```
!pip -q install -U transformers datasets peft accelerate
import torch, transformers, datasets, peft, accelerate
print("Torch:", torch.__version__)
print("Transformers:", transformers.__version__)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.4/41.4 kB 2.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 54.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 503.6/503.6 kB 21.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.8/42.8 MB 13.4 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is th
pylibcudf-cu12 25.6.0 requires pyarrow<20.0.0a0,>=14.0.0; platform_machine == "x86_64", but you have pyarrow 21.0.0 which is in
cudf-cu12 25.6.0 requires pyarrow<20.0.0a0,>=14.0.0; platform_machine == "x86_64", but you have pyarrow 21.0.0 which is incompa
Torch: 2.8.0+cu126
Transformers: 4.57.0
```

```
from dataclasses import dataclass
from typing import Optional

@dataclass
class TrainConfig:
    model_name: str = "facebook/opt-350m"   # safer for Colab; change to opt-1.3b if GPU is strong
    train_path: str = "data/processed/train.jsonl"
    val_path: str = "data/processed/validation.jsonl"
    test_path: str = "data/processed/test.jsonl"
    output_dir: str = "alpacare-lora"
    num_train_epochs: float = 1.0
    per_device_train_batch_size: int = 1
    per_device_eval_batch_size: int = 1
    gradient_accumulation_steps: int = 16
    learning_rate: float = 2e-4
    warmup_ratio: float = 0.03
    logging_steps: int = 50
    save_steps: int = 2000
    fp16: bool = True
    max_length: int = 512  # set 256 if OOM
    seed: int = 42
    max_train: Optional[int] = 5000
    max_val: Optional[int] = 500
    max_test: Optional[int] = 500

cfg = TrainConfig()
cfg
```

```
TrainConfig(model_name='facebook/opt-350m', train_path='data/processed/train.jsonl',
val_path='data/processed/validation.jsonl', test_path='data/processed/test.jsonl', output_dir='alpacare-lora',
num_train_epochs=1.0, per_device_train_batch_size=1, per_device_eval_batch_size=1, gradient_accumulation_steps=16,
learning_rate=0.0002, warmup_ratio=0.03, logging_steps=50, save_steps=2000, fp16=True, max_length=512, seed=42,
max_train=5000, max_val=500, max_test=500)
```

```
import os
from datasets import load_dataset, Dataset

def _standardize_record(rec):
    instr_keys = ["instruction","question","prompt"]
    out_keys   = ["output","answer","response","target"]
    instruction = next((rec.get(k) for k in instr_keys if rec.get(k)), None)
    output      = next((rec.get(k) for k in out_keys   if rec.get(k)), None)
    if not instruction or not output:
        return None
    return {"prompt": f"Instruction: {instruction}\n\nResponse:", "response": output}

def prepare_data_if_needed(cfg):
    if all(os.path.exists(p) for p in [cfg.train_path, cfg.val_path, cfg.test_path]):
        print("Found JSONL splits.")
        return
    raw = load_dataset("lavita/AlpaCare-MedInstruct-52k")
    base = raw["train"].map(_standardize_record).filter(lambda r: r["prompt"] is not None)
    dsd = base.train_test_split(test_size=0.05, seed=cfg.seed)
    train_full, test = dsd["train"], dsd["test"]
    val_size = 0.05 / 0.95
    dsd_tv = train_full.train_test_split(test_size=val_size, seed=cfg.seed)
    train, val = dsd_tv["train"], dsd_tv["test"]
    train.to_json(cfg.train_path)
    val.to_json(cfg.val_path)
    test.to_json(cfg.test_path)
    print(" Saved JSONL splits.")
```

```
prepare_data_if_needed(cfg)
```

```
/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 i
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(
README.md: 100%                                          944/944 [00:00<00:00, 69.2kB/s]
data/train-00000-of-00001-297892d5d4e8a0(…): 100%                          36.7M/36.7M [00:01<00:00, 31.8MB/s]
Generating train split: 100%                                   52002/52002 [00:00<00:00, 90505.08 examples/s]
Map: 100%                                          52002/52002 [00:06<00:00, 9003.58 examples/s]
Filter: 100%                                       52002/52002 [00:02<00:00, 17313.28 examples/s]
Creating json from Arrow format: 100%                                   47/47 [00:02<00:00, 18.98ba/s]
Creating json from Arrow format: 100%                                   3/3 [00:00<00:00,  8.97ba/s]
Creating json from Arrow format: 100%                                   3/3 [00:00<00:00,  9.14ba/s]
 Saved JSONL splits.
