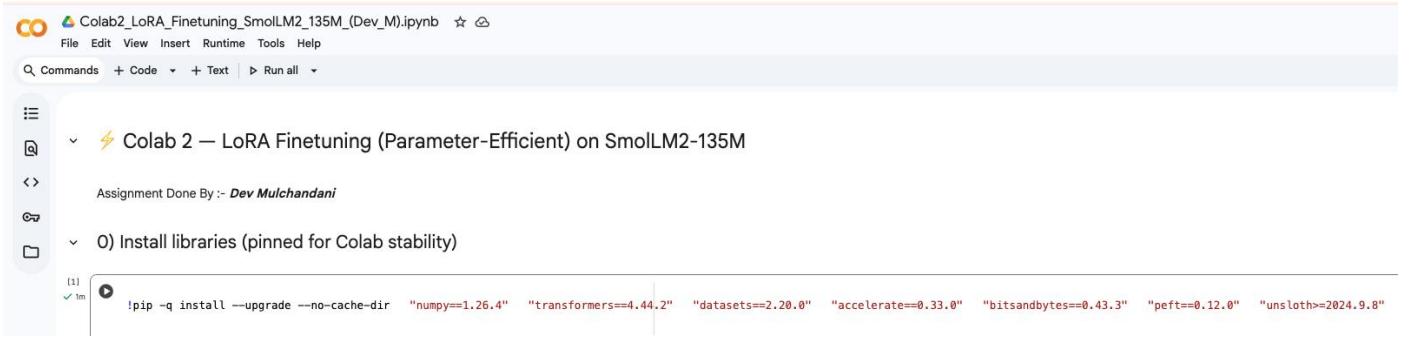
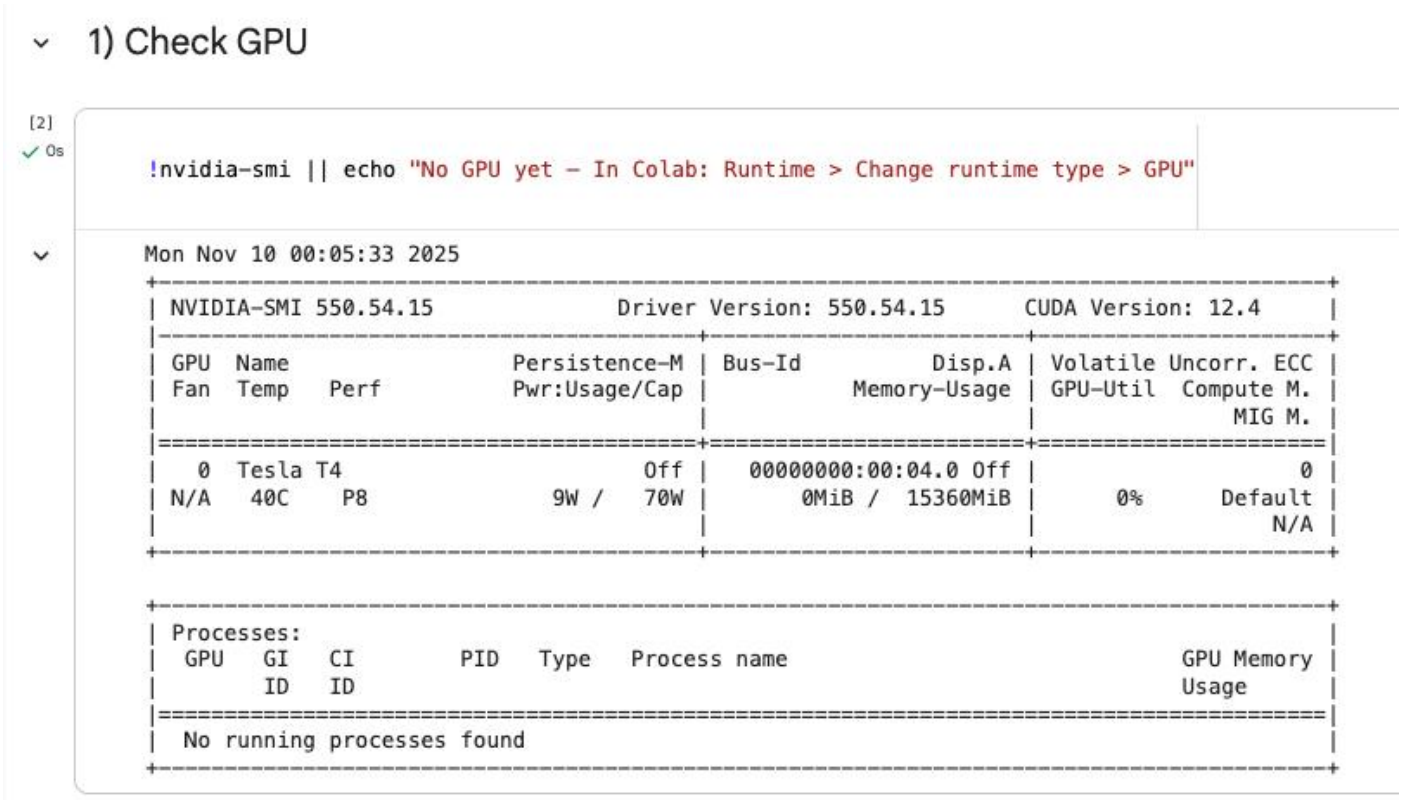
Modern AI with unsloth.ai

