**Complete Guide to Hybrid Fine-Tuning LLMs with LoRA + Prefix + Adapter on Bangla Data**

**1. Goal of This Tutorial**

* Fine-tune a large language model (LLM) using **Bangla instructions** the model hasn’t seen before.
* Demonstrate **actual learning** by evaluating responses in Bangla.
* Use **memory-efficient techniques**:
  + LoRA (Low-Rank Adaptation)
  + Prefix Tuning
  + Adapter / Modular Fine-Tuning
* Save the trained model for **future inference**.

**2. Why This Setup Works**

* **LoRA** → Updates only a small subset of parameters; low memory.
* **Prefix Tuning** → Adds learnable embeddings at the start of every input, guiding the model.
* **Adapters** → Small extra modules inserted inside transformer layers; modular and reusable.
* **Hybrid approach** → All three together:
  + Base model stays frozen → avoids catastrophic forgetting
  + Small modules learn new knowledge → efficient and modular
  + Works for low-memory setups (4-bit or 8-bit quantization)

**3. Dataset Preparation (Bangla Novel Instructions)**

We’ll create a **JSONL dataset** of **50–100 instructions** in Bangla:

import json

import random

import os

OUTPUT\_FILE = "./bangla\_novel\_dataset.jsonl"

NUM\_EXAMPLES = 50 # Increase if needed

instructions = [

"একটি ছোট কল্পকাহিনী লিখো যেখানে একটি ড্রাগন এবং একটি রোবট বন্ধু হয়।",

"স্টেপ বাই স্টেপ কিভাবে একটি অরিগাম ফুল বানাতে হয় তা ব্যাখ্যা করো।",

"একটি নতুন স্পেস এক্সপ্লোরেশন কোম্পানির তিনটি সৃজনশীল নাম দাও।",

"বাংলা ভাষায় একটি ছোট কবিতা লিখো প্রকৃতির উপর।",

"একটি মজার রেসিপি বানাও যা শুধুমাত্র ৫ মিনিটে করা যায়।",

]

responses = [

"একটি সুন্দর বন ছিল যেখানে একটি ড্রাগন এবং রোবট একসাথে খেলে।",

"প্রথমে একটি বর্গাকার কাগজ নাও, তারপর কোণগুলো ভাঁজ করো ...",

"1. তারকা ভ্রমণকারী 2. নভা ফ্রন্টিয়ার 3. কসমিক পাইওনিয়ার্স",

"প্রকৃতির সৌন্দর্য যেন মনকে শান্তি দেয়। পাখির গান, বাতাসের শব্দ ...",

"৫ মিনিটে তৈরি হওয়া এই রেসিপি খুবই সহজ এবং সুস্বাদু।",

]

os.makedirs(os.path.dirname(OUTPUT\_FILE) or ".", exist\_ok=True)

with open(OUTPUT\_FILE, "w", encoding="utf-8") as f:

for i in range(NUM\_EXAMPLES):

instr = random.choice(instructions)

resp = random.choice(responses)

f.write(json.dumps({"instruction": instr, "output": resp}, ensure\_ascii=False) + "\n")

print(f"✅ {NUM\_EXAMPLES} Bangla instructions saved to {OUTPUT\_FILE}")

**Key Points:**

* **JSONL format** → compatible with Hugging Face datasets.
* **Bangla language** ensures novelty; model must learn new knowledge.
* **50–100 examples** is enough for demonstration using PEFT.

**4. Understanding LoRA Parameters**

LoRA (Low-Rank Adaptation) is a **parameter-efficient fine-tuning technique**.

**Key Parameters**

| **Parameter** | **Meaning** |
| --- | --- |
| r | Rank of the low-rank matrices. Small values = fewer trainable params. Typically 4–16. |
| target\_modules | Names of model submodules where LoRA matrices are applied (e.g., q\_proj, v\_proj in attention layers). |
| lora\_alpha | Scaling factor applied to LoRA output. Higher = stronger updates. |
| lora\_dropout | Dropout applied to LoRA updates to reduce overfitting. |
| bias | Bias term options: "none" (no bias trained), "all", or "lora\_only". |
| task\_type | Type of task: "CAUSAL\_LM", "SEQ\_2\_SEQ\_LM", etc. |

**Analogy:** LoRA is like **adding sticky notes** to a textbook:

* Only small parts of the knowledge are adjusted.
* Base model remains untouched.

