

## Use of AI

**Prompt:** I want to fine-tune a base LLM (Llama 3 from hugging face hub) to translate from Marathi to English, based on this dataset: <https://huggingface.co/datasets/anujsahani01/English-Marathi>. We want to load it in from hugging face, fine-tune using LoRA, and evaluate inferenced samples. We only want to use 100K samples from the dataset. Give starter code in Python, we'll be working in Jupyter notebook.

**Note:** We generated the code below with the prompt in order to start our project. We made modifications throughout when altering the fine-tuning arguments/hyperparameters, debugging, adding new evaluation metrics, testing inference ourselves, etc.

### Response:

Below is a **single, clean Jupyter Notebook**, fully combining:

- Downloading + sampling the 3M-entry dataset ( $\rightarrow$  100k examples)
- Train/val/test split
- LoRA fine-tuning on Llama-3
- Evaluation with BLEU
- Inference cell for interactive translation

You can **copy/paste this directly into a .ipynb** file or into a new Jupyter notebook cell-by-cell.

---



## Complete Jupyter Notebook (all cells)

---



### Cell 1 — Install dependencies

```
!pip install torch transformers datasets accelerate peft sentencepiece sacrebleu -q
```

---



### Cell 2 — Imports

```
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    TrainingArguments,
    DataCollatorForSeq2Seq,
    Trainer
)
```

```
from peft import LoraConfig, get_peft_model, PeftModel
import random
import sacrebleu
import torch
```

---

## 📌 Cell 3 — Download 3M dataset + random sample 100k

```
SAMPLE_SIZE = 100_000
TRAIN_SPLIT = 0.9
VAL_SPLIT = 0.05 # remaining 0.05 is test

print("Loading full dataset...")
ds = load_dataset("anujsahani01/English-Marathi", split="train")

print("Random sampling 100k...")
sampled = ds.shuffle(seed=42).select(range(SAMPLE_SIZE))

# Split indices
train_end = int(TRAIN_SPLIT * SAMPLE_SIZE)
val_end = int((TRAIN_SPLIT + VAL_SPLIT) * SAMPLE_SIZE)

train_ds = sampled.select(range(train_end))
val_ds = sampled.select(range(train_end, val_end))
test_ds = sampled.select(range(val_end, SAMPLE_SIZE))

train_ds, val_ds, test_ds
```

---

## 📌 Cell 4 — Format dataset for supervised fine-tuning

```
def format_example(ex):
    return {
        "prompt": f"Translate from Marathi to English:\nMarathi: {ex['marathi']}\nEnglish:",
        "label": ex["english"]
    }

train_ds = train_ds.map(format_example)
val_ds = val_ds.map(format_example)
test_ds = test_ds.map(format_example)

train_ds[0]
```

---

## 📌 Cell 5 — Load tokenizer + Llama-3 model + apply LoRA

```
MODEL = "meta-llama/Llama-3-8B-Instruct"

tokenizer = AutoTokenizer.from_pretrained(MODEL)
tokenizer.pad_token = tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(
    MODEL,
    device_map="auto"
)

# LoRA config
lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    task_type="CAUSAL_LM"
)

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

---

## 📌 Cell 6 — Tokenization

```
def tokenize_batch(batch):
    return tokenizer(
        batch["prompt"],
        text_target=batch["label"],
        max_length=256,
        truncation=True
    )

train_tok = train_ds.map(tokenize_batch, batched=True)
val_tok = val_ds.map(tokenize_batch, batched=True)

data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

---

## Cell 7 — Training setup

```
training_args = TrainingArguments(  
    output_dir="lora-marathi",  
    per_device_train_batch_size=2,  
    per_device_eval_batch_size=2,  
    gradient_accumulation_steps=8,  
    logging_steps=50,  
    eval_steps=500,  
    save_steps=500,  
    num_train_epochs=2,  
    learning_rate=2e-4,  
    bf16=True,  
    report_to="none"  
)  
  
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=train_tok,  
    eval_dataset=val_tok,  
    data_collator=data_collator,  
    tokenizer=tokenizer  
)
```

---

## Cell 8 — Start training

```
trainer.train()  
  
# Save LoRA adapters  
model.save_pretrained("lora-marathi")
```

---

## Cell 9 — Helper function for generation

```
def generate_translation(model, text):  
    prompt = f"Translate from Marathi to English:\nMarathi: {text}\nEnglish:  
    inputs = tokenizer(prompt, return_tensors="pt").to("cuda")  
    output = model.generate(**inputs, max_new_tokens=80)  
    return tokenizer.decode(output[0], skip_special_tokens=True).split("English:")[-1].strip()
```

---

## Cell 10 — Load base model + fine-tuned LoRA for evaluation

```
print("Loading base model...")
base_model = AutoModelForCausalLM.from_pretrained(MODEL, device_map="auto")

print("Loading fine-tuned LoRA model...")
lora_model = PeftModel.from_pretrained(
    AutoModelForCausalLM.from_pretrained(MODEL, device_map="auto"),
    "lora-marathi"
)
```

---

## Cell 11 — Compute BLEU scores on 500 test examples

```
refs = []
base_preds = []
lora_preds = []

subset = test_ds.select(range(500)) # 500 test samples

for ex in subset:
    mar = ex["marathi"]
    eng = ex["english"]
    refs.append(eng)

    base_preds.append(generate_translation(base_model, mar))
    lora_preds.append(generate_translation(lora_model, mar))

base_bleu = sacrebleu.corpus_bleu(base_preds, [refs]).score
lora_bleu = sacrebleu.corpus_bleu(lora_preds, [refs]).score

print("BLEU Scores:")
print(f"Base model: {base_bleu:.2f}")
print(f"LoRA fine-tuned: {lora_bleu:.2f}")
```

---

## Cell 12 — Interactive Inference

```
def translate(text):
    return generate_translation(lora_model, text)
```

```
while True:
    mar = input("Marathi → ")
    print("English:", translate(mar))
```

---



## Your Notebook Is Ready

If you'd like, I can also:

- Add **charts** (loss curves, BLEU over epochs)
- Add a **Gradio UI** translator
- Convert this into a **Google Colab notebook**
- Build a **HuggingFace model card** for your LoRA adapter

Just tell me what you'd like next!