index(drop=True)
valid_subset = valid_subset.reset_index(drop=True)
```

3) 학습 데이터 200 샘플 제한 해제(성능↑) — GPU 여유 시

```
# BEFORE
train_df = train_df.sample(n=200, random_state=SEED).reset_index(drop=True) # 200

# AFTER
# 전체 데이터로 학습(메모리 빡세면 주석)
# train_df = train_df.sample(n=200, random_state=SEED).reset_index(drop=True)
```

4) 라벨 마스킹 로직 가볍게 정리(동일 동작, 메모리/속도 미세 개선)

```
# BEFORE (full/no_assist 두 번 인코딩)
full_ids = self.processor(text=full_texts, images=images, padding=True,
return_tensors="pt")
no_assist_ids = self.processor(text=no_assist_texts, images=images, padding=True,
return_tensors="pt")
enc = full_ids
enc["labels"] = enc["input_ids"].clone()
for i in range(enc["labels"].size(0)):
    cutoff = (no_assist_ids["input_ids"][i] != self.processor.tokenizer.pad_token_id).sum()
    enc["labels"][i, :cutoff] = -100

# AFTER (동일 의미, 변수명/주석만 간결화)
enc_full = self.processor(text=full_texts, images=images, padding=True,
return_tensors="pt")
enc_mask = self.processor(text=no_assist_texts, images=images, padding=True,
return_tensors="pt")
enc = enc_full
enc["labels"] = enc_full["input_ids"].clone()
pad_id = self.processor.tokenizer.pad_token_id
for i in range(enc["labels"].size(0)):
```

```
    cutoff = (enc_mask["input_ids"][i] != pad_id).sum()
    enc["labels"][i, :cutoff] = -100
```

5) 스케줄러: Cosine + 더 부드러운 워밍업 (수렴 안정화)

```
# BEFORE
from transformers import get_linear_schedule_with_warmup
scheduler = get_linear_schedule_with_warmup(optimizer, warmup_steps,
num_training_steps)

# AFTER
from transformers import get_cosine_schedule_with_warmup
scheduler = get_cosine_schedule_with_warmup(optimizer,
num_warmup_steps=warmup_steps, num_training_steps=num_training_steps)
```

6) GradScaler 제거에 맞춰 backward/step 정리 (깔끔/안전)

```
# BEFORE
with torch.cuda.amp.autocast(dtype=torch.bfloat16):
    outputs = model(**batch)
    loss = outputs.loss / GRAD_ACCUM
scaler.scale(loss).backward()   # enabled=False 이라 동작은 하지만 불필요
...
torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
optimizer.step()

# AFTER
with torch.cuda.amp.autocast(dtype=torch.bfloat16):
    outputs = model(**batch)
    loss = outputs.loss / GRAD_ACCUM
loss.backward()
torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
optimizer.step()
```

7) 검증 정확도(정답 한 글자) 측정 추가

```
# BEFORE (loss만 출력)

# AFTER
def decode_gold_letters(labels, tokenizer):
    gold = []
    for i in range(labels.size(0)):
        idx = (labels[i] != -100).nonzero(as_tuple=True)[0]
        last_id = labels[i, idx[-1]].item()
        gold.append(tokenizer.decode([last_id]).strip().lower())
    return gold

correct = 0; total = 0
with torch.no_grad(), torch.cuda.amp.autocast(dtype=torch.bfloat16):
    for vb in tqdm(valid_loader, desc=f"Epoch {epoch+1} [valid]", unit="batch"):
        vb = {k:v.to(device) for k,v in vb.items()}
        out = model(**vb)
        val_loss += out.loss.item()
```

```
        gen_ids = model.generate(
            input_ids=vb["input_ids"], pixel_values=vb["pixel_values"],
            attention_mask=vb["attention_mask"],
            max_new_tokens=1, do_sample=False,
            eos_token_id=processor.tokenizer.eos_token_id,
            pad_token_id=processor.tokenizer.pad_token_id
        )
        txt = processor.batch_decode(gen_ids, skip_special_tokens=True)
        pred = [extract_choice(t) for t in txt]
        gold = decode_gold_letters(vb["labels"].cpu(), processor.tokenizer)
        correct += sum(p == g for p, g in zip(pred, gold)); total += len(gold)

val_acc = correct / max(1, total)
print(f"[Epoch {epoch+1}] valid loss {val_loss/val_steps:.4f}, acc {val_acc:.4f}")
```

8) 디코딩 제약(잡문자 차단) + MAX_NEW_TOKENS 일관성

```
# BEFORE
out_ids = model.generate(**inputs, max_new_tokens=2, do_sample=False,
                         eos_token_id=processor.tokenizer.eos_token_id,
                         pad_token_id=processor.tokenizer.pad_token_id)

# AFTER
# a/b/c/d 외 단일 알파벳을 막아 한 글자 정답에 집중
bad = []
vocab = processor.tokenizer.get_vocab()
allow = set(processor.tokenizer.convert_tokens_to_ids(list("abcd")))
for tok, tid in vocab.items():
    if len(tok) == 1 and tok.isalpha() and tid not in allow:
        bad.append([tid])

out_ids = model.generate(
    **inputs,
    max_new_tokens=MAX_NEW_TOKENS,      # ← 상단 상수와 일관
    do_sample=False,
    eos_token_id=processor.tokenizer.eos_token_id,
    pad_token_id=processor.tokenizer.pad_token_id,
    bad_words_ids=bad                   # 선택: 모델/토크나이저에 따라 무시될 수도 있음
)
```

- 세번째 수정

1) 모델 클래스 교체(경고 제거 + 호환성↑)

```
# BEFORE
from transformers import (
    AutoModelForVision2Seq,
    AutoProcessor,
    BitsAndBytesConfig,
    get_linear_schedule_with_warmup
)
```