```

```
from transformers import AutoTokenizer, AutoModelForCausalLM
from peft import LoraConfig, get_peft_model

tokenizer = AutoTokenizer.from_pretrained(cfg.model_name, use_fast=True)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(cfg.model_name)

lora_config = LoraConfig(
    r=16, 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.gradient_checkpointing_enable()
model.config.use_cache = False
model.print_trainable_parameters()
```

```
tokenizer_config.json: 100%                                   685/685 [00:00<00:00, 18.1kB/s]
config.json: 100%                                    644/644 [00:00<00:00, 23.7kB/s]
vocab.json:     899k/? [00:00<00:00, 19.4MB/s]
merges.txt:     456k/? [00:00<00:00, 18.9MB/s]
special_tokens_map.json: 100%                                   441/441 [00:00<00:00, 52.7kB/s]
pytorch_model.bin: 100%                                   663M/663M [00:13<00:00, 52.6MB/s]
model.safetensors: 100%                                   662M/662M [00:09<00:00, 87.0MB/s]
generation_config.json: 100%                                   137/137 [00:00<00:00, 3.24kB/s]
trainable params: 1,572,864 || all params: 332,769,280 || trainable%: 0.4727
```

```
from datasets import load_dataset

train_ds = load_dataset("json", data_files=cfg.train_path, split="train")
val_ds   = load_dataset("json", data_files=cfg.val_path, split="train")

def format_example(example):
    return {"text": example["prompt"] + "\n" + example["response"]}

train_fmt = train_ds.map(format_example, remove_columns=train_ds.column_names)
val_fmt   = val_ds.map(format_example, remove_columns=val_ds.column_names)

def tokenize(examples):
    out = tokenizer(examples["text"], truncation=True, max_length=cfg.max_length)
    out["labels"] = out["input_ids"].copy()
    return out

train_tok = train_fmt.map(tokenize, batched=True, remove_columns=train_fmt.column_names)
val_tok   = val_fmt.map(tokenize, batched=True, remove_columns=val_fmt.column_names)
```

```
Generating train split:    46800/0  [00:02<00:00, 14031.63 examples/s]

Generating train split:    2601/0  [00:00<00:00, 14684.07 examples/s]

Map: 100%                                        46800/46800 [00:08<00:00, 7436.00 examples/s]

Map: 100%                                        2601/2601 [00:00<00:00, 8500.61 examples/s]

Map: 100%                                        46800/46800 [01:07<00:00, 630.31 examples/s]

Map: 100%                                        2601/2601 [00:06<00:00, 381.01 examples/s]
```

```
from transformers import DataCollatorForLanguageModeling, Trainer, TrainingArguments

data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

args = TrainingArguments(
    output_dir=cfg.output_dir,
    learning_rate=cfg.learning_rate,
    per_device_train_batch_size=cfg.per_device_train_batch_size,
    per_device_eval_batch_size=cfg.per_device_eval_batch_size,
    gradient_accumulation_steps=cfg.gradient_accumulation_steps,
    num_train_epochs=cfg.num_train_epochs,
    warmup_ratio=cfg.warmup_ratio,
    logging_steps=cfg.logging_steps,
    save_steps=cfg.save_steps,
    fp16=cfg.fp16,
    report_to="none",
    remove_unused_columns=False
)

# Ensure model parameters require gradients
for param in model.parameters():
    param.requires_grad = True

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_tok,
    eval_dataset=None,
    data_collator=data_collator,
)

trainer.train()
trainer.save_model(cfg.output_dir)
tokenizer.save_pretrained(cfg.output_dir)
print(" LoRA adapter saved to", cfg.output_dir)
```

[1213/2925 53:05 < 1:15:03, 0.38 it/s, Epoch 0.41/1]

Step	Training Loss
50	2.032900
100	2.098900
150	2.125400
200	2.116300
250	2.107300
300	2.077100
350	2.045400
400	2.038100
450	1.981400
500	1.956300
550	1.969700
600	1.914500
650	1.912800
700	1.889000
750	1.887400
800	1.887200
850	1.868500
900	1.825300
950	1.844400
1000	1.809600
1050	1.801600
1100	1.776000
1150	1.776100
1200	1.760100

[2925/2925 2:07:30, Epoch 1/1]

Step	Training Loss
50	2.032900
100	2.098900
150	2.125400
200	2.116300
250	2.107300
300	2.077100
350	2.045400
400	2.038100
450	1.981400

```
import re
import torch
from peft import PeftModel
from transformers import AutoModelForCausalLM

# load base + adapters (same as you did)
base = AutoModelForCausalLM.from_pretrained(cfg.model_name)
ft = PeftModel.from_pretrained(base, cfg.output_dir)
ft.eval()

#Safety config
DISCLAIMER = "\n\n Disclaimer: This is for educational purposes only. Please consult a doctor."

