Projet LLM : de la récupération de texte à la création d'un modèle génératif (Python)

Objectif du projet

Créer un **mini modèle génératif spécialisé sur un thème**, en partant de textes bruts récupérés sur Internet ou depuis des fichiers.

Exemples de thèmes :

- cuisine (recettes, conseils)
- support informatique (FAQ, messages d'aide)
- sport (résumés de matchs, commentaires)
- finance perso, jeux vidéo, etc.

On ne va **pas** entraîner un modèle de zéro (trop coûteux).
On va **fine-tuner** un petit modèle existant (`distilgpt2`) sur notre propre corpus.

---

1. Préparer l'environnement Python

1.1. Créer un environnement (optionnel mais conseillé)

```
python -m venv venv
source venv/bin/activate  # sous Windows: venv\Scripts\activate
```

1.2. Installer les librairies nécessaires

```
pip install transformers datasets accelerate sentencepiece
pip install torch  # ou suivez la commande recommandée sur le site de PyTorch
```

---

2. Choisir le thème et la source de texte

2.1. Définir le thème

Exemples :

- « Chatbot de conseils en cuisine »
- « Assistant pour le support IT »
- « Générateur d'explications de code Python simples »

> **À faire (en groupe) :**
>
> - Choisir un thème

> - Écrire en 3–4 phrases ce que le chatbot doit être capable de faire.

2.2. Choisir la source de texte

Options possibles :

- Copier-coller des **articles** (blogs, wikis, documentation)
- Récupérer des **FAQ** (support technique, forums)
- Utiliser des **fichiers texte** (`.txt`, `.md`) que vous avez déjà

> ⚠ Attention au droit d'auteur : privilégier les contenus libres ou purement pédagogiques.

---

3. Récupérer les textes (scraping simple ou fichiers)

3.1. Exemple 1 : charger des fichiers texte locaux

Organisation des fichiers :

```
data/
  recette1.txt
  recette2.txt
  recette3.txt
```

Code pour charger tous les fichiers :

```
import os

DATA_DIR = "data"

texts = []
for filename in os.listdir(DATA_DIR):
    if filename.endswith(".txt"):
        with open(os.path.join(DATA_DIR, filename), "r", encoding="utf-8") as f:
            texts.append(f.read())

len(texts), texts[0][:500]
```

3.2. Exemple 2 : récupérer quelques pages web (simple)

```
import requests
from bs4 import BeautifulSoup

urls = [
    "https://fr.wikipedia.org/wiki/Apprentissage_automatique",
    # ajouter d'autres URLs adaptées à votre thème
]
```

```
texts = []

for url in urls:
    html = requests.get(url).text
    soup = BeautifulSoup(html, "html.parser")
    # on récupère le texte principal des paragraphes
    page_text = "\n".join(p.get_text() for p in soup.find_all("p"))
    texts.append(page_text)

len(texts), texts[0][:500]
```

> 💡 Vous pouvez combiner plusieurs sources (fichiers + web).

---

4. Nettoyage et préparation du texte

On veut obtenir **un seul gros texte** ou **une liste de textes propres**.

```
import re

def clean_text(text: str) -> str:
    # retirer les multiples espaces
    text = re.sub(r"\s+", " ", text)
    # retirer certains caractères bizarres si besoin
    text = text.replace("\xa0", " ")
    return text.strip()

cleaned_texts = [clean_text(t) for t in texts]

# option : concaténer tous les textes en un seul
full_corpus = "\n\n".join(cleaned_texts)
len(full_corpus), full_corpus[:1000]
```

> **But :** enlever le bruit, garder un texte lisible.

---

5. Créer un dataset pour le fine-tuning

On va transformer le texte en un objet `Dataset` de Hugging Face.

5.1. Version « un seul long texte »

```
from datasets import Dataset

dataset = Dataset.from_dict({"text": [full_corpus]})
dataset
```

5.2. Version « plusieurs textes séparés »

```
from datasets import Dataset

dataset = Dataset.from_dict({"text": cleaned_texts})
dataset
```

---

6. Choisir un modèle de base : `distilgpt2`

On utilise un petit modèle GPT, déjà entraîné en anglais. Si vous avez beaucoup de texte français, ça fonctionne quand même comme démonstration.

```
from transformers import AutoTokenizer, AutoModelForCausalLM

model_name = "distilgpt2"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
```

> ⚠ GPT2 n'a pas de token spécial de padding par défaut. On va le définir :

```
tokenizer.pad_token = tokenizer.eos_token
model.config.pad_token_id = tokenizer.eos_token_id
```

