In [31]:
```
# ============================================================
# CELL 1 — IMPORT & CONFIG
# ============================================================
import os, json, math, random, re
from dataclasses import dataclass
from typing import List, Dict, Any

import numpy as np
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm

import torch
from torch.utils.data import Dataset, DataLoader

from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    get_linear_schedule_with_warmup,
)

from peft import LoraConfig, get_peft_model
from huggingface_hub import login


# -------------------------
# Reproducibility
# -------------------------
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)

# -------------------------
# Paths
# -------------------------
DATA_JSON_PATH = "dataset_qa.json"
SPLIT_DIR      = "splits_json"
OUTPUT_DIR     = "output_lora"
os.makedirs(SPLIT_DIR, exist_ok=True)
os.makedirs(OUTPUT_DIR, exist_ok=True)

# -------------------------
# Models
# -------------------------
BASE_MODEL = "meta-llama/Llama-3.2-1B-Instruct"   # gated (but you already got acce
EVAL_MODEL = "Qwen/Qwen2.5-0.5B-Instruct"         # evaluator model (scoring 0-5)

# -------------------------
# Training hyperparams
# -------------------------
MAX_LENGTH = 512
BATCH_SIZE = 2
```

```
GRAD_ACCUM_STEPS = 8
EPOCHS = 3
LR = 2e-4
WEIGHT_DECAY = 0.01
WARMUP_RATIO = 0.06
MAX_GRAD_NORM = 1.0

# Generation (prediksi test set)
GEN_MAX_NEW_TOKENS = 256
GEN_TEMPERATURE = 0.7
GEN_TOP_P = 0.95

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
DTYPE  = torch.float16 if torch.cuda.is_available() else torch.float32

print("DEVICE:", DEVICE, "| DTYPE:", DTYPE)
print("BASE_MODEL:", BASE_MODEL)
print("EVAL_MODEL:", EVAL_MODEL)
```

```
DEVICE: cuda | DTYPE: torch.float16
BASE_MODEL: meta-llama/Llama-3.2-1B-Instruct
EVAL_MODEL: Qwen/Qwen2.5-0.5B-Instruct
```

In [32]:

```
## ============================================================
# CELL 2 — HF LOGIN TOKEN |
# ============================================================
# Cara paling aman: paste token sekali per runtime
# Token ambil dari HF: Settings -> Access Tokens (Read)
HF_TOKEN = os.environ.get("HF_TOKEN", None)
if HF_TOKEN is None:
    HF_TOKEN = getpass("Token : ")

login(HF_TOKEN)
print("[OK] HF login done")
```

```
[OK] HF login done
```