```
# AFTER
from transformers import (
    AutoModelForImageTextToText,  # ← 신형 클래스
    AutoProcessor,
    BitsAndBytesConfig,
    get_linear_schedule_with_warmup
)

# BEFORE
base_model = AutoModelForVision2Seq.from_pretrained(
    MODEL_ID,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True,
)

# AFTER
base_model = AutoModelForImageTextToText.from_pretrained(
    MODEL_ID,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True,
)
```

2) 토크나이저 PAD/EOS 고정(생성/패딩 안정화)

```
# BEFORE (없음)
# AFTER
processor.tokenizer.pad_token = processor.tokenizer.eos_token
base_model.config.pad_token_id = processor.tokenizer.pad_token_id
base_model.config.eos_token_id = processor.tokenizer.eos_token_id
```

3) LoRA 표준 튠(5epoch 안정 수렴)

```
# BEFORE
lora_config = LoraConfig(
    r=32,           # 32는 빠르게 맞지만 5epoch 기준 과적합/발산 가능성↑
    lora_alpha=64,
    lora_dropout=0.10,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM",
)

# AFTER  # underfit이면 r=24로만 올려보면 좋음
lora_config = LoraConfig(
    r=16,               # ← 일반적으로 가장 안정 (필요시 24)
    lora_alpha=64,
    lora_dropout=0.10,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","up_proj","down_proj"],  # gate_proj
제외로 미세 안정화
    task_type="CAUSAL_LM",
```

```
)
```

4) 학습 하이퍼파라미터(5epoch 최적화)

```
# BEFORE
optimizer = torch.optim.AdamW(model.parameters(), lr=2.5e-4, weight_decay=0.01,
betas=(0.9, 0.999))
EPOCHS = 5
warmup_steps = int(num_training_steps * 0.1)
from transformers import get_cosine_schedule_with_warmup
scheduler = get_cosine_schedule_with_warmup(optimizer,
num_warmup_steps=warmup_steps, num_training_steps=num_training_steps)

# AFTER
# 소규모 LoRA는 대체로 1e-4 ~ 2e-4가 가장 안전. 5epoch 기준 과적합 줄이려면 wd 약간
↑
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-4, weight_decay=0.05,
betas=(0.9, 0.999))
EPOCHS = 5
warmup_steps = int(num_training_steps * 0.08)  # 8%로 약간 보수
from transformers import get_cosine_schedule_with_warmup
scheduler = get_cosine_schedule_with_warmup(optimizer,
num_warmup_steps=warmup_steps, num_training_steps=num_training_steps)
```

5) Early Stopping + Best Checkpoint 저장(5epoch 내 성능 극대화)

```
# BEFORE (없음)

# AFTER
best_val = float("inf")
best_path = f"{SAVE_DIR}_best"

...
for epoch in range(EPOCHS):
    ...
    # === VALID ===
    model.eval()
    val_loss = 0.0
    with torch.no_grad(), torch.cuda.amp.autocast(dtype=torch.bfloat16):
        for vb in tqdm(valid_loader, desc=f"Epoch {epoch+1} [valid]", unit="batch"):
            vb = {k:v.to(device) for k,v in vb.items()}
            val_loss += model(**vb).loss.item()
    val_loss /= len(valid_loader)
    print(f"[Epoch {epoch+1}] valid loss {val_loss:.4f}")

    # Early stopping & save best
    if val_loss < best_val - 1e-4:  # 작은 개선도 인정
        best_val = val_loss
        model.save_pretrained(best_path)      # ← best만 별도 저장
        processor.save_pretrained(best_path)
        patience = 0
    else:
        patience = patience + 1 if 'patience' in locals() else 1
```

```
        if patience >= 2:   # 2연속 개선 없음 → 조기 종료 (5epoch 내 효율 MAX)
            print("Early stopping triggered.")
            break
    model.train()
```

6) 검증 정확도 함께 모니터(손실만 보는 오류 방지)

```
# BEFORE (loss만 측정)

# AFTER
def _decode_gold_letter(labels, tokenizer):
    gold = []
    for i in range(labels.size(0)):
        idx = (labels[i] != -100).nonzero(as_tuple=True)[0]
        last_id = labels[i, idx[-1]].item()
        gold.append(tokenizer.decode([last_id]).strip().lower())
    return gold

...
correct = 0; total = 0
with torch.no_grad(), torch.cuda.amp.autocast(dtype=torch.bfloat16):
    for vb in tqdm(valid_loader, desc=f"Epoch {epoch+1} [valid]", unit="batch"):
        vb = {k:v.to(device) for k,v in vb.items()}
        out = model(**vb)
        val_loss += out.loss.item()

        gen_ids = model.generate(
            input_ids=vb["input_ids"], pixel_values=vb["pixel_values"],
            attention_mask=vb["attention_mask"],
            max_new_tokens=1, do_sample=False,
            eos_token_id=processor.tokenizer.eos_token_id,
            pad_token_id=processor.tokenizer.pad_token_id
        )
        txt = processor.batch_decode(gen_ids, skip_special_tokens=True)
        pred = [extract_choice(t) for t in txt]
        gold = _decode_gold_letter(vb["labels"].cpu(), processor.tokenizer)
        correct += sum(p == g for p, g in zip(pred, gold)); total += len(gold)

val_acc = correct / max(1,total)
print(f"[Epoch {epoch+1}] valid loss {val_loss/len(valid_loader):.4f}, acc {val_acc:.4f}")
```

7) 프롬프트 미세수정(잡출력 억제)

```
# BEFORE
SYSTEM_INSTRUCT = (
    "You are a helpful visual QA assistant for multiple-choice. "
    "Reply with exactly ONE lowercase letter: a, b, c, or d. "
    "No punctuation, no newline, no extra text. If unsure, guess."
)

