Guide Technique Détaillé : Créer une IA Locale de A à Z

Manuel Complet avec Commandes et Code Prêt à l’Emploi

Documentation Technique IA Locale

Janvier 2025

Table of Contents

# Glossaire des Termes et Acronymes

## A

**AI / IA (Intelligence Artificielle)** : Capacité d’un système informatique à effectuer des tâches qui nécessitent normalement l’intelligence humaine (raisonnement, apprentissage, compréhension du langage).

**API (Application Programming Interface)** : Interface permettant à des applications de communiquer entre elles via des protocoles standardisés.

**ASR (Automatic Speech Recognition)** : Reconnaissance automatique de la parole, conversion audio → texte.

## B

**Batch Size** : Nombre d’exemples traités simultanément lors de l’entraînement d’un modèle.

**bitsandbytes** : Bibliothèque Python pour la quantification de modèles en 4 et 8 bits, réduisant drastiquement l’utilisation mémoire.

## C

**Chunking** : Processus de segmentation de documents longs en morceaux (chunks) plus petits pour faciliter le traitement par l’IA.

**CUDA (Compute Unified Device Architecture)** : Plateforme de calcul parallèle de NVIDIA pour exploiter la puissance des GPU.

**Cross-Encoder** : Type de modèle qui traite simultanément requête + document pour calculer un score de pertinence (plus précis mais plus lent que bi-encoder).

## D

**DVC (Data Version Control)** : Système de gestion de versions pour les données et modèles de machine learning, équivalent de Git pour les données.

## E

**Embeddings** : Représentations vectorielles denses (tableaux de nombres réels) de textes ou autres données, capturant leur sens sémantique dans un espace multidimensionnel.

**Epoch** : Une passe complète sur l’ensemble des données d’entraînement.

## F

**FAISS (Facebook AI Similarity Search)** : Bibliothèque ultra-efficace pour la recherche de similarité et le clustering de vecteurs denses, optimisée CPU et GPU.

**Fine-tuning** : Processus d’ajustement d’un modèle pré-entraîné sur des données spécifiques à une tâche ou un domaine particulier.

## G

**GGUF (GPT-Generated Unified Format)** : Format de fichier pour stocker des modèles de langage quantifiés, utilisé par llama.cpp et Ollama.

**GPU (Graphics Processing Unit)** : Processeur graphique utilisé pour accélérer massivement les calculs parallèles en machine learning.

**Gradient Accumulation** : Technique pour simuler un grand batch size en accumulant les gradients sur plusieurs mini-batches.

## H

**HNSW (Hierarchical Navigable Small World)** : Algorithme de graphe pour la recherche approximative de plus proches voisins, très performant.

**Hugging Face** : Plateforme et écosystème open-source pour le machine learning (modèles, datasets, bibliothèques).

## K

**kNN (k-Nearest Neighbors)** : Algorithme de recherche des k voisins les plus proches dans un espace vectoriel.

**KV-Cache (Key-Value Cache)** : Mécanisme d’optimisation pour stocker les clés et valeurs calculées dans les transformers, accélérant la génération.

## L

**LLM (Large Language Model)** : Modèle de langage de grande taille pré-entraîné sur de vastes corpus de texte (billions de mots), capable de comprendre et générer du texte.

**LoRA (Low-Rank Adaptation)** : Technique d’adaptation de modèles par ajout de matrices de rang faible, permettant un fine-tuning efficace avec peu de paramètres.

**Learning Rate** : Taux d’apprentissage, contrôle la vitesse de mise à jour des paramètres lors de l’entraînement.

## M

**MLOps (Machine Learning Operations)** : Ensemble de pratiques pour déployer et maintenir des modèles de machine learning en production.

**MTEB (Massive Text Embedding Benchmark)** : Benchmark de référence pour évaluer la qualité des embeddings de texte sur 58 datasets.

## N

**NLP (Natural Language Processing)** : Traitement automatique du langage naturel, branche de l’IA dédiée à la compréhension et génération de texte.

## P

**PEFT (Parameter-Efficient Fine-Tuning)** : Famille de techniques de fine-tuning qui modifient seulement une petite partie des paramètres (LoRA, Adapters, Prefix Tuning).

**PII (Personally Identifiable Information)** : Informations permettant d’identifier une personne (nom, email, téléphone, adresse, numéro de sécurité sociale, etc.).

**POC (Proof of Concept)** : Preuve de concept, prototype démontrant la faisabilité technique et la valeur d’une solution.