In [33]:

```
# ============================================================
# CELL 3 — LOAD JSON + VALIDASI + SPLIT (train/val/test)
# Rubrik: Dataset + Data Preparation
# ============================================================
if not os.path.exists(DATA_JSON_PATH):
    raise FileNotFoundError(f"File JSON tidak ditemukan: {DATA_JSON_PATH}")

with open(DATA_JSON_PATH, "r", encoding="utf-8") as f:
    data = json.load(f)

# aman jika format {"data":[...]}
if isinstance(data, dict) and "data" in data and isinstance(data["data"], list):
    data = data["data"]

def validate_records(records: List[Dict[str, Any]]) -> List[Dict[str, str]]:
    if not isinstance(records, list):
        raise ValueError("JSON harus list of dict")

    cleaned: List[Dict[str, str]] = []
    for i, r in enumerate(records):
```

```
        if not isinstance(r, dict):
            raise ValueError(f"Record #{i} bukan dict: {type(r)}")

        for k in ("instruction", "input", "output"):
            if k not in r:
                raise ValueError(f"Record #{i} tidak punya key '{k}'. Keys: {list(r

        instr = str(r["instruction"]).strip()
        inp   = str(r["input"]).strip()
        out   = str(r["output"]).strip()

        if instr == "" or out == "":
            raise ValueError(f"Record #{i} instruction/output kosong")

        cleaned.append({"instruction": instr, "input": inp, "output": out})
    return cleaned

data = validate_records(data)
print("[OK] Total records:", len(data))
print("Contoh 1 record:\n", json.dumps(data[0], ensure_ascii=False, indent=2))

train_data, temp_data = train_test_split(
    data, test_size=0.2, random_state=SEED, shuffle=True
)
val_data, test_data = train_test_split(
    temp_data, test_size=0.5, random_state=SEED, shuffle=True
)

print("train:", len(train_data), "val:", len(val_data), "test:", len(test_data))

with open(os.path.join(SPLIT_DIR, "train.json"), "w", encoding="utf-8") as f:
    json.dump(train_data, f, ensure_ascii=False, indent=2)
with open(os.path.join(SPLIT_DIR, "val.json"), "w", encoding="utf-8") as f:
    json.dump(val_data, f, ensure_ascii=False, indent=2)
with open(os.path.join(SPLIT_DIR, "test.json"), "w", encoding="utf-8") as f:
    json.dump(test_data, f, ensure_ascii=False, indent=2)

print("[OK] Split saved in:", SPLIT_DIR)
```

```
[OK] Total records: 170
Contoh 1 record:
 {
  "instruction": "Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat
terkait informasi Mirota Kampus.",
  "input": "Apa itu Mirota Kampus?",
  "output": "Mirota Kampus adalah jaringan toko retail di Yogyakarta yang menjual be
rbagai kebutuhan mahasiswa dan masyarakat, termasuk alat tulis, kosmetik, sembako, s
nack, dan perlengkapan rumah tangga."
}
train: 136 val: 17 test: 17
[OK] Split saved in: splits_json
```

In [34]:

```
# ============================================================
# CELL 4 — TOKENIZER + PROMPT FORMATTING (chat template)
# Rubrik: Tokenization & Prompt Formatting
# ============================================================
```

```
def tok_from_pretrained(name: str, token: str):
    try:
        return AutoTokenizer.from_pretrained(name, use_fast=True, token=token)
    except TypeError:
        return AutoTokenizer.from_pretrained(name, use_fast=True, use_auth_token=to

tokenizer = tok_from_pretrained(BASE_MODEL, HF_TOKEN)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"

SYSTEM_PROMPT = "Kamu adalah asisten AI yang membantu dan menjawab dengan jelas."

def build_user_text(instruction: str, user_input: str) -> str:
    instruction = instruction.strip()
    user_input = (user_input or "").strip()
    if user_input:
        return f"{instruction}\n\nInput:\n{user_input}"
    return instruction

def encode_prompt_and_full(instruction: str, user_input: str, answer: str, max_leng
    """
    prompt_ids: system+user sampai titik assistant (generation prompt)
    full_ids  : system+user+assistant(answer)
    labels    : -100 untuk prompt, token jawaban dihitung loss
    """
    user_text = build_user_text(instruction, user_input)
    answer = answer.strip()

    messages_prompt = [
        {"role": "system", "content": SYSTEM_PROMPT},
        {"role": "user", "content": user_text},
    ]
    prompt_ids = tokenizer.apply_chat_template(
        messages_prompt,
        tokenize=True,
        add_generation_prompt=True,
        truncation=True,
        max_length=max_length,
    )

    messages_full = messages_prompt + [{"role": "assistant", "content": answer}]
    full_ids = tokenizer.apply_chat_template(
        messages_full,
        tokenize=True,
        add_generation_prompt=False,
        truncation=True,
        max_length=max_length,
    )

    prompt_len = len(prompt_ids)
    labels = [-100] * min(prompt_len, len(full_ids)) + full_ids[min(prompt_len, len
    labels = labels[:len(full_ids)]
    return full_ids, labels

print("Contoh user_text:\n", build_user_text(train_data[0]["instruction"], train_da
```

Contoh user_text:
 Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat terkait informasi Mirota Kampus.