---

7. Tokenisation du dataset

On convertit le texte en **tokens** pour que le modèle puisse l'utiliser.

```
block_size = 128  # taille des séquences (en tokens)

def tokenize_function(examples):
    return tokenizer(
        examples["text"],
        truncation=True,
        max_length=block_size,
        padding="max_length",
    )

tokenized_dataset = dataset.map(tokenize_function, batched=True, remove_columns=
["text"])
tokenized_dataset[0]
```

7.1. Split train / validation

```
tokenized_dataset = tokenized_dataset.train_test_split(test_size=0.1)
train_dataset = tokenized_dataset["train"]
eval_dataset = tokenized_dataset["test"]
len(train_dataset), len(eval_dataset)
```

---

8. Préparer le fine-tuning avec `Trainer`

On va utiliser la classe `Trainer` de 🤗 Transformers.

```
from transformers import Trainer, TrainingArguments

output_dir = "./mini-gpt-finetuned"

training_args = TrainingArguments(
    output_dir=output_dir,
    evaluation_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_steps=10,
    save_strategy="epoch",
)
```

8.1. Définir le Trainer

```
def data_collator(features):
    # simple collator pour LM causal
    import torch
    batch = {}
    batch["input_ids"] = torch.stack([torch.tensor(f["input_ids"]) for f in
features])
    batch["attention_mask"] = torch.stack([torch.tensor(f["attention_mask"]) for f
in features])
    # pour un modèle GPT, on prédit le prochain token = même séquence décalée
    batch["labels"] = batch["input_ids"].clone()
    return batch

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    data_collator=data_collator,
)
```

8.2. Lancer l'entraînement

```
trainer.train()
```

> Selon la machine, ça peut prendre quelques minutes. Sur un GPU (Google Colab) c'est plus rapide.

---

9. Sauvegarder le modèle fine-tuné

```
trainer.save_model(output_dir)
tokenizer.save_pretrained(output_dir)
```

Vous avez maintenant un **mini modèle GPT spécialisé** sur vos textes.

---

10. Générer du texte avec le modèle

On recharge le modèle (si besoin) et on génère.

```
from transformers import pipeline

generator = pipeline(
    "text-generation",
    model=output_dir,
    tokenizer=output_dir,
)

prompt = "Voici une astuce importante en cuisine :"
outputs = generator(
    prompt,
    max_length=100,
    num_return_sequences=3,
    do_sample=True,
    top_k=50,
    top_p=0.95,
)

for i, out in enumerate(outputs, 1):
    print(f"--- Génération {i} ---")
    print(out["generated_text"])
    print()
```

> **À faire :**
>
> - Tester plusieurs prompts adaptés à votre thème
> - Comparer : avant / après fine-tuning (avec le modèle de base)

---

11. Évaluation simple du modèle génératif

Idées d'évaluation qualitatives (sans métriques avancées) :

1. **Cohérence** : le texte est-il logique, grammatical ?
2. **Style** : ressemble-t-il aux textes d'entraînement ?
3. **Spécialisation** : le modèle reste-t-il dans le bon domaine (cuisine, IT, etc.) ?
4. **Hallucinations** : invente-t-il des choses fausses / dangereuses ?

> Exercice :
>
> - En groupe, générer 5 réponses du modèle
> - Les annoter : ☑ correct / ❌ incorrect
> - Discuter ce qui pourrait améliorer le modèle : plus de données, nettoyage, prompts, etc.

---

12. Extensions possibles (pour aller plus loin)

12.1. Passer à un dataset « instruction » (format Alpaca)

Au lieu de juste continuer du texte, vous pouvez créer un dataset :

```
{"instruction": "Donne une astuce de cuisine pour les pâtes", "input": "",
"output": "Pour éviter qu'elles collent..."}
```

Puis construire un prompt de type :

```
def format_example(example):
    return f"Instruction: {example['instruction']}\nRéponse:
{example['output']}\n\n"
```

Et entraîner le modèle sur ces textes formatés.

12.2. Utiliser LoRA (PEFT) pour un modèle plus gros

Avec la librairie `peft`, vous pouvez adapter un modèle plus grand (ex: `gpt2`) sans modifier tous les paramètres.

---

13. Récapitulatif du pipeline

1. Choisir un **thème** et définir ce que le chatbot doit faire
2. Récupérer des **textes bruts** (web, fichiers, FAQ…)
3. Nettoyer et fusionner les textes
4. Créer un **dataset Hugging Face**
5. Charger un **modèle de base** (distilgpt2) + tokenizer
6. Tokeniser le dataset, fixer la taille des séquences

7. Split train / validation
8. Fine-tuner le modèle avec `Trainer`
9. Sauvegarder le modèle fine-tuné
10. Générer du texte et évaluer qualitativement
11. (Optionnel) Passer à un dataset d'instructions ou à LoRA

Vous avez alors un **mini modèle génératif spécialisé**, construit de bout en bout en Python.