# AFTER
SYSTEM_INSTRUCT = (
    "You are a visual QA assistant for multiple-choice.\n"
```

```
    "- Output exactly ONE lowercase letter: a, b, c, or d\n"
    "- No punctuation, no space, no newline, no explanation\n"
    "- If unsure, guess one letter"
)
```

8) 생성 제약(비허용 알파벳 차단) — 1회 전처리로 비용↓

```
# BEFORE (매 샘플마다 bad_words_ids 구축)
bad = []
vocab = processor.tokenizer.get_vocab()
allow = set(processor.tokenizer.convert_tokens_to_ids(list("abcd")))
for tok, tid in vocab.items():
    if len(tok) == 1 and tok.isalpha() and tid not in allow:
        bad.append([tid])

# AFTER (학습 시작 전 1회만 만들어 재사용)
# 전역에서 한 번:
ALLOWED = set(processor.tokenizer.convert_tokens_to_ids(list("abcd")))
BAD_WORDS = [[tid] for tok, tid in processor.tokenizer.get_vocab().items()
             if len(tok) == 1 and tok.isalpha() and tid not in ALLOWED]

# 추론 루프에서는 그대로 사용:
out_ids = model.generate(
    **inputs,
    max_new_tokens=MAX_NEW_TOKENS,
    do_sample=False,
    eos_token_id=processor.tokenizer.eos_token_id,
    pad_token_id=processor.tokenizer.pad_token_id,
    bad_words_ids=BAD_WORDS
)
```

9) DataLoader I/O 튠(끊김 최소화)

```
# BEFORE
train_loader = DataLoader(..., num_workers=4, pin_memory=True,
persistent_workers=True, prefetch_factor=4)

# AFTER  # CPU 코어 여유면 살짝만 ↑
train_loader = DataLoader(
    train_ds, batch_size=2, shuffle=True,
    collate_fn=DataCollator(processor, True),
    num_workers=6, pin_memory=True,
    persistent_workers=True, prefetch_factor=6
)
valid_loader = DataLoader(
    valid_ds, batch_size=2, shuffle=False,
    collate_fn=DataCollator(processor, True),
    num_workers=6, pin_memory=True,
    persistent_workers=True, prefetch_factor=6
)
```

10) 재현성 강화(스코어 흔들림↓)

```
# BEFORE
random.seed(SEED); torch.manual_seed(SEED); torch.cuda.manual_seed_all(SEED)

# AFTER
random.seed(SEED); torch.manual_seed(SEED); torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True    # 재현성↑
torch.backends.cudnn.benchmark = False       # 성능 소폭↓ 대신 일관성↑
```

- 상위호환 모델 요청 프롬프트

```
아래 사이트 모델에서 qwen2_5_vl_3b_lora 보다 성능좋은 모델 알려주고, 적용할 수 있는 코드를 (전) (후) 를 주석으로 표현해서 알려줘. (전)이 없는경우, 위아래 코드를 보여줌으로써 (후) 코드를 복붙해서 넣을 위치를 알려줘

https://huggingface.co/spaces/opencompass/open_vlm_leaderboard
```

```
셋 중에 vs sssh 서버에서 돌아갈 수 있는 모델은 뭐야?
```

- 네번째 수정 : 모델 교체

```
# 모델/프로세서 로드 지점 패치

# (전)
MODEL_ID = "Qwen/Qwen2.5-VL-3B-Instruct"
base_model = AutoModelForImageTextToText.from_pretrained(
    MODEL_ID, quantization_config=bnb_config, device_map="auto",
trust_remote_code=True,
)
processor = AutoProcessor.from_pretrained(
    MODEL_ID, min_pixels=IMAGE_SIZE*IMAGE_SIZE,
max_pixels=IMAGE_SIZE*IMAGE_SIZE, trust_remote_code=True,
)

# (후)
MODEL_ID = "Qwen/Qwen2.5-VL-7B-Instruct"
base_model = AutoModelForImageTextToText.from_pretrained(
    MODEL_ID, quantization_config=bnb_config, device_map="auto",
trust_remote_code=True,
)
processor = AutoProcessor.from_pretrained(
    MODEL_ID, min_pixels=IMAGE_SIZE*IMAGE_SIZE,
max_pixels=IMAGE_SIZE*IMAGE_SIZE, trust_remote_code=True,
)
```

✅ 가능성 높은 모델

- Qwen2.5-VL-7B-Instruct
  - 리포지토리 토론에 "RTX 4090(24 GB VRAM)로도 잘 돌아간다"는 사용자 후기 있음. Hugging Face
  - "Minimum: NVIDIA GPU with 24 GB VRAM; RAM: at least 16 GB" 등의 설치 가이드 존재. GitHub +1
  - 따라서 **24 GB 이상 VRAM** 및 충분한 CPU/RAM이 있는 SSH 서버라면 충분히 돌릴 가능성 높아요.

⚠️ 가능성 낮거나 조건이 매우 빡센 모델

- LLaVA-OneVision-7B
  - "Inference process uses approximately 20 GB of GPU memory" 라는 언급도 있고, 어떤 세팅에선 "72 GB GPU 메모리 필요"라는 보고도 있어요. Hugging Face
  - 7B임에도 이미지/비디오 처리를 포함해서 메모리 요구가 높기 때문에 보통 **24 GB 이하 서버에서는 빡셀 수 있어요.**
- InternVL2-8B
  - 모델 카드·토론에서 정확한 VRAM 요구가 명시적이지 않지만 "full-LLM fine-tuning에는 다수의 32G/40G GPU 필요"라는 언급이 있어요. internvl.readthe... +1
  - 따라서 추론만 한다 해도 **24–32GB 이상 GPU + 여러 최적화**가 필요할 가능성 높음.

- 다섯번째 수정

① "a/b/c/d만" 내게 하는 강제 디코딩(LogitsProcessor)