Input:
Kalau cari ATK buat kuliah lengkap nggak di sana?

In [35]:
```
# ============================================================
# CELL 5 — DATASET + COLLATE FUNCTION
# Rubrik: InstructionDataset + custom_collate_fn
# ============================================================
@dataclass
class EncodedExample:
    input_ids: List[int]
    labels: List[int]

class InstructionDataset(Dataset):
    def __init__(self, records: List[Dict[str, str]], max_length: int = 512):
        self.records = records
        self.max_length = max_length
        self.encoded: List[EncodedExample] = []
        self._build()

    def _build(self):
        for r in tqdm(self.records, desc="Encoding dataset"):
            full_ids, labels = encode_prompt_and_full(
                r["instruction"], r["input"], r["output"], self.max_length
            )
            self.encoded.append(EncodedExample(input_ids=full_ids, labels=labels))

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

    def __getitem__(self, idx):
        ex = self.encoded[idx]
        return {"input_ids": ex.input_ids, "labels": ex.labels}

def custom_collate_fn(batch):
    input_ids = [torch.tensor(x["input_ids"], dtype=torch.long) for x in batch]
    labels    = [torch.tensor(x["labels"], dtype=torch.long) for x in batch]

    input_ids = torch.nn.utils.rnn.pad_sequence(
        input_ids, batch_first=True, padding_value=tokenizer.pad_token_id
    )
    labels = torch.nn.utils.rnn.pad_sequence(
        labels, batch_first=True, padding_value=-100
    )

    attention_mask = (input_ids != tokenizer.pad_token_id).long()
    return {"input_ids": input_ids, "attention_mask": attention_mask, "labels": lab

train_ds = InstructionDataset(train_data, max_length=MAX_LENGTH)
val_ds   = InstructionDataset(val_data,   max_length=MAX_LENGTH)
test_ds  = InstructionDataset(test_data,  max_length=MAX_LENGTH)

train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,  collate_f
```

```
val_loader   = DataLoader(val_ds,   batch_size=BATCH_SIZE, shuffle=False, collate_f
test_loader  = DataLoader(test_ds,  batch_size=1,          shuffle=False, collate_fn

print(f"sizes: train={len(train_ds)} val={len(val_ds)} test={len(test_ds)}")
```

```
Encoding dataset: 100%|██████████| 136/136 [00:00<00:00, 605.68it/s]
Encoding dataset: 100%|██████████| 17/17 [00:00<00:00, 427.13it/s]
Encoding dataset: 100%|██████████| 17/17 [00:00<00:00, 341.90it/s]
sizes: train=136 val=17 test=17
```

In [36]:
```
# ============================================================
# CELL 6 — LOAD BASE MODEL + LoRA
# Rubrik: Fine-Tuning LLM (load model, pindah device)
# ============================================================
def model_from_pretrained(name: str, token: str, dtype):
    try:
        return AutoModelForCausalLM.from_pretrained(name, token=token, torch_dtype=
    except TypeError:
        return AutoModelForCausalLM.from_pretrained(name, use_auth_token=token, tor

base_model = model_from_pretrained(BASE_MODEL, HF_TOKEN, DTYPE)
base_model.to(DEVICE)
base_model.config.use_cache = False

lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down
)

model = get_peft_model(base_model, lora_config)
model.print_trainable_parameters()
```

```
trainable params: 11,272,192 || all params: 1,247,086,592 || trainable%: 0.9039
```

