```
!pip -q install -U "transformers>=4.44" accelerate peft safetensors
!pip -q install -U bitsandbytes
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.1/59.1 MB 46.3 MB/s eta
0:00:00
```

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
from torch.utils.data import Dataset
```

Dataset + Data Preparation

Load Dataset

```
import json


file_path = "dataset_pmb_uajy.json"

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


    if not isinstance(data, list):
        raise ValueError("JSON harus berupa LIST of objects")


    required = {"category", "instruction", "input", "output"}
    for i, ex in enumerate(data[:20]):  # cek 20 pertama
        missing = required - set(ex.keys())
        if missing:
            raise ValueError(f"Index {i} missing keys: {missing}")

    categories = sorted({d["category"] for d in data})

    print("✅ JSON berhasil dibaca")
    print("Jumlah data :", len(data))
    print("Categories  :", categories)
    print("Contoh data:", data[0])

except FileNotFoundError:
    print(f"❌ File '{file_path}' tidak ditemukan. Pastikan nama file &
lokasi benar.")
except json.JSONDecodeError:
    print("❌ Format JSON tidak valid. Cek koma, kurung, dll.")
except Exception as e:
    print("❌ Error lain:", e)
```

```
 JSON berhasil dibaca
Jumlah data : 463
Categories  : ['Alur', 'Beasiswa', 'Biaya', 'PMB_Umum', 'Pembayaran',
'Prodi', 'Profesi', 'S1', 'S2', 'S3', 'Umum']
Contoh data: {'category': 'S1', 'instruction': 'Jawablah pertanyaan
berikut berdasarkan informasi resmi PMB Universitas Atma Jaya
Yogyakarta.', 'input': 'Siapa saja yang dapat mendaftar Program
Sarjana (S1) di Universitas Atma Jaya Yogyakarta melalui Program Nilai
Ijazah?', 'output': 'Program Nilai Ijazah terbuka bagi siswa SMA/SMK
yang telah menyelesaikan studi (Lulusan tahun 2026 dan sebelumnya)
yang penerimaannya didasarkan pada nilai ijazah.'}
```

Split train/val/test stratified by category

```
from sklearn.model_selection import train_test_split

labels = [d["category"] for d in data]

train_data, temp_data = train_test_split(
    data, test_size=0.2, random_state=42, shuffle=True,
stratify=labels
)

temp_labels = [d["category"] for d in temp_data]
val_data, test_data = train_test_split(
    temp_data, test_size=0.5, random_state=42, shuffle=True,
stratify=temp_labels
)

print(len(train_data), len(val_data), len(test_data))
```

```
370 46 47
```

Templating

```
def format_prompt(category: str, instruction: str, inp: str) -> str:
    category = (category or "").strip()
    instruction = (instruction or "").strip()
    inp = (inp or "").strip()

    header = f"### Category: {category}\n### Instruction:\
n{instruction}\n\n"
    if inp:
        return header + f"### Input:\n{inp}\n\n### Response:\n"
    else:
        return header + "### Response:\n"
```

Tokenization & Prompt Formatting

Tokenization

```
model_id = "Sahabat-AI/gemma2-9b-cpt-sahabatai-v1-base"
tokenizer = AutoTokenizer.from_pretrained(model_id)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

MAX_LEN = 1024

class InstructionDataset(Dataset):
    def __init__(self, data, tokenizer, max_len=1024):
        self.data = data
        self.tok = tokenizer
        self.max_len = max_len

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

    def __getitem__(self, idx):
        ex = self.data[idx]

        prompt = format_prompt(
            ex.get("category", ""),
            ex.get("instruction", ""),
            ex.get("input", "")
        )
        answer = (ex.get("output", "") or "").strip()

        full_text = prompt + answer + self.tok.eos_token

        enc = self.tok(
            full_text,
            truncation=True,
            max_length=self.max_len,
        )

        input_ids = enc["input_ids"]
        attention_mask = enc["attention_mask"]

        prompt_ids = self.tok(
            prompt,
            truncation=True,
            max_length=self.max_len,
        )["input_ids"]
```

