start

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install -q transformers accelerate bitsandbytes huggingface_hub
datasets
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.1/76.1 MB 10.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 491.4/491.4 kB 38.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 11.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 193.6/193.6 kB 17.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 13.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 1.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 116.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 99.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 57.8 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 12.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 6.8 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 7.0 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 95.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 20.7 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 the source of
the following dependency conflicts.
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2025.3.0
which is incompatible.
```

```
!pip install -U bitsandbytes

Requirement already satisfied: bitsandbytes in
/usr/local/lib/python3.11/dist-packages (0.45.5)
Requirement already satisfied: torch<3,>=2.0 in
/usr/local/lib/python3.11/dist-packages (from bitsandbytes)
(2.6.0+cu124)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.11/dist-packages (from bitsandbytes) (2.0.2)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (4.13.2)
Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (3.4.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (3.1.6)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127
in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
```

```
>bitsandbytes) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (12.4.127)
Requirement already satisfied: triton==3.2.0 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in
/usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0-
>bitsandbytes) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1-
>torch<3,>=2.0->bitsandbytes) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch<3,>=2.0-
>bitsandbytes) (3.0.2)
```

model

```
model_id = "meta-llama/Llama-2-7b-hf"

from huggingface_hub import notebook_login
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer,
BitsAndBytesConfig
from transformers import TrainingArguments, Trainer
from peft import prepare_model_for_kbit_training, LoraConfig,
get_peft_model
from datasets import load_dataset
from transformers import DataCollatorForLanguageModeling
import json



from huggingface_hub import notebook_login
notebook_login()
```


```
{"model_id":"e4853d4f0b374df7a606d64aa356877d","version_major":2,"vers
ion_minor":0}
```

model 2

```
# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

```
/usr/local/lib/python3.11/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":"9e6dcb5aacae478b8ac151b1fec91ee8","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"a54acfe0e77e4ee380fbeaf41f863770","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"a1d8b041d6464f6bb737b7f5bb4cd2ee","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"b36a17dc6470455b9c21776182a9f098","version_major":2,"vers
ion_minor":0}
```


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


# Define tokenization function
def tokenize_function(examples):
    return tokenizer(
        examples["text"],
        truncation=True,
        max_length=512,
        padding="max_length",
        return_tensors="pt"
    )

# Load model with 8-bit precision (needs less memory)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map="auto",
    load_in_8bit=True,  # For lower RAM usage (needs `bitsandbytes`)
    torch_dtype=torch.float16,
)
```

{"model_id":"142d7e2334374bfe819e8eb60b112597","version_major":2,"version_minor":0}

```
The `load_in_4bit` and `load_in_8bit` arguments are deprecated and will be removed in the future versions. Please, pass a `BitsAndBytesConfig` object in `quantization_config` argument instead.
```

{"model_id":"9aea226884254d399985a5b875812725","version_major":2,"version_minor":0}

{"model_id":"0344ffd90b7346968145af7f5f3c5130","version_major":2,"version_minor":0}

{"model_id":"eb562b30b80a4ead92d87b61f2bd80b6","version_major":2,"version_minor":0}

{"model_id":"9104ab2751964cfe94d791c2d51a225c","version_major":2,"version_minor":0}

{"model_id":"5ec60edaa71b4320ab702fa188ececfc","version_major":2,"version_minor":0}

{"model_id":"d37e71d0c6664dab929c9fca57d9b3e4","version_major":2,"version_minor":0}

data for fine tune

```
# Load the dataset that you've already formatted
dataset = load_dataset('json',
data_files='/content/drive/MyDrive/llm/medqu.jsonl')
print(f"Dataset loaded with {len(dataset['train'])} examples")
```

{"model_id":"ffc73c243ab548679adcd8ba091f845c","version_major":2,"version_minor":0}

```
Dataset loaded with 16407 examples

dataset

DatasetDict({
    train: Dataset({
        features: ['text'],
        num_rows: 16407
    })
})

# Display some sample data from the dataset
print(dataset['train'][0:1])
```

```
{'text': ['<s>[INST] <<SYS>>\nProvide accurate, concise answers to medical questions.\n<</SYS>>\n\nQuestion: Who is at risk for Lymphocytic Choriomeningitis (LCM)? ? [/INST] Answer: LCMV infections can occur after exposure to fresh urine, droppings, saliva, or nesting materials from infected rodents.  Transmission may also occur when these materials are directly introduced into broken skin, the nose, the eyes, or the mouth, or presumably, via the bite of an infected rodent. Person-to-person transmission has not been reported, with the exception of vertical transmission from infected mother to fetus, and rarely, through organ transplantation.</s>']}
```

```
from datasets import Dataset

# Select the first 2000 examples from the training split
subset_dataset = Dataset.from_dict(dataset['train'][0:2000])

# Now 'subset_dataset' contains only 2000 examples
print(f"Subset dataset created with {len(subset_dataset)} examples")

# Proceed with training using subset_dataset instead of dataset
# trainer = Trainer(model=model, args=training_args,
train_dataset=subset_dataset)
# trainer.train()
```

