```
from huggingface_hub import login
from google.colab import userdata

# Log in using your HF_TOKEN secret
login(token=userdata.get("HF_TOKEN"))
```

```
# ================================================
# 🩺 AlpaCare Medical Instruction Assistant (LoRA)
# Fine-tune meta-llama/Llama-3.2-1B-Instruct (4-bit)
# ================================================

!pip install -qU transformers==4.44.0 datasets==2.16.0 peft accelerate bitsandbytes
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.7/43.7 kB 4.1 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.5/9.5 MB 67.5 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 507.1/507.1 kB 21.4 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.1/60.1 MB 13.2 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 10.7 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 166.4/166.4 kB 13.0 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 70.0 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 10.2 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
gcsfs 2025.3.0 requires fsspec==2025.3.0, but you have fsspec 2023.10.0 which is incompatible.
```

```
import torch
from datasets import load_dataset
from transformers import AutoModelForCausalLM, AutoTokenizer, TrainingArguments, Trainer
from peft import LoraConfig, get_peft_model
from google.colab import userdata
import pandas as pd

print("✅ Libraries imported successfully!")
```

```
✅ Libraries imported successfully!
```

```
print("📥 Loading dataset...")
dataset = load_dataset("lavita/AlpaCare-MedInstruct-52k")["train"]

print(f"📊 Original dataset size: {len(dataset)}")

# Convert to pandas for cleaning
df = dataset.to_pandas()

# 🧹 Basic Cleaning
df = df.dropna(subset=["instruction", "output"])      # remove missing instructions/outputs
df = df.drop_duplicates(subset=["instruction", "input", "output"])  # remove duplicates
df = df.reset_index(drop=True)

print(f"✅ Cleaned dataset size: {len(df)}")

# Convert back to Hugging Face Dataset
from datasets import Dataset
dataset_clean = Dataset.from_pandas(df)
```

```
📥 Loading dataset...
Downloading readme: 100%  944/944 [00:00<00:00, 115kB/s]
Downloading data: 100%  36.7M/36.7M [00:00<00:00, 33.9MB/s]
Generating train split: 100%  52002/52002 [00:00<00:00, 180039.28 examples/s]
📊 Original dataset size: 52002
✅ Cleaned dataset size: 52002
```

```
print("📥 Loading dataset...")
dataset = load_dataset("lavita/AlpaCare-MedInstruct-52k")["train"]

print(f"📊 Original dataset size: {len(dataset)}")

import pandas as pd
df = dataset.to_pandas()

# 🧹 Basic cleaning
df = df.dropna(subset=["instruction", "output"])           # remove rows missing key fields
df = df.drop_duplicates(subset=["instruction","input","output"])  # remove duplicate samples
df = df.reset_index(drop=True)
```

```
print(f"✅ After initial cleaning: {len(df)} samples")
```

```
📥 Loading dataset...
📊 Original dataset size: 52002
✅ After initial cleaning: 52002 samples
```

```
import re

def clean_text(text):
    """Clean and normalize text fields."""
    if not isinstance(text, str):
        return ""
    text = text.strip()
    text = re.sub(r"\s+", " ", text)
    text = re.sub(r'[“”]', '"', text)
    text = re.sub(r"[‘’]", "'", text)
    text = text.replace("", "")
    return text

# Apply cleaning to text columns
for col in ["instruction", "input", "output"]:
    df[col] = df[col].apply(clean_text)

# Filter out empty or too short samples
df = df[df["instruction"].str.len() > 5]
df = df[df["output"].str.len() > 5]

print(f"✅ After text normalization: {len(df)} samples")

# Show a sample
df.sample(2)
```

```
✅ After text normalization: 52000 samples
```