```
        labels = input_ids.copy()

        for i in range(min(len(prompt_ids), len(labels))):
            labels[i] = -100

        return {
            "input_ids": torch.tensor(input_ids, dtype=torch.long),
            "attention_mask": torch.tensor(attention_mask,
dtype=torch.long),
            "labels": torch.tensor(labels, dtype=torch.long),
        }

train_ds = InstructionDataset(train_data, tokenizer, MAX_LEN)
val_ds   = InstructionDataset(val_data, tokenizer, MAX_LEN)
test_ds  = InstructionDataset(test_data, tokenizer, MAX_LEN)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"50638a7eba9c467487b5e9bad069831c","version_major":2,"vers
ion_minor":0}

{"model_id":"0fe17946e4ab4374ab5cf7a06f1a21cf","version_major":2,"vers
ion_minor":0}

{"model_id":"cf4fa061b731436a823486e803882053","version_major":2,"vers
ion_minor":0}

{"model_id":"c511a7baf50e4dd8b15b8cad9d6c4972","version_major":2,"vers
ion_minor":0}
```


Collate function (padding batch)

```
from torch.nn.utils.rnn import pad_sequence

def collate_fn(batch):
    input_ids = pad_sequence([x["input_ids"] for x in batch],
batch_first=True, padding_value=tokenizer.pad_token_id)
    attention_mask = pad_sequence([x["attention_mask"] for x in
batch], batch_first=True, padding_value=0)
    labels = pad_sequence([x["labels"] for x in batch],
batch_first=True, padding_value=-100)
```

```
    return {"input_ids": input_ids, "attention_mask": attention_mask,
"labels": labels}
```

Test DataLoader + sanity check decode

```
from torch.utils.data import DataLoader

train_loader = DataLoader(train_ds, batch_size=2, shuffle=True,
collate_fn=collate_fn)
batch = next(iter(train_loader))
print({k: v.shape for k, v in batch.items()})


sample = train_data[0]
print("\n--- Preview ---")
print(format_prompt(sample["category"], sample["instruction"],
sample.get("input","")) + sample["output"][:200])
```

```
{'input_ids': torch.Size([2, 73]), 'attention_mask': torch.Size([2,
73]), 'labels': torch.Size([2, 73])}

--- Preview ---
### Category: Profesi
### Instruction:
Sebutkan minimal IPK yang disyaratkan untuk mendaftar PPAr UAJY.

### Input:
Minimal IPK PPAr UAJY

### Response:
Minimal IPK yang disyaratkan untuk mendaftar PPAr UAJY adalah 3,00.
```

Fine Tunning

Load model

```
import torch
from transformers import AutoModelForCausalLM

model_id = "Sahabat-AI/gemma2-9b-cpt-sahabatai-v1-base"

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    device_map="auto"
)

print("Model Sahabat-AI loaded")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"5d87b0e43fc447f2a69a1f104bfd4415","version_major":2,"vers
ion_minor":0}
```


```
`torch_dtype` is deprecated! Use `dtype` instead!
```


```
{"model_id":"32aa14c2180a41fdb707d88c07fb6424","version_major":2,"vers
ion_minor":0}

{"model_id":"48e5924e14874d0faa9aebdc7e44f531","version_major":2,"vers
ion_minor":0}

{"model_id":"c2eb407d286144e4944fc3078fa47954","version_major":2,"vers
ion_minor":0}

{"model_id":"f25a5c00939c4376a56e92636c4d3641","version_major":2,"vers
ion_minor":0}

{"model_id":"23743b5ecfb7434d92a4fae6f02baf85","version_major":2,"vers
ion_minor":0}

{"model_id":"ac1859eb44834bcaa5fb981361b3382c","version_major":2,"vers
ion_minor":0}
```


```
The following generation flags are not valid and may be ignored:
['cache_implementation']. Set `TRANSFORMERS_VERBOSITY=info` for more
details.
```


```
{"model_id":"d5e9decb4b9443e4883076cf7e1ca3bd","version_major":2,"vers
ion_minor":0}

{"model_id":"65cc14407bb14c4baeb9ca3c932ce0df","version_major":2,"vers
ion_minor":0}
```


```
Model Sahabat-AI loaded
```

LoRA

```
!pip -q install peft

from peft import LoraConfig, get_peft_model
```

```
lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["q_proj", "v_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

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

Training Loop

```
from torch.optim import AdamW

optimizer = AdamW(model.parameters(), lr=2e-4)
model.train()
```