In [37]:
```
# ============================================================
# CELL 7 — BASELINE TRAIN/VAL LOSS (SEBELUM TRAINING)
# Rubrik: hitung train/val loss awal
# ============================================================
use_amp = (DEVICE == "cuda")
scaler = torch.cuda.amp.GradScaler(enabled=use_amp)

@torch.no_grad()
def evaluate_loss(dataloader):
    model.eval()
    losses = []
    for batch in dataloader:
        batch = {k: v.to(DEVICE) for k, v in batch.items()}
        with torch.cuda.amp.autocast(enabled=use_amp):
            out = model(**batch)
            losses.append(out.loss.item())
    model.train()
    return float(np.mean(losses)) if losses else 0.0
```

```
train_loss0 = evaluate_loss(train_loader)
val_loss0   = evaluate_loss(val_loader)
print("Baseline train_loss:", train_loss0)
print("Baseline val_loss  :", val_loss0)
```

```
/tmp/ipykernel_1427855/3310612489.py:6: FutureWarning: `torch.cuda.amp.GradScaler(ar
gs...)` is deprecated. Please use `torch.amp.GradScaler('cuda', args...)` instead.
  scaler = torch.cuda.amp.GradScaler(enabled=use_amp)
/tmp/ipykernel_1427855/3310612489.py:14: FutureWarning: `torch.cuda.amp.autocast(arg
s...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(enabled=use_amp):
Baseline train_loss: 2.4883751413401436
Baseline val_loss  : 2.5526473124821982
```

In [38]:

```
# ============================================================
# CELL 8 — TRAIN LOOP (train_model_simple)
# Rubrik: train_model_simple + training loop
# ============================================================
def train_model_simple(
    model,
    train_loader,
    val_loader,
    epochs: int = 3,
    lr: float = 2e-4,
    grad_accum_steps: int = 8,
    weight_decay: float = 0.01,
    warmup_ratio: float = 0.06,
    max_grad_norm: float = 1.0,
):
    optimizer = torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=weight_de

    total_steps  = math.ceil(len(train_loader) / grad_accum_steps) * epochs
    warmup_steps = int(total_steps * warmup_ratio)

    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        num_warmup_steps=warmup_steps,
        num_training_steps=total_steps,
    )

    model.train()
    for epoch in range(1, epochs + 1):
        pbar = tqdm(train_loader, desc=f"Epoch {epoch}/{epochs}")
        optimizer.zero_grad(set_to_none=True)
        running = 0.0

        for step, batch in enumerate(pbar, start=1):
            batch = {k: v.to(DEVICE) for k, v in batch.items()}

            with torch.cuda.amp.autocast(enabled=use_amp):
                out = model(**batch)
                loss = out.loss / grad_accum_steps

            if use_amp:
                scaler.scale(loss).backward()
```

```
            else:
                loss.backward()

            running += loss.item()

            do_step = (step % grad_accum_steps == 0)
            is_last = (step == len(train_loader))
            if do_step or is_last:
                if use_amp:
                    scaler.unscale_(optimizer)

                torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)

                if use_amp:
                    scaler.step(optimizer)
                    scaler.update()
                else:
                    optimizer.step()

                optimizer.zero_grad(set_to_none=True)
                scheduler.step()

                pbar.set_postfix({"loss": float(running)})
                running = 0.0

        val_loss = evaluate_loss(val_loader)
        print(f"[Epoch {epoch}] val_loss={val_loss:.4f}")

    return model

model = train_model_simple(
    model=model,
    train_loader=train_loader,
    val_loader=val_loader,
    epochs=EPOCHS,
    lr=LR,
    grad_accum_steps=GRAD_ACCUM_STEPS,
    weight_decay=WEIGHT_DECAY,
    warmup_ratio=WARMUP_RATIO,
    max_grad_norm=MAX_GRAD_NORM,
)
```

```
/tmp/ipykernel_1427855/3911211188.py:36: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(enabled=use_amp):
Epoch 1/3: 100%|██████████| 68/68 [00:13<00:00,  4.97it/s, loss=0.775]
/tmp/ipykernel_1427855/3310612489.py:14: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(enabled=use_amp):
[Epoch 1] val_loss=1.4080
Epoch 2/3: 100%|██████████| 68/68 [00:13<00:00,  5.12it/s, loss=0.607]
[Epoch 2] val_loss=1.2403
Epoch 3/3: 100%|██████████| 68/68 [00:13<00:00,  4.92it/s, loss=0.539]
[Epoch 3] val_loss=1.1914
```