FORBIDDEN = [
    "diagnose","diagnosis","prescribe","prescription",
    "dose","dosage","mg","tablet","capsule","antibiotic","steroid"
]
EMERGENCY = [
    "heart pain","chest pain","can't breathe","cannot breathe",
    "shortness of breath","unconscious","bleeding heavily","stroke","seizure"
]
```

```
def is_emergency(text: str) -> bool:
    t = text.lower()
    return any(k in t for k in EMERGENCY)

def safety_filter(text: str) -> bool:
    t = text.lower()
    return any(k in t for k in FORBIDDEN)

# Helper: clean repetition & trim
def clean_text(t: str) -> str:
    # collapse obvious repeated clauses (A A A)
    t = re.sub(r'(?:\b[\w,;:()\'"-]{2,}\b[ \t]*){1,}', lambda m: m.group(0), t)
    # remove exact sentence repeats (case-insensitive)
    sentences = re.split(r'(?<=[.!?])\s+', t)
    seen = set()
    cleaned = []
    for s in sentences:
        s2 = s.strip()
        if not s2:
            continue
        k = s2.lower()
        if k not in seen:
            cleaned.append(s2)
            seen.add(k)
    t = " ".join(cleaned)
    # small tidy-ups
    t = re.sub(r'\s+', ' ', t).strip()
    return t

# Generation with safe defaults
def generate_raw(prompt: str, max_new_tokens: int = 160) -> str:
    inputs = tokenizer(prompt, return_tensors="pt")
    inputs = {k: v.to(ft.device) for k, v in inputs.items()}

    gen_kwargs = dict(
        max_new_tokens=max_new_tokens,
        do_sample=True,
        temperature=0.7,
        top_p=0.9,
        repetition_penalty=1.2,    # combats loops
        no_repeat_ngram_size=4,    # blocks n-gram repeats
        early_stopping=True,
        eos_token_id=tokenizer.eos_token_id,
        pad_token_id=tokenizer.pad_token_id,
    )

    with torch.no_grad():
        out = ft.generate(**inputs, **gen_kwargs)

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

    # cut off any prompt echo — keep only what comes after "Response:"
    if "Response:" in text:
        text = text.split("Response:", 1)[1].strip()

    return clean_text(text)

# Final assistant pipeline
def medical_assistant(user_input: str) -> str:
    # 0) emergency triage on INPUT
    if is_emergency(user_input):
        return " This seems urgent. Please call emergency services immediately." + DISCLAIMER

    # 1) block dosage/prescription queries on INPUT too
    if safety_filter(user_input):
        return " This goes beyond my scope (no prescriptions or dosages). Please consult a doctor." + DISCLAIMER

    # 2) safe instruction prompt (keeps the model in-bounds)
    prompt = (
        "Instruction: Provide brief, safe, non-diagnostic general guidance in 3-5 bullet points. "
        "Do NOT prescribe medicines or give dosages. Suggest home-care basics and when to seek care. "
        "Use plain language suitable for a layperson.\n"
        f"Input: {user_input}\n\nResponse:"
    )

    raw = generate_raw(prompt, max_new_tokens=140)

    # 3) post-output safety check as a second guardrail
    if safety_filter(raw):
        return " This goes beyond my scope (no prescriptions or dosages). Please consult a doctor." + DISCLAIMER
    # 4) structure a little for readability
    # if the model didn't produce bullets, add simple formatting
    if "•" not in raw and "-" not in raw:
        raw = "- " + raw.replace(". ", ".\n- ")

    return raw.strip() + DISCLAIMER
```

```
print(medical_assistant("I have mild fever and body ache. What can I do at home?"))
print()
print(medical_assistant("I have chest pain and shortness of breath."))
print()
print(medical_assistant("What dosage of ibuprofen 400 mg should I take?"))
```

```
The following generation flags are not valid and may be ignored: ['early_stopping']. Set `TRANSFORMERS_VERBOSITY=info` for more
- Please advise on what steps you are taking to treat the fever/body aches as well as other symptoms of fever and body pain tha

⚠️ Disclaimer: This is for educational purposes only. Please consult a doctor.

🚨 This seems urgent. Please call emergency services immediately.

⚠️ Disclaimer: This is for educational purposes only. Please consult a doctor.

⚠️ This goes beyond my scope (no prescriptions or dosages). Please consult a doctor.

⚠️ Disclaimer: This is for educational purposes only. Please consult a doctor.
```

```
!pip -q install gradio
import gradio as gr

def ui_fn(text):
    text = (text or "").strip()
```