❖ Submitted By :- Dev Mulchandani

❖ Colab Notebook :- [Link]

❖ Colab 2 — LoRA Finetuning





2) Imports & versions

[3] ✓ 45s

```
import transformers, torch, os, sys, numpy as np
from datasets import Dataset
from transformers import AutoModelForCausalLM, AutoTokenizer, TrainingArguments, Trainer, DataCollatorForLanguageModeling
from peft import LoraConfig, get_peft_model, PeftModel, TaskType

print("Python:", sys.version)
print("Transformers:", transformers.__version__)
print("Torch:", torch.__version__)
print("NumPy:", np.__version__)
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
Python: 3.12.12 (main, Oct 10 2025, 08:52:57) [GCC 11.4.0]
Transformers: 4.57.1
Torch: 2.8.0+cu126
NumPy: 2.0.2
'cuda'
```

3) Build a tiny toy chat dataset

[4] ✓ 0s

```
base_pairs = [
    ("Explain Python lists to a beginner.", "A list stores items in order like [1, 2, 3]. You can add, remove, and index them."),
    ("Give a tip to study better.", "Use short active recall sessions, then test yourself. Sleep well and space practice."),
    ("What is AI?", "AI means machines performing tasks that usually require human intelligence, like language and vision."),
    ("Write a friendly greeting.", "Hey there! Hope your day is going great 😊"),
    ("How to stay safe online?", "Use strong unique passwords, enable 2FA, avoid unknown links, and keep software updated."),
]
pairs = [(q, a) for q, a in base_pairs for _ in range(12)]

def to_chat_example(instruction, response):
    return {"text": f"### Instruction:\n{instruction}\n\n### Response:\n{response}\n"}

dataset = Dataset.from_list([to_chat_example(q, a) for q, a in pairs])
dataset
```

```
Dataset({
    features: ['text'],
    num_rows: 60
})
```

4) Load base model & tokenizer

[5] ✓ 6s

```
model_name = "HuggingFaceTB/SmolLM2-135M-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=True)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
    torch_dtype=torch.bfloat16 if torch.cuda.is_available() else torch.float32,
)
model.config.use_cache = False
```

```
/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),
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(
tokenizer_config.json: 3.76k/? [00:00<00:00, 81.9kB/s]
vocab.json: 801k/? [00:00<00:00, 4.19MB/s]
merges.txt: 466k/? [00:00<00:00, 5.39MB/s]
tokenizer.json: 2.10M/? [00:00<00:00, 27.7MB/s]
special_tokens_map.json: 100% 655/655 [00:00<00:00, 37.3kB/s]
config.json: 100% 861/861 [00:00<00:00, 24.1kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors: 100% 269M/269M [00:02<00:00, 252MB/s]
generation_config.json: 100% 132/132 [00:00<00:00, 9.35kB/s]
```

5) Apply LoRA adapters (auto-detect common projection names)