In [39]:
```
# ============================================================
# CELL 9 — SAVE ADAPTER + TOKENIZER
# Rubrik: deployment preparation (artifacts)
# ============================================================
model.save_pretrained(OUTPUT_DIR)
tokenizer.save_pretrained(OUTPUT_DIR)

with open(os.path.join(OUTPUT_DIR, "meta.json"), "w", encoding="utf-8") as f:
    json.dump({"base_model": BASE_MODEL}, f, ensure_ascii=False, indent=2)

print("[OK] saved adapter to:", OUTPUT_DIR)
```

```
[OK] saved adapter to: output_lora
```

In [40]:
```
# ============================================================
# CELL 10 — GENERATE PREDICTIONS TEST SET + SAVE JSON
# Rubrik: Model Prediction / Demo
# ============================================================
@torch.inference_mode()
def generate_answer(instruction: str, user_input: str = "") -> str:
    user_text = build_user_text(instruction, user_input)
    messages = [
        {"role": "system", "content": SYSTEM_PROMPT},
        {"role": "user", "content": user_text},
    ]

    input_ids = tokenizer.apply_chat_template(
        messages,
        tokenize=True,
        add_generation_prompt=True,
        return_tensors="pt",
    ).to(DEVICE)

    gen_ids = model.generate(
        input_ids=input_ids,
        max_new_tokens=GEN_MAX_NEW_TOKENS,
        do_sample=True,
        temperature=GEN_TEMPERATURE,
        top_p=GEN_TOP_P,
        eos_token_id=tokenizer.eos_token_id,
        pad_token_id=tokenizer.pad_token_id,
    )

    new_tokens = gen_ids[0, input_ids.shape[1]:]
    text = tokenizer.decode(new_tokens, skip_special_tokens=True).strip()
    return text

predictions = []
for r in tqdm(test_data, desc="Generating test predictions"):
    resp = generate_answer(r["instruction"], r["input"])
    out = dict(r)
    out["model_response"] = resp
    predictions.append(out)

PRED_PATH = "predictions_test.json"
with open(PRED_PATH, "w", encoding="utf-8") as f:
```

```
    json.dump(predictions, f, ensure_ascii=False, indent=2)

print("[OK] saved:", PRED_PATH)
print("Sample prediction:\n", json.dumps(predictions[0], ensure_ascii=False, indent
```

```
Generating test predictions: 100%|██████████| 17/17 [00:18<00:00,  1.07s/it]
[OK] saved: predictions_test.json
Sample prediction:
 {
  "instruction": "Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat
terkait informasi Mirota Kampus.",
  "input": "Mirota bisa jadi tempat belanja anak kos nggak?",
  "output": "Bisa, karena Mirota Kampus menyediakan banyak kebutuhan anak kos dan ma
hasiswa.",
  "model_response": "Ya, Mirota Kampus menyediakan perlengkapan kos anak, perlengkap
an rumah tangga, dan kebutuhan hiburan."
}
```