```
Subset dataset created with 2000 examples
```

```
dataset=subset_dataset
```

```
len(dataset)
```

```
2000
```

```
# Tokenize the dataset
tokenized_dataset = dataset.map(
    tokenize_function,
    batched=True,
    remove_columns=dataset.column_names
)
```


```
{"model_id":"a9730b14bbd141a69a257358b2b9d5c6","version_major":2,"vers
ion_minor":0}
```


```
tokenized_dataset
```

```
Dataset({
    features: ['input_ids', 'attention_mask'],
    num_rows: 2000
})
```

```
# Split into train/validation sets
split_dataset = tokenized_dataset.train_test_split(test_size=0.1,
seed=42)
```

start finetune

load_in_4bit=True: Load model weights in 4-bit precision instead of standard 32-bit

bnb_4bit_quant_type="nf4": Uses Normal Float 4-bit quantization (optimal for normal weight distributions)

bnb_4bit_compute_dtype=torch.float16: Computations happen in 16-bit for better speed/accuracy balance

bnb_4bit_use_double_quant=True: Applies secondary quantization to save even more memory

```
#  quantization for memory efficiency in
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
)
```

To load the pre-trained model with memory optimization

```
# Load the model with quantization
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=bnb_config,
    device_map="auto",
    torch_dtype=torch.float16,
)
```


```
{"model_id":"e581e947a9974dd89fa51ef596b58e9a","version_major":2,"vers
ion_minor":0}
```


load model

- prepare_model_for_kbit_training: This function prepares a quantized model for training by adding special computation hooks that make it possible to train 4-bit quantized weights.
- LoraConfig: This creates the configuration for LoRA, where I set rank=8 and targeted only query and value projections in the attention layers. It defines how to add small trainable matrices alongside the frozen base model.
- get_peft_model: This applies the LoRA configuration to the model, freezing the original weights and adding the small trainable adapter layers.

```
# Prepare the model for LoRA training
model = prepare_model_for_kbit_training(model)
```

define the parameter-efficient fine-tuning approach

The rank (r) and alpha parameters balance adaptation strength and parameter efficiency

```
# # LoRA configuration for medical domain adaptation of Llama-2
# lora_config = LoraConfig(
#     # Rank of the low-rank decomposition matrices (int)
#     # Lower rank = more memory efficient but less adaptable
#     # Higher rank = better adaptation capacity but more parameters
#     # For medical domain (which has specialized terminology), r=8 is
a good starting point
#     r=8,

#     # Scaling factor for LoRA weights (float)
#     # Controls magnitude of adaptation - higher values mean LoRA has
stronger influence
#     # The ratio alpha/r determines scaling (here 16/8 = 2)
#     # Medical domains may benefit from slightly higher alpha to
learn specialized terms
#     lora_alpha=16,

#     # Which transformer modules to apply LoRA to (List[str])
#     # Targeting attention projections is most effective:
#     # - q_proj: query projections (affects what the model focuses
on)
#     # - v_proj: value projections (affects content representation)
#     # For medical QA, these help adapt both attention patterns and
medical concept representation
#     target_modules=[
#         "q_proj",  # Query projections affects what the model
focuses on
#         "v_proj",  # Value projections (affects content
representatio
#     ],

#     # Dropout probability for LoRA layers (float)
#     # Regularization to prevent overfitting on medical data
#     # 0.05 is conservative - may increase if dataset is small
#     lora_dropout=0.05,

#     # Bias treatment during adaptation (str)
#     # Options: "none", "all", or "lora_only"
#     # "none" keeps original biases frozen (recommended default)
#     bias="none",

#     # Task type specification (str)
#     # "CAUSAL_LM" for autoregressive models like Llama-2
```

```
#     # Ensures proper adaptation for medical text generation
#     task_type="CAUSAL_LM",

#     # Note: For medical applications, you might later add:
#     # modules_to_save=["lm_head"] to also adapt the output layer
#     # if medical terminology differs significantly from base
training
# )

lora_config = LoraConfig(
    r=4,  #  Reduce rank from 8 to limit parameter count
    lora_alpha=8,  #  Reduce scaling factor (keeps alpha/r=2 ratio)
    target_modules=["q_proj", "v_proj"],  # Keep only most critical
modules
    lora_dropout=0.2,  #  Increase dropout significantly
    modules_to_save=[],  # Remove if present - don't adapt base layers
    task_type="CAUSAL_LM"
)
```

- r=

Rank of the low-rank decomposition matrices in LoRA.

Controls how many new trainable weights are added smaller r means fewer parameters and lighter fine-tuning.

- lora_alpha=

Scaling factor applied to LoRA updates.

Determines the strength of the injected LoRA adaptation (effective scale = alpha/r).

- target_modules=

The model layers where LoRA is applied.

Restricts fine-tuning to only the query (q_proj) and value (v_proj) attention projections, saving memory and compute.

- lora_dropout=

Dropout probability before applying LoRA weights.

Randomly drops activations before LoRA updates to prevent overfitting and improve generalization.

- modules_to_save=[]

Extra modules (outside LoRA) to keep trainable.

Keeps this list empty so only LoRA adapters are trained, freezing the rest of the model.

- task_type="CAUSAL_LM"

The training objective/task type.

Configures LoRA for causal language modeling (autoregressive text generation like GPT).