[6] ✓ 0s

```
preferred_targets = ["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj","wi","wo"]
mods = [n for n, m in model.named_modules()]
found = []
for t in preferred_targets:
    if any(t in n for n in mods):
        found.append(t)
if not found:
    found = ["q_proj","v_proj","o_proj"]
print("LoRA target modules:", sorted(set(found)))

lora_cfg = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    task_type=TaskType.CAUSAL_LM,
    target_modules=list(sorted(set(found)))
)
model = get_peft_model(model, lora_cfg)
model.print_trainable_parameters()
```

```
LoRA target modules: ['down_proj', 'gate_proj', 'k_proj', 'o_proj', 'q_proj', 'up_proj', 'v_proj']
trainable params: 2,442,240 || all params: 136,957,248 || trainable%: 1.7832
```

6) Tokenize dataset

[7] ✓ 0s

```
MAX_LEN = 256
def tokenize(batch):
    out = tokenizer(batch["text"], truncation=True, max_length=MAX_LEN)
    out["labels"] = out["input_ids"].copy()
    return out

tokenized = dataset.map(tokenize, batched=True, remove_columns=["text"])
tokenized = tokenized.train_test_split(test_size=0.1, seed=42)
collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
tokenized
```

```
Map: 100% 60/60 [00:00<00:00, 1201.14 examples/s]
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 54
    })
    test: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 6
    })
})
```

7) Training setup (LoRA only trains adapters)

[8] ✓ 0s

```
from dataclasses import fields
BATCH = 16

base_kwargs = dict(
    output_dir="smollm2-lora",
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=BATCH,
    learning_rate=1e-4,
    num_train_epochs=2,
    logging_steps=10,
    save_steps=200,
    save_total_limit=1,
    bf16=torch.cuda.is_available(),
    fp16=False,
    report_to="none",
)

has_eval_strategy = "evaluation_strategy" in {f.name for f in fields(TrainingArguments)}
if has_eval_strategy:
    args = TrainingArguments(evaluation_strategy="steps", eval_steps=50, **base_kwargs)
else:
    print("Note: this transformers build lacks 'evaluation_strategy'; proceeding without step-wise eval.")
    args = TrainingArguments(**base_kwargs)
```

```
Note: this transformers build lacks 'evaluation_strategy'; proceeding without step-wise eval.
```

8) Train

[9]
✓ 28s

```
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=tokenized["train"],
    eval_dataset=tokenized["test"],
    data_collator=collator,
)
trainer.train()
```

```
The model is already on multiple devices. Skipping the move to device specified in `args`.
```


[8/8 00:23, Epoch 2/2]

Step	Training Loss

```
TrainOutput(global_step=8, training_loss=3.1859419345855713, metrics={'train_runtime': 27.8491,
'epoch': 2.0})
```

9) Try the adapted model

[10]
✓ 10s

```
def generate(prompt):
    inputs = tokenizer(f"### Instruction:\n{prompt}\n\n### Response:\n", return_tensors="pt").to(model.device)
    with torch.no_grad():
        out = model.generate(**inputs, max_new_tokens=120, do_sample=True, temperature=0.8, top_p=0.9)
    print(tokenizer.decode(out[0], skip_special_tokens=True).split("### Response:\n")[-1].strip())

generate("List three ways to practice coding every day.")
```

```
There are three ways to practice coding every day.
1. **Automate tasks**: Automate tasks, such as setting reminders, updating your email, or scheduling calendar appointments, to help you stay organized and focused.
2. **Practice regularly**: Incorporate coding practice into your routine at the same time each day, such as before bed, in the morning, or during lunch. This helps keep your eyes on the task at hand and can increase productivity.
3. **Use coding tools**: Utilize coding tools, such as Google Docs, Slack, or Microsoft Office, to help you
```

10) Save the LoRA adapter weights

[11]
✓ 0s

```
adapter_dir = "smollm2-lora-adapter"
model.save_pretrained(adapter_dir)
tokenizer.save_pretrained(adapter_dir)
print("Saved LoRA adapter to:", adapter_dir)
```

```
Saved LoRA adapter to: smollm2-lora-adapter
```