**Prompt** : Instruction ou question fournie à un modèle de langage pour guider sa génération de texte.

## Q

**QLoRA (Quantized Low-Rank Adaptation)** : Combinaison de quantification 4-bit et LoRA pour réduire drastiquement l’utilisation mémoire lors du fine-tuning.

**Quantification** : Processus de réduction de la précision des poids d’un modèle (ex: de 32 bits à 8, 4 ou 2 bits) pour économiser mémoire et calculs.

## R

**RAG (Retrieval-Augmented Generation)** : Approche combinant recherche documentaire (retrieval) et génération de texte, permettant au LLM de s’appuyer sur une base de connaissances externe.

**RAM (Random Access Memory)** : Mémoire vive de l’ordinateur, utilisée pour stocker les données en cours de traitement.

**Reranking** : Processus de reclassement de résultats de recherche pour améliorer leur pertinence, utilisant généralement un cross-encoder.

**RGPD (Règlement Général sur la Protection des Données)** : Réglementation européenne sur la protection des données personnelles, en vigueur depuis 2018.

**ROCm (Radeon Open Compute)** : Plateforme de calcul GPU open-source d’AMD, alternative à CUDA.

## T

**Tokenizer** : Outil qui découpe un texte en unités (tokens) pour le traitement par un modèle.

**Transformer** : Architecture de réseau de neurones basée sur le mécanisme d’attention, fondement des LLM modernes (GPT, BERT, Llama).

**TTS (Text-to-Speech)** : Synthèse vocale, conversion de texte en parole.

## V

**ViT (Vision Transformer)** : Architecture transformer adaptée aux tâches de vision par ordinateur.

**VRAM (Video RAM)** : Mémoire dédiée de la carte graphique, cruciale pour l’entraînement et l’inférence de modèles sur GPU.

## Y

**YOLO (You Only Look Once)** : Famille d’algorithmes de détection d’objets en temps réel, très performants pour la vision par ordinateur.

# Table des Matières Détaillée

1. [Introduction et Objectifs](#introduction)
2. [Prérequis Matériels et Logiciels](#prerequis)
3. [Installation et Configuration de l’Environnement](#installation)
4. [Définition du Problème et Choix de l’Approche](#definition-probleme)
5. [Préparation et Nettoyage des Données](#preparation-donnees)
6. [Algorithmes Clés : RAG, LoRA, Embeddings](#algorithmes)
7. [Implémentation Complète d’un Système RAG](#implementation-rag)
8. [Fine-tuning avec QLoRA](#fine-tuning)
9. [Déploiement d’une API Locale](#deploiement-api)
10. [Évaluation et Optimisation](#evaluation)
11. [Sécurité, Anonymisation et Conformité RGPD](#securite)
12. [Scripts Complets Prêts à l’Emploi](#scripts-complets)
13. [Troubleshooting et FAQ](#troubleshooting)
14. [Ressources et Références](#ressources)

# 1. Introduction et Objectifs

## 1.1 Pourquoi une IA Locale ?

Ce guide complet vous accompagne dans la création d’un système d’**Intelligence Artificielle entièrement local**, fonctionnant sur votre propre infrastructure sans dépendre de services cloud externes.

### Avantages d’une IA Locale

**Confidentialité Maximale** - Vos données sensibles ne quittent **jamais** votre infrastructure - Aucune transmission vers des serveurs tiers - Contrôle total sur le traitement des informations

**Maîtrise des Coûts** - Élimination des frais d’abonnement récurrents (souvent $20-100/mois par utilisateur) - Investissement initial amorti sur 2-3 ans - Pas de facturation à l’usage

**Souveraineté Technologique** - Indépendance vis-à-vis des fournisseurs externes - Pas de risque de changement de tarification - Pas de dépendance à une disponibilité de service externe

**Conformité RGPD** - Contrôle total sur le stockage et le traitement des données personnelles - Traçabilité complète - Facilite les audits de conformité

**Personnalisation** - Adaptation complète aux besoins spécifiques - Fine-tuning sur vos propres données - Aucune limitation d’utilisation

## 1.2 Cas d’Usage Couverts

Ce guide permet d’implémenter :

1. **Assistant conversationnel interne** (Q&A sur documentation)
2. **Système de recherche sémantique** (base de connaissances)
3. **Résumeur de documents** (rapports, emails, tickets)
4. **Extracteur d’informations** (analyse de contrats, CVs)
5. **Générateur de contenu** (emails, descriptions produits)
6. **Analyseur de sentiment** (feedbacks clients)

