

```

import os
import torch
from datasets import load_dataset
from peft import LoraConfig, prepare_model_for_kbit_training,
get_peft_model
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    BitsAndBytesConfig,
    TrainingArguments
)
from trl import DPOTrainer

# --- Configuration ---
MODEL_ID = "mistralai/Mistral-7B-v0.1"
OUTPUT_DIR = "./neuro-doc-adapter"

def train():
    # 1. Load Model in 4-bit to save GPU Memory (QLoRA)
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16
    )

    model = AutoModelForCausalLM.from_pretrained(
        MODEL_ID,
        quantization_config=bnb_config,
        device_map="auto"
    )

    # 2. Configure LoRA (Low-Rank Adaptation)
    peft_config = LoraConfig(
        r=16,
        lora_alpha=16,
        lora_dropout=0.05,
        bias="none",
        task_type="CAUSAL_LM",
        target_modules=["q_proj", "v_proj"]
    )
    model = prepare_model_for_kbit_training(model)
    model = get_peft_model(model, peft_config)

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

    # 3. Load Preference Dataset (Chosen vs Rejected)
    # Ensure your dataset has 'prompt', 'chosen', and 'rejected'

```

```
columns
    dataset = load_dataset("json",
data_files="data/preference_data.json", split="train")

# 4. Initialize DPO Trainer
training_args = TrainingArguments(
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    max_steps=500,
    learning_rate=2e-4,
    fp16=True,
    logging_steps=10,
    output_dir=OUTPUT_DIR,
    optim="paged_adamw_32bit"
)

trainer = DPOTrainer(
    model=model,
    ref_model=None, # LoRA acts as implicit ref
    args=training_args,
    train_dataset=dataset,
    tokenizer=tokenizer,
    peft_config=peft_config,
    beta=0.1,
    max_length=1024,
    max_prompt_length=512,
)

# 5. Train & Save
print("Starting DPO Alignment...")
trainer.train()
trainer.model.save_pretrained(OUTPUT_DIR)
print(f"Model saved to {OUTPUT_DIR}")

if __name__ == "__main__":
    train()
```