Hitung Loss awal (1 batch)

```
batch = next(iter(train_loader))
batch = {k: v.to(model.device) for k, v in batch.items()}

with torch.no_grad():
    out = model(**batch)
    print("Initial loss:", out.loss.item())

EPOCHS = 1

for epoch in range(EPOCHS):
    total_loss = 0
    for step, batch in enumerate(train_loader):
        batch = {k: v.to(model.device) for k, v in batch.items()}

        outputs = model(**batch)
        loss = outputs.loss

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        total_loss += loss.item()

        if step % 50 == 0:
            print(f"Epoch {epoch} Step {step} Loss {loss.item():.4f}")

    print(f"Epoch {epoch} Avg loss:", total_loss / len(train_loader))
```

Save Adapter

```
save_dir = "gemma2b_lora_adapter"
model.save_pretrained(save_dir)
tokenizer.save_pretrained(save_dir)

print("Adapter saved to", save_dir)
```

Model Prediction/ Demo

Load base model + adapter (untuk inference)

```
from peft import PeftModel

base_model_id = "Sahabat-AI/gemma2-9b-cpt-sahabatai-v1-base"
adapter_dir = "gemma9b_lora_adapter"

tokenizer = AutoTokenizer.from_pretrained(base_model_id)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

base_model = AutoModelForCausalLM.from_pretrained(
    base_model_id,
    torch_dtype=torch.float16,
    device_map="auto"
)

model = PeftModel.from_pretrained(base_model, adapter_dir)
model.eval()
```

Fungsi generate jawaban

```
def generate_answer(sample, max_new_tokens=200):
    prompt = format_prompt(sample["category"], sample["instruction"],
sample.get("input",""))
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    with torch.no_grad():
        out = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            do_sample=False,
            temperature=0.0,
            pad_token_id=tokenizer.eos_token_id
        )

    text = tokenizer.decode(out[0], skip_special_tokens=True)

    # ambil bagian setelah "### Response:"
    marker = "### Response:"
    if marker in text:
```

```
        return text.split(marker, 1)[1].strip()
    return text.strip()
```

Generate untuk test set + simpan

```
import json

pred_test = []
for s in test_data:
    s2 = dict(s)
    s2["model_response"] = generate_answer(s2, max_new_tokens=200)
    pred_test.append(s2)

out_path = "test_predictions.json"
with open(out_path, "w", encoding="utf-8") as f:
    json.dump(pred_test, f, ensure_ascii=False, indent=2)

print("Saved:", out_path)
```

Model Evaluation (LLM-based Scoring)

Setup Judge

Model yang kami gunakan adalah Qwen karena Qwen mendukung Multi bahasa, salah satunya adalah bahasa indonesia yang kami gunakan pada LLM Sahabat AI Gemma

```
!pip -q install -U huggingface_hub
import os
from huggingface_hub import InferenceClient

#HF_TOKEN = os.environ.get("HF_TOKEN")
judge_model = "Qwen/Qwen2.5-7B-Instruct"

judge = InferenceClient(model=judge_model, token=HF_TOKEN)

import re
import json
import numpy as np

def judge_prompt(sample):
    return f"""
Nilai jawaban model berikut dari 0 sampai 5.
Balas HANYA JSON valid: {{"score":0-5,"reason":"singkat"}}

Instruction: {sample["instruction"]}
Input: {sample.get("input","")}
Expected: {sample["output"]}
Model Response: {sample["model_response"]}
""".strip()
```

```
def parse_score(text):
    m = re.search(r"\{.*\}", text, flags=re.DOTALL)
    if not m:
        return None
    try:
        obj = json.loads(m.group(0))
        score = int(obj.get("score"))
        if 0 <= score <= 5:
            return score, str(obj.get("reason","")).strip()
    except:
        return None
    return None
```