## 1.3 Technologies Utilisées

* **Python 3.10/3.11** : langage principal
* **PyTorch** : framework de deep learning
* **Hugging Face Transformers** : modèles pré-entraînés
* **FAISS / Chroma** : bases de données vectorielles
* **Ollama / llama.cpp** : inférence LLM locale
* **LangChain** : framework RAG
* **FastAPI** : serveur API
* **Pandoc** : conversion de documents

# 2. Prérequis Matériels et Logiciels

## 2.1 Configuration Matérielle

### Configuration Minimale (RAG Simple - CPU uniquement)

Processeur : Intel Core i5/i7 ou AMD Ryzen 5/7 (4+ cœurs)  
RAM : 16 Go minimum  
Stockage : SSD 256 Go minimum  
OS : Windows 10/11, macOS 11+, ou Linux (Ubuntu 20.04+)

**Cas d’usage** : RAG avec modèles 7B quantifiés, embeddings légers, corpus < 10 000 documents

### Configuration Recommandée (RAG + Fine-tuning Léger)

Processeur : Intel Core i7/i9 ou AMD Ryzen 7/9 (8+ cœurs)  
RAM : 32 Go  
GPU : NVIDIA RTX 3060 (12 Go VRAM) ou supérieur  
Stockage : SSD NVMe 512 Go minimum  
OS : Ubuntu 22.04 LTS (meilleur support GPU)

**Cas d’usage** : RAG avancé, fine-tuning QLoRA modèles 7-13B, corpus < 100 000 documents

### Configuration Optimale (Production)

Processeur : Intel Xeon ou AMD EPYC (16+ cœurs)  
RAM : 64-128 Go  
GPU : NVIDIA RTX 4090 (24 Go VRAM) ou A100 (40-80 Go VRAM)  
Stockage : SSD NVMe 1 To+  
OS : Ubuntu 22.04 LTS Server

**Cas d’usage** : Fine-tuning modèles 70B+, RAG temps réel, multi-utilisateurs

### Choix du GPU

| GPU | VRAM | Modèles Supportés | Prix Approx. | Recommandation |
| --- | --- | --- | --- | --- |
| RTX 3060 | 12 Go | 7B quantifiés | 300-400€ | Débutant |
| RTX 3090 | 24 Go | 13B, 7B fine-tuning | 1000-1200€ | Bon rapport qualité/prix |
| RTX 4090 | 24 Go | 13B fine-tuning, 70B inférence | 1800-2000€ | Haute performance |
| A100 | 40-80 Go | 70B fine-tuning | 10000€+ | Professionnel |

**Apple Silicon (M1/M2/M3)** : - M1/M2 Pro (16 Go) : équivalent RTX 3060 - M1/M2 Max (32-64 Go) : équivalent RTX 3090 - M1/M2 Ultra (128 Go) : équivalent A100

## 2.2 Logiciels Requis

### Système d’Exploitation

**Linux (Recommandé pour GPU)**

# Ubuntu 22.04 LTS recommandé  
lsb\_release -a  
# Description: Ubuntu 22.04.3 LTS

**macOS (Excellent pour Apple Silicon)**

# macOS 13+ (Ventura) recommandé pour Metal Performance  
sw\_vers  
# ProductVersion: 13.0

**Windows (Possible mais plus complexe pour GPU)**

# Windows 10/11 avec WSL2 recommandé  
wsl --version  
# Version WSL : 2.0.0.0

### Python

# Version 3.10 ou 3.11 (éviter 3.12 pour compatibilité)  
python --version  
# Python 3.11.7

**Installation Python (si nécessaire)** :

# Linux (Ubuntu/Debian)  
sudo apt update  
sudo apt install python3.11 python3.11-venv python3.11-dev  
  
# macOS (avec Homebrew)  
brew install python@3.11  
  
# Vérification  
python3.11 --version

### Pilotes et Toolkits GPU

#### NVIDIA (Linux/Windows)

**1. Vérifier le GPU**

# Lister les GPU NVIDIA  
lspci | grep -i nvidia  
# 01:00.0 VGA compatible controller: NVIDIA Corporation Device...

**2. Installer les pilotes NVIDIA**

# Ubuntu - méthode recommandée  
sudo ubuntu-drivers devices  
sudo ubuntu-drivers autoinstall  
  
# Ou installation manuelle d'une version spécifique  
sudo apt install nvidia-driver-535  
  
# Redémarrer  
sudo reboot  
  
# Vérifier l'installation  
nvidia-smi

**Sortie attendue de nvidia-smi** :

+-----------------------------------------------------------------------------+  
| NVIDIA-SMI 535.129.03 Driver Version: 535.129.03 CUDA Version: 12.2 |  
|-------------------------------+----------------------+----------------------+  
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |  
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |  
|===============================+======================+======================|  
| 0 NVIDIA GeForce ... Off | 00000000:01:00.0 On | N/A |  
| 30% 45C P8 15W / 350W | 523MiB / 24576MiB | 0% Default |  
+-------------------------------+----------------------+----------------------+