In [41]:

```
# ============================================================
# CELL 10A — QUICK TEST GENERATION (cek kesesuaian vs dataset)
# ============================================================
import random

# pilih sumber sampel: "val" atau "test"
QUICK_SOURCE = "val"   # ganti ke "test" kalau mau
N_SAMPLES = 5          # jumlah contoh yang mau dicek
SEED_QUICK = 123       # biar konsisten

def _short(text: str, max_chars: int = 800) -> str:
    text = (text or "").strip()
    return text if len(text) <= max_chars else text[:max_chars] + "\n...[TRUNCATED]

# pilih data
if QUICK_SOURCE.lower() == "test":
    pool = test_data
else:
    pool = val_data

if len(pool) == 0:
    raise ValueError("Pool data kosong. Pastikan val_data/test_data ada.")

random.seed(SEED_QUICK)
indices = random.sample(range(len(pool)), k=min(N_SAMPLES, len(pool)))

print(f"\n🧪 QUICK TEST ({QUICK_SOURCE.upper()}) — {len(indices)} samples\n")

for j, idx in enumerate(indices, start=1):
    r = pool[idx]
    instr = r["instruction"]
    inp   = r["input"]
    ref   = r["output"]

    pred = generate_answer(instr, inp)

    print("=" * 90)
    print(f"[{j}] idx={idx}")
```

```
    print("-" * 90)
    print("INSTRUCTION:")
    print(_short(instr, 600))
    print("\nINPUT:")
    print(_short(inp, 600))
    print("\nGROUND TRUTH (dataset output):")
    print(_short(ref, 800))
    print("\nMODEL RESPONSE:")
    print(_short(pred, 800))

    # kalau evaluator scoring sudah ada (CELL 12), tampilkan skor 0-5
    if "llm_score_one" in globals():
        try:
            s, raw = llm_score_one(instr, inp, ref, pred)
            print("\nEVAL SCORE (0-5):", s)
            print("EVAL RAW:")
            print(_short(raw, 500))
        except Exception as e:
            print("\n(Evaluator tersedia tapi gagal jalan):", repr(e))

print("\n✅ Selesai QUICK TEST. Lihat apakah MODEL RESPONSE sudah mengikuti pola ja
```

🧪 QUICK TEST (VAL) — 5 samples

```
================================================================================
======
[1] idx=1
--------------------------------------------------------------------------------
------
INSTRUCTION:
Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat terkait informasi
Mirota Kampus.

INPUT:
Tadi katanya bisa print, tapi ternyata nggak bisa.

GROUND TRUTH (dataset output):
Mohon maaf atas kebingungannya. Layanan dapat berbeda tergantung cabang dan jam oper
asional.

MODEL RESPONSE:
Kami memahami masalah print yang tidak berjalan. Kami sarankan Anda untuk memeriksa
ketersediaan stok dan kondisi printer.

(Evaluator tersedia tapi gagal jalan): NameError("name 'eval_tokenizer' is not defin
ed")
================================================================================
======
[2] idx=8
--------------------------------------------------------------------------------
------
INSTRUCTION:
Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat terkait informasi
Mirota Kampus.

INPUT:
Mirota ada jual kosmetik juga nggak sih?

GROUND TRUTH (dataset output):
Ya, tersedia berbagai produk kosmetik dan perawatan diri di beberapa cabang.

MODEL RESPONSE:
Ya, Mirota Kampus menjual berbagai kebutuhan kosmetik dan hiburan.

(Evaluator tersedia tapi gagal jalan): NameError("name 'eval_tokenizer' is not defin
ed")
================================================================================
======
[3] idx=16
--------------------------------------------------------------------------------
------
INSTRUCTION:
Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat terkait informasi
Mirota Kampus.

INPUT:
Kalau anak kos, kebutuhannya bisa kebeli semua nggak di Mirota?
```

```
GROUND TRUTH (dataset output):
Banyak kebutuhan anak kos seperti alat kebersihan dan perlengkapan rumah tangga ters
edia di Mirota Kampus.

MODEL RESPONSE:
Beberapa produk umum tersedia di Mirota Kampus, namun kebijakan dan ketersediaan dap
at berbeda.

(Evaluator tersedia tapi gagal jalan): NameError("name 'eval_tokenizer' is not defin
ed")
================================================================================
======
[4] idx=12
--------------------------------------------------------------------------------
------
INSTRUCTION:
Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat terkait informasi
Mirota Kampus.

INPUT:
Tadi saya ke Mirota tapi barang yang dicari nggak ada.

GROUND TRUTH (dataset output):
Mohon maaf atas ketidaknyamanannya. Stok barang bisa berbeda di tiap cabang dan wakt
u tertentu.

MODEL RESPONSE:
Kami mengerti masalah itu. Stok bisa berubah sebelum jam tertentu.

(Evaluator tersedia tapi gagal jalan): NameError("name 'eval_tokenizer' is not defin
ed")
================================================================================
======
[5] idx=6
--------------------------------------------------------------------------------
------
INSTRUCTION:
Jawab pertanyaan pengguna berikut dengan jelas, sopan, dan akurat terkait informasi
Mirota Kampus.

INPUT:
Ada promo atau diskon?

GROUND TRUTH (dataset output):
Mirota Kampus sering menyediakan promo musiman dan diskon produk tertentu.

MODEL RESPONSE:
Promo dan diskon tersedia di beberapa cabang dan waktu tertentu.

(Evaluator tersedia tapi gagal jalan): NameError("name 'eval_tokenizer' is not defin
ed")

✅ Selesai QUICK TEST. Lihat apakah MODEL RESPONSE sudah mengikuti pola jawaban dat
aset kamu.
```