```
# lora_config = LoraConfig(
#     r=8,
#     lora_alpha=16,
#     target_modules=[
#         "q_proj",
#         "v_proj",
#     ],
#     lora_dropout=0.05,
#     bias="none",
#     task_type="CAUSAL_LM",
# )
```

How LoRA Works:

The Math Behind LoRA:

For each target weight matrix W (e.g., in attention layers) LoRA adds a low-rank update: W + ΔW where ΔW = (α/r) × BA B is a d×r matrix and A is an r×k matrix (where r is much smaller than d and k) This parameterizes the update with far fewer parameters

```
# Apply LoRA to the model
model = get_peft_model(model, lora_config)
print(f"Trainable parameters: {model.print_trainable_parameters()}")

trainable params: 4,194,304 || all params: 6,742,609,920 || trainable
%: 0.0622
Trainable parameters: None
```

The printed parameter count (4.2M trainable out of 6.7B total) confirms the efficiency

define how the training process will run

- Small batch size (1) with gradient accumulation (16) simulates larger batch training
- The learning rate (2e-4) is specifically tuned for LoRA fine-tuning
- 8-bit optimizer and mixed precision further reduce memory usage
- Regular evaluation and checkpointing track progress and save the best model

```
# Set up training arguments with the latest parameter syntax
training_args = TrainingArguments(
    output_dir="./medical_lora_model",
    per_device_train_batch_size=1,     # Very small batch size for
limited VRAM
    gradient_accumulation_steps=16,    # Accumulate gradients to
compensate
    warmup_ratio=0.03,                 # Percentage of steps for
warmup instead of fixed steps
```

```
    max_steps=200,                      # Adjust based on your resource
constraints
    learning_rate=2e-4,
    fp16=True,
    logging_steps=10,

    # Updated evaluation parameters
    eval_strategy="steps",              # Updated from
evaluation_strategy
    eval_steps=50,                      # How often to evaluate

    save_strategy="steps",              # Matching the eval strategy
    save_steps=50,                      # How often to save checkpoints

    optim="paged_adamw_8bit",           # Memory-efficient optimizer
    load_best_model_at_end=True,
    report_to=["none"],  # This disables all integrations including
wandb
)
```

prepare batches of training data properly for a causal language model

```
# Data collator for causal language modeling
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False, #Setting mlm=False confirms we're training an
autoregressive model, not a masked LM
)
```

integrate all components into a managed training process

```
#  Initialize trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=split_dataset["train"],
    eval_dataset=split_dataset["test"],
    data_collator=data_collator,
)
```

```
No label_names provided for model class `PeftModelForCausalLM`. Since
`PeftModel` hides base models input arguments, if label_names is not
given, label_names can't be set automatically within `Trainer`. Note
that empty label_names list will be used instead.
```

```
# Start training
trainer.train()
```

```
`use_cache=True` is incompatible with gradient checkpointing. Setting
`use_cache=False`.
```

```
/usr/local/lib/python3.11/dist-packages/torch/_dynamo/eval_frame.py:745: UserWarning: torch.utils.checkpoint: the use_reentrant parameter should be passed explicitly. In version 2.5 we will raise an exception if use_reentrant is not passed. use_reentrant=False is recommended, but if you need to preserve the current default behavior, you can pass use_reentrant=True. Refer to docs for more details on the differences between the two variants.
  return fn(*args, **kwargs)

<IPython.core.display.HTML object>

/usr/local/lib/python3.11/dist-packages/torch/_dynamo/eval_frame.py:745: UserWarning: torch.utils.checkpoint: the use_reentrant parameter should be passed explicitly. In version 2.5 we will raise an exception if use_reentrant is not passed. use_reentrant=False is recommended, but if you need to preserve the current default behavior, you can pass use_reentrant=True. Refer to docs for more details on the differences between the two variants.
  return fn(*args, **kwargs)
/usr/local/lib/python3.11/dist-packages/torch/_dynamo/eval_frame.py:745: UserWarning: torch.utils.checkpoint: the use_reentrant parameter should be passed explicitly. In version 2.5 we will raise an exception if use_reentrant is not passed. use_reentrant=False is recommended, but if you need to preserve the current default behavior, you can pass use_reentrant=True. Refer to docs for more details on the differences between the two variants.
  return fn(*args, **kwargs)
```

here traning is not completed due to lack of gpu

save trained adapter

```
# 1. After training is complete, save the LoRA adapter weights
model.save_pretrained("./medical_lora_adapter")

# 2. Compress the saved model directory into a zip file for easier download
!zip -r medical_lora_adapter.zip ./medical_lora_adapter

# 3. Download the zip file to your local machine using Colab's files utility
from google.colab import files
files.download('medical_lora_adapter.zip')
```