**3. Installer CUDA Toolkit**

# Télécharger depuis https://developer.nvidia.com/cuda-downloads  
# Ou via apt (Ubuntu)  
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86\_64/cuda-keyring\_1.0-1\_all.deb  
sudo dpkg -i cuda-keyring\_1.0-1\_all.deb  
sudo apt update  
sudo apt install cuda-toolkit-12-1  
  
# Ajouter au PATH  
echo 'export PATH=/usr/local/cuda-12.1/bin:$PATH' >> ~/.bashrc  
echo 'export LD\_LIBRARY\_PATH=/usr/local/cuda-12.1/lib64:$LD\_LIBRARY\_PATH' >> ~/.bashrc  
source ~/.bashrc  
  
# Vérifier  
nvcc --version  
# Cuda compilation tools, release 12.1, V12.1.105

**4. Installer cuDNN**

# Télécharger depuis https://developer.nvidia.com/cudnn  
# Nécessite un compte NVIDIA (gratuit)  
  
# Installation (exemple pour cuDNN 8.9)  
tar -xvf cudnn-linux-x86\_64-8.9.0.131\_cuda12-archive.tar.xz  
sudo cp cudnn-\*-archive/include/cudnn\*.h /usr/local/cuda/include  
sudo cp -P cudnn-\*-archive/lib/libcudnn\* /usr/local/cuda/lib64  
sudo chmod a+r /usr/local/cuda/include/cudnn\*.h /usr/local/cuda/lib64/libcudnn\*  
  
# Vérifier  
cat /usr/local/cuda/include/cudnn\_version.h | grep CUDNN\_MAJOR -A 2

#### AMD (Linux uniquement)

# Installer ROCm (Ubuntu 22.04)  
sudo apt update  
wget https://repo.radeon.com/amdgpu-install/latest/ubuntu/jammy/amdgpu-install\_5.7.50700-1\_all.deb  
sudo apt install ./amdgpu-install\_5.7.50700-1\_all.deb  
sudo amdgpu-install --usecase=rocm  
  
# Ajouter l'utilisateur au groupe  
sudo usermod -a -G render,video $LOGNAME  
  
# Redémarrer  
sudo reboot  
  
# Vérifier  
rocm-smi

#### Apple Silicon (macOS)

# Installer Xcode Command Line Tools  
xcode-select --install  
  
# Vérifier  
xcode-select -p  
# /Library/Developer/CommandLineTools  
  
# Metal est intégré, pas d'installation supplémentaire

### Gestionnaires d’Environnement

#### Conda (Recommandé)

# Télécharger Miniconda  
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86\_64.sh  
  
# Installer  
bash Miniconda3-latest-Linux-x86\_64.sh -b -p $HOME/miniconda3  
  
# Initialiser  
~/miniconda3/bin/conda init bash  
source ~/.bashrc  
  
# Vérifier  
conda --version  
# conda 23.11.0  
  
# Désactiver l'activation automatique de base  
conda config --set auto\_activate\_base false

#### Alternative : venv (Intégré à Python)

# Créer un environnement virtuel  
python3.11 -m venv ~/envs/ia-locale  
  
# Activer  
source ~/envs/ia-locale/bin/activate  
  
# Vérifier  
which python  
# /home/user/envs/ia-locale/bin/python

# 3. Installation et Configuration de l’Environnement

## 3.1 Création de l’Environnement

### Option A : Avec Conda (Recommandé)

# Créer l'environnement avec Python 3.11  
conda create -n ia-locale python=3.11 -y  
  
# Activer l'environnement  
conda activate ia-locale  
  
# Vérifier  
python --version  
# Python 3.11.7  
  
which python  
# /home/user/miniconda3/envs/ia-locale/bin/python

### Option B : Avec venv

# Créer l'environnement  
python3.11 -m venv ~/.venvs/ia-locale  
  
# Activer (Linux/macOS)  
source ~/.venvs/ia-locale/bin/activate  
  
# Activer (Windows)  
.\.venvs\ia-locale\Scripts\activate  
  
# Vérifier  
python --version  
which python

## 3.2 Installation de PyTorch

PyTorch est le framework fondamental. L’installation dépend de votre configuration matérielle.

