Projet Mars 2025

[ ]:
```
!pip install transformers torch

from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import torch

model_name = "gpt2-medium"

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

model = AutoModelForCausalLM.from_pretrained(model_name)
model.eval()
model.to("cuda" if torch.cuda.is_available() else "cpu")

generator = pipeline("text-generation", model=model, tokenizer=tokenizer,␣
 ↪device=0 if torch.cuda.is_available() else -1)

examples = [
    "What is the capital of France?",
    "What are the three primary colors?",
    "Traduit : What happened ?",
    "Give three tips for staying healthy."
]

for instruction in examples:
    prompt = f"Instruction: {instruction}\nRéponse:"
    output = generator(prompt, max_new_tokens=60)[0]["generated_text"]
    print("\n", instruction)
    print(output)
```

[ ]:
```
!pip install datasets

from datasets import load_dataset

# Charger Alpaca, en gardant seulement 'instruction' et 'output'
dataset = load_dataset("tatsu-lab/alpaca")["train"]
dataset = dataset.remove_columns(["input"])
```

```
# Optionnel : filtrer les réponses longues (> 80 mots)
def is_simple(example):
    return len(example["output"].split()) <= 80

dataset = dataset.filter(is_simple)
```

[ ]:
```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("gpt2-medium")
tokenizer.pad_token = tokenizer.eos_token  # GPT-2 n'a pas de token de padding␣
↪à la base

def tokenize(example):
    prompt = f"Instruction: {example['instruction']}\nRéponse:"
    full_text = prompt + " " + example["output"]
    tokenized = tokenizer(full_text, truncation=True, max_length=128,␣
↪padding="max_length")
    tokenized["labels"] = tokenized["input_ids"].copy()
    return tokenized

tokenized_dataset = dataset.map(tokenize, remove_columns=dataset.column_names)

from transformers import AutoModelForCausalLM

base_model = AutoModelForCausalLM.from_pretrained("gpt2-medium")
base_model.resize_token_embeddings(len(tokenizer))  # Pour intégrer le token de␣
↪padding
```

[ ]:
```
from peft import get_peft_model, LoraConfig, TaskType

lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=["c_attn"],
    lora_dropout=0.1,
    bias="none",
    task_type=TaskType.CAUSAL_LM,
)

model = get_peft_model(base_model, lora_config)
```

[ ]:
```
from transformers import TrainingArguments, Trainer,␣
↪DataCollatorForLanguageModeling

output_dir = "./gpt2-medium-alpaca-lora"
```

```
training_args = TrainingArguments(
    output_dir=output_dir,
    per_device_train_batch_size=8,
    gradient_accumulation_steps=2,
    logging_steps=1000,
    learning_rate=3e-4,
    num_train_epochs=2,
    fp16=torch.cuda.is_available(),
    save_strategy="epoch",
    save_total_limit=1,
    report_to=[],
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    data_collator=DataCollatorForLanguageModeling(tokenizer=tokenizer,␣
 ↪mlm=False),
)

trainer.train()

model.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)
```

[ ]:
```
from transformers import pipeline

pipe = pipeline("text-generation", model=model, tokenizer=tokenizer, device=0␣
 ↪if torch.cuda.is_available() else -1)

instruction = "What is the capital of France?"
prompt = f"Instruction: {instruction}\nRéponse:"
result = pipe(
    prompt,
    max_new_tokens=80,
    temperature=0.7,       # plus conservateur
    top_p=0.9,             # nucleus sampling
    top_k=50,              # coupe les options absurdes
    eos_token_id=tokenizer.eos_token_id
)[0]["generated_text"]

print("\n Réponse générée :\n")
print(result)
```

[ ]:
```python
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM
from peft import PeftModel

# Instructions de test
eval_instructions = [
    "What is the capital of France?",
    "Give three tips for staying healthy.",
    "Create a list of five different animals",
    "Who wrote 'Romeo and Juliet'?"
]

# Paramètres de génération (réglés pour du bon contrôle)
gen_kwargs = dict(
    max_new_tokens=80,
    temperature=0.7,
    top_p=0.9,
    do_sample=True,
    eos_token_id=50256  # Fin de séquence pour GPT2
)

# Charger modèle de base
model_base = AutoModelForCausalLM.from_pretrained("gpt2-medium")
tokenizer_base = AutoTokenizer.from_pretrained("gpt2-medium")

# Charger modèle fine-tuné
model_ft = AutoModelForCausalLM.from_pretrained("gpt2-medium")
tokenizer_ft = AutoTokenizer.from_pretrained("gpt2-medium")
model_ft.resize_token_embeddings(len(tokenizer_ft))
model_ft = PeftModel.from_pretrained(model_ft, "./gpt2-medium-alpaca-lora")

# Pipelines
pipe_base = pipeline("text-generation", model=model_base,␣
 ↪tokenizer=tokenizer_base, device_map="auto")
pipe_ft = pipeline("text-generation", model=model_ft, tokenizer=tokenizer_ft,␣
 ↪device_map="auto")

# Comparaison
for instr in eval_instructions:
    prompt = f"Instruction: {instr}\nRéponse:"

    base_resp = pipe_base(prompt, **gen_kwargs)[0]["generated_text"]
    ft_resp = pipe_ft(prompt, **gen_kwargs)[0]["generated_text"]

    # Affichage propre
    print("\n" + "="*80)
    print(f" Instruction: {instr}")
    print("\n Réponse GPT2 (base):")
```

```
    print(base_resp.replace(prompt, "").strip())

    print("\n Réponse GPT2 fine-tuné:")
    print(ft_resp.replace(prompt, "").strip())
    print("="*80)
```

[ ]:
```
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM
from peft import PeftModel

# Charger modèle fine-tuné
base_model = AutoModelForCausalLM.from_pretrained("gpt2-medium")
tokenizer = AutoTokenizer.from_pretrained("gpt2-medium")
base_model.resize_token_embeddings(len(tokenizer))

# Charger les poids LoRA
model = PeftModel.from_pretrained(base_model, "./gpt2-medium-alpaca-lora")

# Pipeline avec réglages pour qualité
pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    device_map="auto"
)

# Paramètres de génération
gen_kwargs = dict(
    max_new_tokens=100,
    temperature=0.7,       # Gère la créativité (0.7 = bon équilibre)
    top_p=0.9,             # Nucleus sampling
    do_sample=True,
    eos_token_id=50256     # Pour forcer l'arrêt en fin de phrase
)

# Interface utilisateur
print("Assistant Fine-tuné Alpaca | Tape 'exit' pour quitter\n")

while True:
    instr = input(" Instruction: ")
    if instr.lower() in ["exit", "quit"]:
        break

    prompt = f"Instruction: {instr}\nRéponse:"

    try:
        result = pipe(prompt, **gen_kwargs)[0]["generated_text"]
        print("\n Réponse générée :")
```

```
        print(result.replace(prompt, "").strip())
        print("=" * 80 + "\n")
    except Exception as e:
        print(" Erreur :", e)
```