```
# (전) 없음
# (후) ↓ import 근처 혹은 모델 로드 아래에 넣기
from transformers import LogitsProcessor

class OnlyABCDProcessor(LogitsProcessor):
    def __init__(self, tokenizer):
        self.allowed = set(tokenizer.convert_tokens_to_ids(list("abcd")))
    def __call__(self, input_ids, scores):
        # 1스텝만 한 글자 뽑으므로 전체 토큰에서 a/b/c/d 외엔 -inf
        mask = torch.full_like(scores, float("-inf"))
        idx = torch.tensor(list(self.allowed), device=scores.device)
        mask[:, idx] = scores[:, idx]
        return mask
```

```
# (전) 추론 generate
```

```
out_ids = model.generate(
    **inputs,
    max_new_tokens=MAX_NEW_TOKENS,
    do_sample=False,
    eos_token_id=processor.tokenizer.eos_token_id,
    pad_token_id=processor.tokenizer.pad_token_id,
    bad_words_ids=BAD_WORDS
)

# (후) OnlyABCDProcessor를 사용 (bad_words_ids 보다 강력·간단)
logits_proc = [OnlyABCDProcessor(processor.tokenizer)]
out_ids = model.generate(
    **inputs,
    max_new_tokens=MAX_NEW_TOKENS,
    do_sample=False,
    eos_token_id=processor.tokenizer.eos_token_id,
    pad_token_id=processor.tokenizer.pad_token_id,
    logits_processor=logits_proc
)
```

② 라벨 스무딩로 “헷갈리는 보기”에 덜 과적합 + 일반화↑

```
# (전) 학습 루프에서 모델의 loss 사용
with torch.cuda.amp.autocast(dtype=torch.bfloat16):
    outputs = model(**batch)
    loss = outputs.loss / GRAD_ACCUM

# (후) 스무딩 손실(ε=0.05)로 교체 — labels의 유효 위치(정답 한 글자)만 계산
LABEL_SMOOTHING = 0.05

with torch.cuda.amp.autocast(dtype=torch.bfloat16):
    outputs = model(**batch)
    logits = outputs.logits  # [B, T, V]
    labels = batch["labels"] # [B, T]
    # 유효 토큰(정답 1글자) 인덱스
    valid_mask = labels.ne(-100)
    # 마지막 유효 토큰 위치만 취함
    idxs = [m.nonzero(as_tuple=True)[0][-1].item() for m in valid_mask]
    bsz = labels.size(0)
    gather_logits = logits[torch.arange(bsz), idxs]              # [B, V]
    gold_ids = labels[torch.arange(bsz), idxs]                   # [B]
    logprobs = torch.log_softmax(gather_logits.float(), dim=-1)  # [B, V]

    # 라벨 스무딩된 타겟 분포
    V = logprobs.size(-1)
    smooth_val = LABEL_SMOOTHING / V
    target = torch.full_like(logprobs, smooth_val)
    target.scatter_(1, gold_ids.unsqueeze(1), 1.0 - LABEL_SMOOTHING)

    ce = -(target * logprobs).sum(dim=-1).mean()   # [B] → scalar
    loss = (ce / GRAD_ACCUM)
```

③ 선지 셔플(Augmentation) + 정답 재매핑

```
# (전) VQAMCDataset.__getitem__
q = str(row["question"])
a, b, c, d = str(row["a"]), str(row["b"]), str(row["c"]), str(row["d"])
user_text = build_mc_prompt(q, a, b, c, d)
...
if self.train:
    gold = str(row["answer"]).strip().lower()
    messages.append({"role":"assistant","content":[{"type":"text","text":gold}]})

# (후) 보기 셔플 + 정답 재매핑
q = str(row["question"])
choices = [('a', str(row["a"])), ('b', str(row["b"])), ('c', str(row["c"])), ('d', str(row["d"]))]
if self.train:
    random.shuffle(choices)  # 위치 섞기
label_orig = str(row["answer"]).strip().lower()
# 새 보기 순서에 맞춰 정답 문자 재계산
label_map = {k:i for i,(k,_) in enumerate(choices)}  # {'a':?, 'b':?, ...}
new_idx = label_map[label_orig]                      # 0..3
new_label = "abcd"[new_idx]

a, b, c, d = [t for _, t in choices]
user_text = build_mc_prompt(q, a, b, c, d)

messages = [
    {"role":"system","content":[{"type":"text","text":SYSTEM_INSTRUCT}]},
    {"role":"user","content":[
        {"type":"image","image":img},
        {"type":"text","text":user_text}
    ]}
]
if self.train:
    messages.append({"role":"assistant","content":[{"type":"text","text":new_label}]})
```

④ EMA(지수이동평균) 가중치로 과적합 방지 + 스코어 상향

```
# (전) 없음
# (후) ↓ 학습 시작 전에
ema_decay = 0.999
ema_state = {k: v.detach().clone() for k,v in model.state_dict().items() if
v.dtype.is_floating_point}

def ema_update(model, ema_state, decay):
    with torch.no_grad():
        for k, v in model.state_dict().items():
            if k in ema_state and v.dtype.is_floating_point:
                ema_state[k].mul_(decay).add_(v.detach(), alpha=1.0 - decay)
```

```
# (전) optimizer.step() 만 호출
optimizer.step()

# (후) step 직후 EMA 업데이트
optimizer.step()
```

```
ema_update(model, ema_state, ema_decay)
```

```
# (전) 검증/저장 시 그대로 model 사용
# (후) 검증/저장 직전에 EMA 가중치로 스왑 → 평가/저장 후 원복
def load_state(d, m): m.load_state_dict(d, strict=False)
backup = {k: v.detach().clone() for k,v in model.state_dict().items()}
# EMA 가중치 적용
load_state(ema_state, model)
# → 여기서 valid/저장 수행
# 원복
load_state(backup, model)
```

하이퍼파라미터 미세 조정(5 epoch 최적)