### Pour GPU NVIDIA (CUDA 12.1)

# Activer l'environnement  
conda activate ia-locale  
  
# Installer PyTorch avec support CUDA  
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121  
  
# Vérifier l'installation  
python -c "import torch; print(f'PyTorch: {torch.\_\_version\_\_}'); print(f'CUDA disponible: {torch.cuda.is\_available()}'); print(f'GPU: {torch.cuda.get\_device\_name(0) if torch.cuda.is\_available() else \"Aucun\"}')"

**Sortie attendue** :

PyTorch: 2.1.2+cu121  
CUDA disponible: True  
GPU: NVIDIA GeForce RTX 3090

### Pour CPU uniquement

pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu  
  
python -c "import torch; print(f'PyTorch: {torch.\_\_version\_\_}'); print(f'CUDA disponible: {torch.cuda.is\_available()}')"

**Sortie attendue** :

PyTorch: 2.1.2+cpu  
CUDA disponible: False

### Pour Apple Silicon (M1/M2/M3)

pip install torch torchvision torchaudio  
  
# Vérifier Metal (GPU Apple)  
python -c "import torch; print(f'PyTorch: {torch.\_\_version\_\_}'); print(f'MPS disponible: {torch.backends.mps.is\_available()}')"

**Sortie attendue** :

PyTorch: 2.1.2  
MPS disponible: True

### Pour AMD GPU (ROCm)

pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/rocm5.7  
  
python -c "import torch; print(f'PyTorch: {torch.\_\_version\_\_}'); print(f'ROCm disponible: {torch.cuda.is\_available()}')"

## 3.3 Installation des Bibliothèques Essentielles

### Bibliothèques de Base

# Activer l'environnement  
conda activate ia-locale  
  
# Installer les bibliothèques essentielles  
pip install transformers accelerate datasets sentencepiece  
  
# Vérifier  
python -c "import transformers; print(f'Transformers: {transformers.\_\_version\_\_}')"  
# Transformers: 4.36.2

### Bibliothèques pour RAG

# Embeddings  
pip install sentence-transformers  
  
# Bases vectorielles  
pip install faiss-cpu # ou faiss-gpu si GPU NVIDIA  
pip install chromadb  
  
# Framework RAG  
pip install langchain langchain-community  
  
# Outils utilitaires  
pip install pypdf python-docx python-pptx # lecture documents  
pip install tiktoken # comptage tokens OpenAI

### Bibliothèques pour Fine-tuning

# PEFT (LoRA, QLoRA)  
pip install peft  
  
# Quantification 4-bit  
pip install bitsandbytes # Nécessite CUDA sur Linux  
  
# TRL (Transformer Reinforcement Learning)  
pip install trl  
  
# Accélération entraînement  
pip install accelerate

### Bibliothèques pour API et Serveur

# FastAPI pour serveur API  
pip install fastapi uvicorn python-multipart  
  
# Clients HTTP  
pip install requests httpx  
  
# Validation de données  
pip install pydantic

### Bibliothèques Utilitaires

# Manipulation de données  
pip install pandas numpy scipy  
  
# Visualisation  
pip install matplotlib seaborn  
  
# Progress bars  
pip install tqdm  
  
# Configuration  
pip install pyyaml python-dotenv  
  
# Logging avancé  
pip install loguru

## 3.4 Installation d’Ollama (Serveur LLM Local)

Ollama est la solution la plus simple pour exécuter des LLM localement.

### Linux

# Installation en une commande  
curl -fsSL https://ollama.com/install.sh | sh  
  
# Vérifier l'installation  
ollama --version  
# ollama version is 0.1.17  
  
# Démarrer le serveur (en arrière-plan)  
ollama serve &  
  
# Vérifier que le serveur est actif  
curl http://localhost:11434/api/tags

### macOS

# Avec Homebrew  
brew install ollama  
  
# Ou télécharger depuis https://ollama.com/download  
  
# Lancer Ollama (démarre automatiquement le serveur)  
ollama --version

### Windows

# Télécharger l'installeur depuis https://ollama.com/download  
# Installer en suivant les instructions  
  
# Vérifier  
ollama --version

### Télécharger un Modèle

# Llama 3.1 8B (recommandé pour commencer)  
ollama pull llama3.1:8b  
  
# Vérifier les modèles installés  
ollama list  
  
# NAME ID SIZE MODIFIED  
# llama3.1:8b 42182419e950 4.7 GB 2 minutes ago  
  
# Tester le modèle  
ollama run llama3.1:8b "Bonjour, qui es-tu ?"

**Autres modèles utiles** :

# Modèles généraux  
ollama pull llama3.1:70b # Plus puissant mais nécessite 40+ Go VRAM  
ollama pull mistral:7b # Excellent rapport qualité/taille  
ollama pull phi3:mini # Très petit (3.8B) mais performant  
  