In [42]:
```
# ============================================================
# CELL 11 — LOAD EVALUATOR LLM (LLM-based scoring 0-5)
# Rubrik: Model Evaluation (LLM-based Scoring)
# ============================================================
eval_tokenizer = AutoTokenizer.from_pretrained(EVAL_MODEL, use_fast=True)
if eval_tokenizer.pad_token is None:
    eval_tokenizer.pad_token = eval_tokenizer.eos_token
eval_tokenizer.padding_side = "right"

eval_model = AutoModelForCausalLM.from_pretrained(EVAL_MODEL, torch_dtype=DTYPE)
eval_model.to(DEVICE)
eval_model.eval()

print("[OK] evaluator loaded")
```

```
[OK] evaluator loaded
```

In [43]:
```
# ============================================================
# CELL 12 — SCORING FUNCTION (0-5) + HANDLING INVALID
# ============================================================
EVAL_SYSTEM = (
    "Kamu adalah evaluator jawaban. Beri skor 0 sampai 5.\n"
    "0=sepenuhnya salah/halu, 1=sebagian besar salah, 2=kurang tepat, "
    "3=cukup benar tapi ada kekurangan, 4=benar & jelas, 5=sangat benar, lengkap, d
    "Output WAJIB format:\n"
    "SCORE: <angka 0-5>\n"
    "REASON: <alasan singkat>"
)

def parse_score(text: str) -> int | None:
    m = re.search(r"SCORE\s*:\s*([0-5])", text)
    if m:
        return int(m.group(1))
    m2 = re.search(r"\b([0-5])\b", text)
    return int(m2.group(1)) if m2 else None

@torch.inference_mode()
def llm_score_one(instruction: str, user_input: str, reference: str, model_response
    user_text = build_user_text(instruction, user_input)

    prompt = (
        "Nilai jawaban model berdasarkan instruksi dan referensi jawaban benar.\n\n
        f"INSTRUCTION+INPUT:\n{user_text}\n\n"
        f"REFERENCE ANSWER:\n{reference}\n\n"
        f"MODEL ANSWER:\n{model_response}\n\n"
        "Beri skor 0-5."
    )

    messages = [
        {"role": "system", "content": EVAL_SYSTEM},
        {"role": "user", "content": prompt},
    ]

    input_ids = eval_tokenizer.apply_chat_template(
        messages,
        tokenize=True,
```

```
        add_generation_prompt=True,
        return_tensors="pt",
    ).to(DEVICE)

    gen_ids = eval_model.generate(
        input_ids=input_ids,
        max_new_tokens=max_new_tokens,
        do_sample=False,
        eos_token_id=eval_tokenizer.eos_token_id,
        pad_token_id=eval_tokenizer.pad_token_id,
    )

    new_tokens = gen_ids[0, input_ids.shape[1]:]
    text = eval_tokenizer.decode(new_tokens, skip_special_tokens=True).strip()

    score = parse_score(text)
    if score is None:
        # handling invalid score
        score = 0

    return int(score), text

# quick test
sample = predictions[0]
s, raw = llm_score_one(sample["instruction"], sample["input"], sample["output"], sa
print("score:", s)
print("raw:\n", raw[:800])
```

```