```
# (전) lr=2e-4
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-4, weight_decay=0.05,
betas=(0.9, 0.999))
# (후) lr=1.5e-4
optimizer = torch.optim.AdamW(model.parameters(), lr=1.5e-4, weight_decay=0.05,
betas=(0.9, 0.999))
```

```
# (전)
warmup_steps = int(num_training_steps * 0.08)
# (후)
warmup_steps = int(num_training_steps * 0.10)
```

프롬프트 미세 조정(노이즈 더 억제)

```
# (전)
SYSTEM_INSTRUCT = (
    "You are a visual QA assistant for multiple-choice.\n"
    "- Output exactly ONE lowercase letter: a, b, c, or d\n"
    "- No punctuation, no space, no newline, no explanation\n"
    "- If unsure, guess one letter"
)

# (후) 'Answer:' 토큰 앵커 추가(모델이 한 글자로 답하게 유도)
SYSTEM_INSTRUCT = (
    "You are a visual QA assistant for multiple-choice.\n"
    "- Output exactly ONE lowercase letter: a, b, c, or d\n"
    "- No punctuation/space/newline/explanation\n"
    "- Respond after the word 'Answer:' with a single letter"
)

def build_mc_prompt(question, a, b, c, d):
    return (
        f"{question}\n"
        f"(a) {a}\n(b) {b}\n(c) {c}\n(d) {d}\n\n"
        "Answer: "  # ← 앵커
    )
```

- 최고기록 Baseline

환경준비

```
!pip -q install "transformers>=4.44.2" "accelerate>=0.34.2" "peft>=0.13.2"
"bitsandbytes>=0.43.1" datasets pillow pandas torch torchvision --upgrade
```

```
!sudo chown -R team020:team020 /opt/hf-cache
```

데이터 준비

```
import os, re, math, random
import pandas as pd
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from dataclasses import dataclass
import torch
from typing import Dict, List, Any
from transformers import (
    AutoModelForVision2Seq,
    AutoProcessor,
    BitsAndBytesConfig,
    get_linear_schedule_with_warmup
)
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
from tqdm import tqdm

# 이미지 로드 시 픽셀 제한 해제
Image.MAX_IMAGE_PIXELS = None

# 디바이스 GPU 우선 사용 설정
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Device:", device)

# 사전 학습 모델 정의
MODEL_ID = "Qwen/Qwen2.5-VL-7B-Instruct"
IMAGE_SIZE = 448
MAX_NEW_TOKENS = 8
SEED = 42
random.seed(SEED); torch.manual_seed(SEED); torch.cuda.manual_seed_all(SEED)

# 데이터셋 로드
train_df = pd.read_csv("/home/team020/train.csv")
test_df  = pd.read_csv("/home/team020/test.csv")

# # 학습데이터 200개만 추출
# train_df = train_df = pd.read_csv("/content/train.csv")
```

모델, Processor

```
# ----------------------------
# 프로세서
# ----------------------------
processor = AutoProcessor.from_pretrained(
```

```
    MODEL_ID,
    min_pixels=IMAGE_SIZE*IMAGE_SIZE,
    max_pixels=IMAGE_SIZE*IMAGE_SIZE,
    trust_remote_code=True,
)

# ----------------------------
# 사전학습 모델 로드 (bf16 full precision)
# ----------------------------
base_model = AutoModelForVision2Seq.from_pretrained(
    MODEL_ID,
    torch_dtype=torch.bfloat16,   # bf16 full precision
    device_map="auto",
    trust_remote_code=True,
)

# Gradient checkpointing 켜서 VRAM 절약
base_model.gradient_checkpointing_enable()

# ----------------------------
# LoRA 세팅
# ----------------------------
lora_config = LoraConfig(
    r=32,
    lora_alpha=64,
    lora_dropout=0.05,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM",
)

# PEFT 모델 생성
model = get_peft_model(base_model, lora_config)
model.print_trainable_parameters()
```

프롬프트 템플릿

```
# 모델 지시사항
SYSTEM_INSTRUCT = (
    "You are a helpful visual question answering assistant. "
    "Answer using exactly one letter among a, b, c, or d. No explanation."
)

# 프롬프트
def build_mc_prompt(question, a, b, c, d):
    return (
        f"{question}\n"
        f"(a) {a}\n(b) {b}\n(c) {c}\n(d) {d}\n\n"
        "정답을 반드시 a, b, c, d 중 하나의 소문자 한 글자로만 출력하세요."
    )
```

Custom Dataset, Collator

```
# 커스텀 데이터셋
class VQAMCDataset(Dataset):
    def __init__(self, df, processor, train=True):
        self.df = df.reset_index(drop=True)
        self.processor = processor
        self.train = train

    def __len__(self): return len(self.df)

    def __getitem__(self, i):
        row = self.df.iloc[i]
        img = Image.open(row["path"]).convert("RGB")

        q = str(row["question"])
        a, b, c, d = str(row["a"]), str(row["b"]), str(row["c"]), str(row["d"])
        user_text = build_mc_prompt(q, a, b, c, d)

        messages = [
            {"role":"system","content":[{"type":"text","text":SYSTEM_INSTRUCT}]},
            {"role":"user","content":[
                {"type":"image","image":img},
                {"type":"text","text":user_text}
            ]}
        ]
        if self.train:
            gold = str(row["answer"]).strip().lower()
            messages.append({"role":"assistant","content":[{"type":"text","text":gold}]})

        return {"messages": messages, "image": img}

# 데이터 콜레이터
@dataclass
class DataCollator:
    processor: Any
    train: bool = True

    def __call__(self, batch):
        texts, images = [], []
        for sample in batch:
            messages = sample["messages"]
            img = sample["image"]

            text = self.processor.apply_chat_template(
                messages,
                tokenize=False,
                add_generation_prompt=False
            )
            texts.append(text)
            images.append(img)

        enc = self.processor(
            text=texts,
            images=images,
            padding=True,
```