**5. Prefix Tuning Parameters**

| **Parameter** | **Meaning** |
| --- | --- |
| num\_virtual\_tokens | Length of learnable prefix tokens prepended to input. |
| encoder\_hidden\_size | Should match model hidden size (important for embeddings). |
| task\_type | Task type: "CAUSAL\_LM" or "SEQ\_2\_SEQ\_LM". |

**Analogy:** Prefix tuning is like giving the model **a hint sentence** that guides how it answers.

**6. Adapter / Modular Fine-Tuning**

| **Parameter** | **Meaning** |
| --- | --- |
| reduction\_factor | Shrinks hidden dimension for adapter to reduce trainable parameters. |
| task\_type | "CAUSAL\_LM" etc. |

**Analogy:** Adapter = **tiny extra neurons/modules** added inside transformer blocks for task-specific knowledge.

**7. Full Hybrid Fine-Tuning Code (Bangla Dataset)**

import torch

from transformers import AutoModelForCausalLM, AutoTokenizer, TrainingArguments

from datasets import load\_dataset

from peft import LoraConfig, PrefixTuningConfig, AdapterConfig, get\_peft\_model

from trl import SFTTrainer

import os

# ---------------- SETTINGS ----------------

MODEL\_NAME = "unsloth/Qwen2.5-0.5B-Instruct"

OUTPUT\_DIR = "./bangla\_hybrid\_finetuned"

MAX\_SEQ\_LENGTH = 384

BATCH\_SIZE = 2

NUM\_EPOCHS = 2

LEARNING\_RATE = 1e-4

os.makedirs(OUTPUT\_DIR, exist\_ok=True)

# ---------------- LOAD CUSTOM DATASET ----------------

dataset = load\_dataset("json", data\_files={"train": "./bangla\_novel\_dataset.jsonl"}, split="train")

def format\_bangla(examples):

return {"text": [f"Instruction: {i}\nResponse: {o}" for i,o in zip(examples["instruction"], examples["output"])]}

dataset = dataset.map(format\_bangla, batched=True, remove\_columns=dataset.column\_names)

# ---------------- LOAD MODEL ----------------

model = AutoModelForCausalLM.from\_pretrained(

MODEL\_NAME,

load\_in\_4bit=True,

torch\_dtype=torch.float16 if torch.cuda.is\_available() else torch.float32,

device\_map="auto"

)

tokenizer = AutoTokenizer.from\_pretrained(MODEL\_NAME)

if tokenizer.pad\_token is None:

tokenizer.pad\_token = tokenizer.eos\_token

# ---------------- PEFT CONFIG ----------------

lora\_config = LoraConfig(

r=4,

target\_modules=["q\_proj","v\_proj"],

lora\_alpha=4,

lora\_dropout=0.1,

bias="none",

task\_type="CAUSAL\_LM"

)

prefix\_config = PrefixTuningConfig(

task\_type="CAUSAL\_LM",

num\_virtual\_tokens=20,

encoder\_hidden\_size=model.config.hidden\_size

)

adapter\_config = AdapterConfig(

task\_type="CAUSAL\_LM",

reduction\_factor=16

)

# ---------------- APPLY PEFT ----------------

model = get\_peft\_model(model, lora\_config)

model = get\_peft\_model(model, prefix\_config)

model = get\_peft\_model(model, adapter\_config)

# ---------------- TRAINING ----------------

training\_args = TrainingArguments(

output\_dir=OUTPUT\_DIR,

per\_device\_train\_batch\_size=BATCH\_SIZE,

gradient\_accumulation\_steps=8,

num\_train\_epochs=NUM\_EPOCHS,

learning\_rate=LEARNING\_RATE,

save\_strategy="no",

logging\_steps=5,

fp16=True,

optim="adamw\_torch",

)

trainer = SFTTrainer(

model=model,

train\_dataset=dataset,

dataset\_text\_field="text",

max\_seq\_length=MAX\_SEQ\_LENGTH,

tokenizer=tokenizer,

args=training\_args

)

model.train()

trainer.train()

# ---------------- SAVE ----------------

model.save\_pretrained(OUTPUT\_DIR)

tokenizer.save\_pretrained(OUTPUT\_DIR)

print("✅ Fine-tuning complete! Model saved.")

**8. Testing Fine-Tuned Model**

test\_instruction = "একটি ছোট কল্পকাহিনী লিখো যেখানে একটি ড্রাগন এবং একটি রোবট বন্ধু হয়।"

inputs = tokenizer(f"Instruction: {test\_instruction}\nResponse:", return\_tensors="pt").to(model.device)

outputs = model.generate(\*\*inputs, max\_new\_tokens=100)

print(tokenizer.decode(outputs[0], skip\_special\_tokens=True))

* The model should now generate **Bangla responses influenced by your dataset**, proving **actual fine-tuning**.

**9. Memory & Practical Tips**

1. **Batch Size & Accumulation:** Small batch size (2–4) + gradient accumulation.
2. **4-bit Quantization:** Reduces memory, allows training large models on consumer GPUs.
3. **LoRA Rank r:** 4–8 is good for small datasets; higher ranks improve flexibility but use more memory.
4. **Number of Virtual Tokens (Prefix):** 20–50 is enough for short instruction tasks.
5. **Adapter Reduction Factor:** 16–32 reduces number of trainable parameters significantly.
6. **Epochs:** Small dataset → 1–3 epochs enough to demonstrate learning.

**✅ 10. Key Takeaways**

* **Hybrid PEFT (LoRA + Prefix + Adapter)** = memory-efficient, modular, multi-task capable.
* **Bangla custom dataset** ensures model learns genuinely new information.
* **Saved model** can be loaded for inference later without retraining.
* **LoRA / Prefix / Adapter parameters** are essential for tuning efficiency & memory.
* This workflow is **practical for real projects** and can be used in interviews to demonstrate your **LLM fine-tuning expertise**.