/opt/tljh/user/envs/dltf/lib/python3.12/site-packages/transformers/generation/config
uration_utils.py:628: UserWarning: `do_sample` is set to `False`. However, `temperat
ure` is set to `0.7` -- this flag is only used in sample-based generation modes. You
should set `do_sample=True` or unset `temperature`.
  warnings.warn(
/opt/tljh/user/envs/dltf/lib/python3.12/site-packages/transformers/generation/config
uration_utils.py:633: UserWarning: `do_sample` is set to `False`. However, `top_p` i
s set to `0.8` -- this flag is only used in sample-based generation modes. You shoul
d set `do_sample=True` or unset `top_p`.
  warnings.warn(
/opt/tljh/user/envs/dltf/lib/python3.12/site-packages/transformers/generation/config
uration_utils.py:650: UserWarning: `do_sample` is set to `False`. However, `top_k` i
s set to `20` -- this flag is only used in sample-based generation modes. You should
set `do_sample=True` or unset `top_k`.
  warnings.warn(
score: 0
raw:
 SCORE: 0
REASON: Jawaban yang diberikan tidak sesuai dengan instruksi dan referensi jawaban y
ang diinginkan. Jelaskanannya lebih lanjut:

"Jawaban yang diberikan tidak mencerminkan bahwa Mirota Kampus menyediakan perlengka
pan kos anak. Ini hanya mengatakan bahwa Mirota Kampus memiliki banyak jenis perleng
kapan untuk anak kos dan mahasiswa."

Dengan kata lain, jawaban tersebut tidak menjelaskan secara spesifik tentang apa yan
g disediakan oleh Mirota Kampus dalam hal perlengk
```

In [44]:
```
# ============================================================
# CELL 13 — SCORE ALL + SAVE + STATS
# ============================================================
scored = []
for r in tqdm(predictions, desc="Scoring predictions (0-5)"):
    s, raw = llm_score_one(r["instruction"], r["input"], r["output"], r["model_resp
    r2 = dict(r)
    r2["score_0_5"] = int(s)
    r2["score_raw"] = raw
    scored.append(r2)

EVAL_PATH = "eval_scores.json"
with open(EVAL_PATH, "w", encoding="utf-8") as f:
    json.dump(scored, f, ensure_ascii=False, indent=2)

arr = np.array([x["score_0_5"] for x in scored], dtype=np.int32)
stats = {
    "n": int(arr.size),
    "mean": float(arr.mean()) if arr.size else 0.0,
    "median": float(np.median(arr)) if arr.size else 0.0,
    "min": int(arr.min()) if arr.size else 0,
    "max": int(arr.max()) if arr.size else 0,
    "distribution": {str(i): int((arr == i).sum()) for i in range(6)},
}

print("[OK] saved:", EVAL_PATH)
print("STATS:", json.dumps(stats, indent=2, ensure_ascii=False))
```

```
Scoring predictions (0-5): 100%|██████████| 17/17 [00:28<00:00,  1.65s/it]
[OK] saved: eval_scores.json
STATS: {
  "n": 17,
  "mean": 0.6470588235294118,
  "median": 0.0,
  "min": 0,
  "max": 2,
  "distribution": {
    "0": 11,
    "1": 1,
    "2": 5,
    "3": 0,
    "4": 0,
    "5": 0
  }
}
```