Menjalankan scoring + stats

```
scored = []
invalid = 0

for s in pred_test:
    prompt = judge_prompt(s)
    resp = judge.text_generation(prompt, max_new_tokens=120,
temperature=0.0)
    parsed = parse_score(resp)

    s2 = dict(s)
    if parsed is None:
        invalid += 1
        s2["judge_score"] = None
        s2["judge_reason"] = "INVALID_JUDGE_OUTPUT"
    else:
        s2["judge_score"], s2["judge_reason"] = parsed

    scored.append(s2)

valid_scores = [x["judge_score"] for x in scored if x["judge_score"]
is not None]
stats = {
    "n_total": len(scored),
    "n_valid": len(valid_scores),
    "n_invalid": invalid,
    "mean": float(np.mean(valid_scores)) if valid_scores else None,
    "median": float(np.median(valid_scores)) if valid_scores else
None,
    "dist": {str(k): sum(1 for v in valid_scores if v == k) for k in
range(6)}
}

print(stats)
```

```
out_path2 = "test_with_judge.json"
with open(out_path2, "w", encoding="utf-8") as f:
    json.dump(scored, f, ensure_ascii=False, indent=2)

print("Saved:", out_path2)
```

```python
import streamlit as st
from llm_hf import LLMClientHF

st.set_page_config(page_title="Chatbot PMB UAJY")

@st.cache_resource
def load_llm():
    return LLMClientHF(
        model_name= "diordty/gemma2b-lora-pmb-uajy"
    )

llm = load_llm()

def format_prompt(question, history=""):
    return f"""### Category: PMB_Umum
### Instruction:
Jawablah pertanyaan berikut berdasarkan informasi resmi PMB Universitas Atma Jaya Yogyakarta.

### Input:
{history}
{question}

### Response:
""".strip()

st.title("pµĤ‟ Chatbot PMB UAJY")
st.caption("Chatbot berbasis LLM hasil fine-tuning")

if "messages" not in st.session_state:
    st.session_state.messages = []

for role, content in st.session_state.messages:
    with st.chat_message(role):
        st.markdown(content)

user_input = st.chat_input("Tanyakan seputar PMB UAJY...")

if user_input:
    st.session_state.messages.append(("user", user_input))

    history_text = ""
    MAX_TURNS = 3
    for role, msg in st.session_state.messages[-2*MAX_TURNS:-1]:
        history_text += f"{role.capitalize()}: {msg}\n"

    prompt = format_prompt(user_input, history_text)

    try:
        response = llm.ask(prompt, temperature=0.0, max_tokens=128)
    except Exception:
        response = "Maaf, sistem sedang mengalami gangguan."

    st.session_state.messages.append(("assistant", response))

    with st.chat_message("assistant"):
        st.markdown(response)
```

```python
from dataclasses import dataclass
from typing import Optional
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import torch

@dataclass
class LLMClientHF:
    model_name: str = "diordty/gemma2b-lora-pmb-uajy"
    device: Optional[str] = None

    def __post_init__(self):
        if self.device is None:
            self.device = "cuda" if torch.cuda.is_available() else "cpu"
        print(f"Loading {self.model_name} on {self.device}...")
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_name, use_fast=True)

        self.model = AutoModelForCausalLM.from_pretrained(
            self.model_name,
            torch_dtype=torch.float16 if self.device == "cuda" else torch.float32,
            device_map="auto" if self.device == "cuda" else None,
        )

        if self.tokenizer.pad_token is None:
            self.tokenizer.pad_token = self.tokenizer.eos_token
            self.model.config.pad_token_id = self.tokenizer.pad_token_id

        self.generator = pipeline(
            "text-generation",
            model=self.model,
            tokenizer=self.tokenizer,
        )
    def ask(self, prompt: str, system: str | None = None,
            max_tokens: int = 16, temperature: float = 0.0) -> str:
        full_prompt = f"<<SYS>>\n{system}\n<</SYS>>\n\n{prompt}" if system else prompt
        do_sample = temperature > 0.0

        out = self.generator(
            full_prompt,
            max_new_tokens=max_tokens,
            do_sample=do_sample,
            temperature=temperature if do_sample else None,
            truncation=True,
            pad_token_id=self.tokenizer.pad_token_id,
            eos_token_id=self.tokenizer.eos_token_id,
        )
        gen = out[0]["generated_text"]
        if gen.startswith(full_prompt):
            gen = gen[len(full_prompt):]
        return gen.strip()

    def ask_many(self, prompts, system=None, max_tokens=16, temperature=0.0, batch_size=16):
        if system:
            prompts = [f"<<SYS>>\n{system}\n<</SYS>>\n\n{p}" for p in prompts]
        do_sample = temperature > 0.0
        outs = self.generator(
            prompts,
            max_new_tokens=max_tokens,
            batch_size=batch_size,
            do_sample=do_sample,
            temperature=temperature if do_sample else None,
            truncation=True,
            pad_token_id=self.tokenizer.pad_token_id,
            eos_token_id=self.tokenizer.eos_token_id,
        )
        gens = []
        for full, prompt in zip(outs, prompts):
            txt = full[0]["generated_text"]
            if txt.startswith(prompt):
                txt = txt[len(prompt):]
            gens.append(txt.strip())
        return gens
```