```
            return_tensors="pt"
        )

        if self.train:
            enc["labels"] = enc["input_ids"].clone()

        return enc
```

DataLoader

```
# 검증용 데이터 분리
split = int(len(train_df)*0.9)
train_subset, valid_subset = train_df.iloc[:split], train_df.iloc[split:]

# VQAMCDataset 형태로 변환
train_ds = VQAMCDataset(train_subset, processor, train=True)
valid_ds = VQAMCDataset(valid_subset, processor, train=True)

# 데이터로더
train_loader = DataLoader(train_ds, batch_size=2, shuffle=True,
collate_fn=DataCollator(processor, True), num_workers=0)
valid_loader = DataLoader(valid_ds, batch_size=2, shuffle=False,
collate_fn=DataCollator(processor, True), num_workers=0)
```

Fine-tuning

```
from tqdm.auto import tqdm

model = model.to(device)
GRAD_ACCUM = 4

# 옵티마이저, 학습 스케줄러
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-4)
num_training_steps = 5 * math.ceil(len(train_loader)/GRAD_ACCUM)
scheduler = get_linear_schedule_with_warmup(optimizer, int(num_training_steps*0.03),
num_training_steps)

# 스케일러
scaler = torch.cuda.amp.GradScaler(enabled=True)

# 학습 루프
global_step = 0
for epoch in range(5):
    running = 0.0
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1} [train]", unit="batch")
    for step, batch in enumerate(progress_bar, start=1):
        batch = {k:v.to(device) for k,v in batch.items()}
        with torch.cuda.amp.autocast(dtype=torch.bfloat16):
            outputs = model(**batch)
            loss = outputs.loss / GRAD_ACCUM

        scaler.scale(loss).backward()
        running += loss.item()
```

```
        if step % GRAD_ACCUM == 0:
            scaler.step(optimizer)
            scaler.update()
            optimizer.zero_grad(set_to_none=True)
            scheduler.step()
            global_step += 1

            avg_loss = running / GRAD_ACCUM
            progress_bar.set_postfix({"loss": f"{avg_loss:.3f}"})
            running = 0.0

    model.eval()
    val_loss = 0.0
    val_steps = 0
    with torch.no_grad(), torch.cuda.amp.autocast(dtype=torch.bfloat16):
        for vb in tqdm(valid_loader, desc=f"Epoch {epoch+1} [valid]", unit="batch"):
            vb = {k:v.to(device) for k,v in vb.items()}
            val_loss += model(**vb).loss.item()
            val_steps += 1
    print(f"[Epoch {epoch+1}] valid loss {val_loss/val_steps:.4f}")
    model.train()

# 모델 저장
SAVE_DIR = "/home/team020/qwen2_5_vl_7b_lora"
model.save_pretrained(SAVE_DIR)
processor.save_pretrained(SAVE_DIR)
print("Saved:", SAVE_DIR)
```

Inference

```
# 데이터 파서 : 모델의 응답에서 선지를 추출
def extract_choice(text: str) -> str:
    text = text.strip().lower()

    lines = [l.strip() for l in text.splitlines() if l.strip()]
    if not lines:
        return "a"
    last = lines[-1]
    if last in ["a", "b", "c", "d"]:
        return last

    tokens = last.split()
    for tok in tokens:
        if tok in ["a", "b", "c", "d"]:
            return tok
    return "a"

# 추론을 위해 모든 레이어 활성화
model.eval()
preds = []

# 추론 루프
```

```
for i in tqdm(range(len(test_df)), desc="Inference", unit="sample"):
    row = test_df.iloc[i]
    img = Image.open(row["path"]).convert("RGB")
    user_text = build_mc_prompt(row["question"], row["a"], row["b"], row["c"], row["d"])

    messages = [
        {"role":"system","content":[{"type":"text","text":SYSTEM_INSTRUCT}]},
        {"role":"user","content":[
            {"type":"image","image":img},
            {"type":"text","text":user_text}
        ]}
    ]

    text = processor.apply_chat_template(messages, tokenize=False,
add_generation_prompt=True)
    inputs = processor(text=[text], images=[img], return_tensors="pt").to(device)

    with torch.no_grad():
        out_ids = model.generate(**inputs, max_new_tokens=2, do_sample=False,
                                 eos_token_id=processor.tokenizer.eos_token_id)
    output_text = processor.batch_decode(out_ids, skip_special_tokens=True)[0]
    # print("output_text:", output_text)
    # print("extract_choice:", extract_choice(output_text))
    preds.append(extract_choice(output_text))

# 제출 파일 생성
submission = pd.DataFrame({"id": test_df["id"], "answer": preds})
submission.to_csv("/home/team020/submission.csv", index=False)
print("Saved /home/team020/submission.csv")
```

- 발표자료

SSAFY AI 챌린지

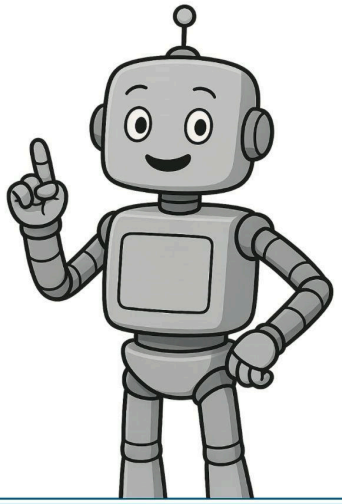
상미나이

김민수 양새하 한상민 황가연

CONTENTS

submissions

submission12.csv Complete · A044_한상민_3441020 · 15h ago	0.93106
12-submission.csv Complete · A044_황가연_3440914 · 16h ago	0.26183
11-submission.csv Complete · A044_황가연_3440914 · 2d ago	0.94135
10-submission.csv Complete · A044_생석찌_3444879 · 3d ago	0.86625
submission.csv Complete · A044_황가연_3440914 · 3d ago	0.87860
8 - submission.csv Complete · A044_한상민_3441020 · 3d ago	0.81224
7 - submission.csv Complete · A044_김민수_3445638 · 3d ago	0.86162
5-submission(5e-3).csv Complete · A044_생석찌_3444879 · 3d ago · 5e-3	0.24176
5- submission.csv Complete · A044_한상민_3441020 · 3d ago	0.79938
4 - submission.csv Complete · A044_김민수_3445638 · 3d ago	0.86162
submission.csv Complete · A044_한상민_3441020 · 3d ago	0.81224
submission.csv Complete · A044_김민수_3445638 · 3d ago	0.76234
submission.csv Complete · A044_한상민_3441020 · 3d ago	0.76028



변경 사항

- 사전 학습 모델 정의
- 모델, processor
- 데이터 로더
- fine-tuning

사전 학습 모델 정의

before