	output	input	instruction
14507	Preparing for medical school exams can be chal...	<noinput>	Explain to a student how to prepare for medica...

```
from datasets import Dataset

# Limit for Colab runtime safety
MAX_SAMPLES = 2000
dataset_subset = Dataset.from_pandas(df.iloc[:MAX_SAMPLES])

# Split: 90% train, 5% val, 5% test
train_test = dataset_subset.train_test_split(test_size=0.10, seed=42)
val_test = train_test["test"].train_test_split(test_size=0.50, seed=42)

train_dataset = train_test["train"]
val_dataset = val_test["train"]
test_dataset = val_test["test"]

print(f"✅ Dataset split complete:")
print(f"   🧠 Train: {len(train_dataset)}")
print(f"   🧪 Val:   {len(val_dataset)}")
print(f"   🧩 Test:  {len(test_dataset)}")
```

```
✅ Dataset split complete:
   🧠 Train: 1800
   🧪 Val:   100
   🧩 Test:  100
```

```
base_model = "meta-llama/Llama-3.2-1B-Instruct"
hf_token = userdata.get("HF_TOKEN")

print("🔤 Loading tokenizer...")
tokenizer = AutoTokenizer.from_pretrained(base_model, use_fast=True, token=hf_token)

# Fix padding token issue
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"

print("✅ Tokenizer loaded and fixed.")
```

```
🔤 Loading tokenizer...
tokenizer_config.json: 100%  54.5k/54.5k [00:00<00:00, 5.07MB/s]
tokenizer.json: 100%  9.09M/9.09M [00:01<00:00, 7.18MB/s]
special_tokens_map.json: 100%  296/296 [00:00<00:00, 32.1kB/s]
✅ Tokenizer loaded and fixed.
```

```
max_length = 512

def tokenize_fn(example):
    instruction = example.get("instruction", "")
    context = example.get("input", "")
    output = example.get("output", "")

    # Format prompt
    if context.strip():
        prompt_text = f"Instruction: {instruction}\nInput: {context}\nAnswer:"
    else:
        prompt_text = f"Instruction: {instruction}\nAnswer:"

    prompt_ids = tokenizer(prompt_text, truncation=True, max_length=max_length, add_special_tokens=False)["input_ids"]
    response_ids = tokenizer(output, truncation=True, max_length=max_length, add_special_tokens=False)["input_ids"]

    input_ids = prompt_ids + response_ids + [tokenizer.eos_token_id]
    labels = [-100]*len(prompt_ids) + response_ids + [tokenizer.eos_token_id]

    return {
        "input_ids": input_ids,
        "labels": labels,
        "attention_mask": [1]*len(input_ids)
    }

print("🔄 Tokenizing datasets...")
tokenized_train = train_dataset.map(tokenize_fn, remove_columns=["instruction","input","output"])
tokenized_val   = val_dataset.map(tokenize_fn, remove_columns=["instruction","input","output"])
tokenized_test  = test_dataset.map(tokenize_fn, remove_columns=["instruction","input","output"])
print("✅ Tokenization complete!")
```

```
🔄 Tokenizing datasets...
Map: 100%  1800/1800 [00:02<00:00, 925.66 examples/s]
Map: 100%  100/100 [00:00<00:00, 710.90 examples/s]
Map: 100%  100/100 [00:00<00:00, 823.61 examples/s]
✅ Tokenization complete!
```

```
from transformers import AutoModelForCausalLM

print("⚙️ Loading model in FP16 mode")

model = AutoModelForCausalLM.from_pretrained(
    "meta-llama/Llama-3.2-1B-Instruct",
    device_map="auto",
    torch_dtype=torch.float16,
    trust_remote_code=True
)

print("✅ Model loaded in FP16 mode!")
```

```
⚙️ Loading model in FP16 mode
config.json: 100%  877/877 [00:00<00:00, 55.1kB/s]
model.safetensors: 100%  2.47G/2.47G [00:36<00:00, 99.0MB/s]
generation_config.json: 100%  189/189 [00:00<00:00, 22.6kB/s]
✅ Model loaded in FP16 mode!
```

```
from peft import LoraConfig, get_peft_model

print("🔧 Applying LoRA configuration...")

lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=["q_proj","v_proj"],  # LoRA applied to Q and V projection layers
    lora_dropout=0.05,
    bias="none",
```

```
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, lora_config)

print("✅ LoRA applied successfully!")
```

```
🔧 Applying LoRA configuration...
✅ LoRA applied successfully!
```

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./alpacare-lora-llama1b-fp16",
    per_device_train_batch_size=1,
    gradient_accumulation_steps=2,  # effective batch size = 2
    learning_rate=2e-4,
    num_train_epochs=5,
    logging_steps=20,
    save_steps=100,
    save_total_limit=2,
    fp16=True,                      # FP16 training
    report_to="none"                 # avoids WandB logging if not needed
)

import torch
torch.cuda.empty_cache()

print("✅ Training arguments set.")
```

```
✅ Training arguments set.
```

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,  # previously tokenized training dataset
    eval_dataset=tokenized_val      # previously tokenized validation dataset
)

print("✅ Trainer initialized and ready for fine-tuning.")
```

```
✅ Trainer initialized and ready for fine-tuning.
```

```
print("🚀 Starting LoRA fine-tuning in FP16 mode...")
trainer.train()
print("✅ Training complete!")
```