```python
# 사전 학습 모델 정의
MODEL_ID = "Qwen/Qwen2.5-VL-3B-Instruct"
IMAGE_SIZE = 384
MAX_NEW_TOKENS = 8
SEED = 42
random.seed(SEED); torch.manual_seed(SEED); torch.cuda.manual_seed_all(SEED)

# 데이터셋 로드
train_df = pd.read_csv("/content/train.csv")
test_df  = pd.read_csv("/content/test.csv")

# 학습데이터 200개만 추출
train_df = train_df.sample(n=200, random_state=SEED).reset_index(drop=True)
```

after

```python
# 사전 학습 모델 정의
MODEL_ID = "Qwen/Qwen2.5-VL-7B-Instruct"
IMAGE_SIZE = 448
MAX_NEW_TOKENS = 8
SEED = 42
random.seed(SEED); torch.manual_seed(SEED); torch.cuda.manual_seed_all(SEED)

# 데이터셋 로드
train_df = pd.read_csv("/home/team020/train.csv")
test_df  = pd.read_csv("/home/team020/test.csv")
```

Qwen/Qwen2.5-VL-3B-Instruct

↓

Qwen/Qwen2.5-VL-**7B**-Instruct

IMAGE_SIZE = 384

↓

IMAGE_SIZE = **448**

학습 데이터 개수 : 200개

↓

학습 데이터 개수 : **전체**

모델, processor

before

```python
# 양자화
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
)

# 프로세서
processor = AutoProcessor.from_pretrained(
    MODEL_ID,
    min_pixels=IMAGE_SIZE*IMAGE_SIZE,
    max_pixels=IMAGE_SIZE*IMAGE_SIZE,
    trust_remote_code=True,
)

# 사전학습 모델
base_model = AutoModelForVision2Seq.from_pretrained(
    MODEL_ID,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True,
)

# 양자화 모델로 로드
base_model = prepare_model_for_kbit_training(base_model)
base_model.gradient_checkpointing_enable()

# LoRA 세팅
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM",
)
```

after

```python
# ----------------------------
# 프로세서
# ----------------------------
processor = AutoProcessor.from_pretrained(
    MODEL_ID,
    min_pixels=IMAGE_SIZE*IMAGE_SIZE,
    max_pixels=IMAGE_SIZE*IMAGE_SIZE,
    trust_remote_code=True,
)

# ----------------------------
# 사전학습 모델 로드 (bf16 full precision)
# ----------------------------
base_model = AutoModelForVision2Seq.from_pretrained(
    MODEL_ID,
    torch_dtype=torch.bfloat16,   # bf16 full precision
    device_map="auto",
    trust_remote_code=True,
)

# Gradient checkpointing 켜서 VRAM 절약
base_model.gradient_checkpointing_enable()

# ----------------------------
# LoRA 세팅
# ----------------------------
lora_config = LoraConfig(
    r=32,
    lora_alpha=64,
    lora_dropout=0.05,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM",
)
```

모델, processor

after

```python
# ----------------------------
# 프로세서
# ----------------------------
processor = AutoProcessor.from_pretrained(
    MODEL_ID,
    min_pixels=IMAGE_SIZE*IMAGE_SIZE,
    max_pixels=IMAGE_SIZE*IMAGE_SIZE,
    trust_remote_code=True,
)

# ----------------------------
# 사전학습 모델 로드 (bf16 full precision)
# ----------------------------
base_model = AutoModelForVision2Seq.from_pretrained(
    MODEL_ID,
    torch_dtype=torch.bfloat16,   # bf16 full precision
    device_map="auto",
    trust_remote_code=True,
)

# Gradient checkpointing 켜서 VRAM 절약
base_model.gradient_checkpointing_enable()

# ----------------------------
# LoRA 세팅
# ----------------------------
lora_config = LoraConfig(
    r=32,
    lora_alpha=64,
    lora_dropout=0.05,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    task_type="CAUSAL_LM",
)
```

항목	기존	변경 후
양자화 설정	BitsAndBytes Config + 4bit	삭제
kbit 학습 준비	prepare_model_ for_kbit_training	삭제
dtype	4bit float16	bf16 full precision
gradient checkpointing	있음	그대로 유지
LoRA	r = 8 lora_alpha = 16	r = 32 lora_alpha = 64

데이터 로더

before

```python
# 검증용 데이터 분리
split = int(len(train_df)*0.9)
train_subset, valid_subset = train_df.iloc[:split], train_df.iloc[split:]

# VQAMCDataset 형태로 변환
train_ds = VQAMCDataset(train_subset, processor, train=True)
valid_ds = VQAMCDataset(valid_subset, processor, train=True)

# 데이터로더
train_loader = DataLoader(train_ds, batch_size=1, shuffle=True, collate_fn=DataCollator(processor, True), num_workers=0)
valid_loader = DataLoader(valid_ds, batch_size=1, shuffle=False, collate_fn=DataCollator(processor, True), num_workers=0)
```

after

```python
# 검증용 데이터 분리
split = int(len(train_df)*0.9)
train_subset, valid_subset = train_df.iloc[:split], train_df.iloc[split:]

# VQAMCDataset 형태로 변환
train_ds = VQAMCDataset(train_subset, processor, train=True)
valid_ds = VQAMCDataset(valid_subset, processor, train=True)

# 데이터로더
train_loader = DataLoader(train_ds, batch_size=2, shuffle=True, collate_fn=DataCollator(processor, True), num_workers=0)
valid_loader = DataLoader(valid_ds, batch_size=2, shuffle=False, collate_fn=DataCollator(processor, True), num_workers=0)
```

fine-tuning

before

```
# 옵티마이저, 학습 스케줄러
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4)
num_training_steps = 1 * math.ceil(len(train_loader)/GRAD_ACCUM)
scheduler = get_linear_schedule_with_warmup(optimizer, int(num_training_steps*0.03), num_training_steps)
# 학습 루프
global_step = 0
for epoch in range(1):
    running = 0.0
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1} [train]", unit="batch")
    for step, batch in enumerate(progress_bar, start=1):
        batch = {k:v.to(device) for k,v in batch.items()}
```

학습률 (lr)

1e-4 -> 2e-4

after

```
# 옵티마이저, 학습 스케줄러
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-4)
num_training_steps = 5 * math.ceil(len(train_loader)/GRAD_ACCUM)
scheduler = get_linear_schedule_with_warmup(optimizer, int(num_training_steps*0.03), num_training_steps)
# 학습 루프
global_step = 0
for epoch in range(5):
    running = 0.0
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1} [train]", unit="batch")
    for step, batch in enumerate(progress_bar, start=1):
        batch = {k:v.to(device) for k,v in batch.items()}
```

epoch

1 -> 5

소감

김민수

AI 공부를 충분히 하지 못한 상태에서 챌린지를 하게 되어 많은 어려움이 있었으나 LLM을 사용하여 코드를 이해하고 사전학습 모델 사용법을 배울 수 있었습니다

황가연

모델 구조와 세부 하이퍼파라미터의 작은 변화만으로도 정확도에 큰 영향을 미친다는 것을 깨달았습니다
시간이 부족해 데이터 정제 과정을 충분히 수행하지 못한 점이 아쉽습니다

한상민

GPU의 중요성을 직접 체감하는 계기가 되었습니다
모델 선정도 중요하다는 것을 알 수 있었습니다

양새하

UI로 표현된 AI 모델링 툴을 여럿 사용해봤지만
직접 코드를 수정하며 AI 모델링을 해본것은 처음이었습니다

코드를 완벽하게 이해하지 못한 상황에서
잘못 수정했다가는 망칠 것 같다는 걱정에
소심하게 수치 조정 정도만 진행하였지만,

주변에서 ChatGPT 를 활용하여 코드를 얹는 것을 보고
용기를 얻어 점점 더 과감하게 코드 수정을 할 수 있었습니다

발표자료 준비를 하며, 이제서야 코드를 이해하게 되어,
SSH 서버 사용 시간이 조금 더 주어졌더라면,
더 제대로 학습시켜보고 싶다는 욕심이 생겼습니다

FIN

Thank you

1-4. 자체 평가 의견

ChatGPT 를 활용하여 코드를 추천받을때, 현재 주어진 환경의 제약사항을 제공하여, SSH 환경에서 최대 성능을 낼 수 있는 모델을 추천받았던 것은 잘했다고 생각함. SSH 서버 시간 제한으로 인해 epoch 수를 크게 높이지 못했던 부분이 아쉬움. 하이퍼파라미터와 프롬프트를 수정하여 성능을 크게 높일 수 있었지만, 팀 기록은 갱신하지 못했음. 이번 프로젝트에서의 프로토타입 개발을 통해, AI 모델링에서의 하이퍼파라미터와 모델의 역할을 이해할 수 있었고, ChatGPT 를 활용하여 빠른 자료조사와 모델 추천을 받을 수 있었기에, 프롬프트 엔지니어링의 중요성을 깨달음.

Part 2. 개인 회고 작성

학습 목표를 달성하기 위해 각 하이퍼파라미터의 역할을 조사하고 이해하려고 함. 이론에만 의지하지 않고 ChatGPT 를 활용하여 시뮬레이션을 통해 최적의 수치를 도출하고자 하였고, 0.24176 에서 0.86625 로 성능을 큰 폭 향상시킬 수 있었음. 다만, loss 0.0706 를 확인하여 0.93 가량의 성능을 예상하였지만, 0.86 성능을 보여 아쉬웠음. 하이퍼파라미터의 역할을 제대로 이해하는 데에 시간이 오래 걸려, 모델링에 제대로 적용해보지 못했지만, 다음에 SSH 서버 시간이 더 주어진다면, 코드를 완벽하게 이해하고 모델링을 자유자재로 수정할 수 있게 되고 싶음.

모델링 코드를 이해하고 적용하는데에 시간이 걸려, 팀원들과 속도를 맞추지 못함. 다른 팀원들이 앞서나가며 빠른 속도로 팀 기록을 갱신하고 있을때, 그제서야 loss 와 성능의 관계를 이해함. 홀로 헤매고 있을때 팀원들이 관심을 갖고 도와줬으면 좋았을 것 같다는 원망과, 전날 수면 부족으로 팀플에 충분한 효율을 내지 못한 것 같다는 죄책감을 느낌. 팀원들과 갈등이 있었지만, 상황분석으로 원인을 파악할 수 있었고, 갈등을 키우지 않고 자연스럽게 팀에 다시 스며들 수 있었음.