# Modèles spécialisés  
ollama pull codellama:7b # Optimisé pour le code  
ollama pull llama3.1:8b-instruct # Optimisé pour instructions  
  
# Lister tous les modèles disponibles  
curl https://ollama.com/library | grep -o 'href="/library/[^"]\*"' | cut -d'"' -f2

## 3.5 Vérification Complète de l’Installation

Créons un script de vérification pour s’assurer que tout fonctionne.

# Créer le fichier de vérification  
cat > verif\_installation.py << 'EOF'  
#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
"""  
Script de vérification de l'installation complète.  
"""  
  
import sys  
  
def verifier\_bibliotheque(nom, import\_nom=None):  
 """Vérifie qu'une bibliothèque est installée."""  
 import\_nom = import\_nom or nom  
 try:  
 module = \_\_import\_\_(import\_nom)  
 version = getattr(module, '\_\_version\_\_', 'version inconnue')  
 print(f"✓ {nom}: {version}")  
 return True  
 except ImportError:  
 print(f"✗ {nom}: NON INSTALLÉ")  
 return False  
  
def verifier\_gpu():  
 """Vérifie la disponibilité du GPU."""  
 try:  
 import torch  
 if torch.cuda.is\_available():  
 print(f"\n✓ GPU NVIDIA détecté: {torch.cuda.get\_device\_name(0)}")  
 print(f" VRAM disponible: {torch.cuda.get\_device\_properties(0).total\_memory / 1e9:.2f} Go")  
 print(f" CUDA version: {torch.version.cuda}")  
 return "cuda"  
 elif hasattr(torch.backends, 'mps') and torch.backends.mps.is\_available():  
 print(f"\n✓ GPU Apple Silicon (MPS) détecté")  
 return "mps"  
 else:  
 print(f"\n⚠ Aucun GPU détecté, utilisation du CPU")  
 return "cpu"  
 except Exception as e:  
 print(f"\n✗ Erreur lors de la vérification GPU: {e}")  
 return None  
  
def verifier\_ollama():  
 """Vérifie qu'Ollama est accessible."""  
 try:  
 import requests  
 response = requests.get("http://localhost:11434/api/tags", timeout=2)  
 if response.status\_code == 200:  
 modeles = response.json().get('models', [])  
 print(f"\n✓ Ollama est actif")  
 print(f" Modèles installés: {len(modeles)}")  
 for model in modeles[:3]: # Afficher max 3 modèles  
 print(f" - {model['name']}")  
 return True  
 else:  
 print(f"\n✗ Ollama: erreur HTTP {response.status\_code}")  
 return False  
 except requests.exceptions.RequestException:  
 print(f"\n✗ Ollama n'est pas accessible (serveur non démarré?)")  
 print(f" Lancer: ollama serve")  
 return False  
 except Exception as e:  
 print(f"\n✗ Erreur Ollama: {e}")  
 return False  
  
def main():  
 """Fonction principale."""  
 print("=" \* 60)  
 print("VÉRIFICATION DE L'INSTALLATION - IA LOCALE")  
 print("=" \* 60)  
  
 print("\n📦 BIBLIOTHÈQUES PYTHON:")  
 bibliotheques = [  
 ('PyTorch', 'torch'),  
 ('Transformers', 'transformers'),  
 ('Sentence-Transformers', 'sentence\_transformers'),  
 ('FAISS', 'faiss'),  
 ('ChromaDB', 'chromadb'),  
 ('LangChain', 'langchain'),  
 ('PEFT', 'peft'),  
 ('BitsAndBytes', 'bitsandbytes'),  
 ('FastAPI', 'fastapi'),  
 ('Requests', 'requests'),  
 ]  
  
 resultats = {}  
 for nom, import\_nom in bibliotheques:  
 resultats[nom] = verifier\_bibliotheque(nom, import\_nom)  
  
 print("\n🖥️ MATÉRIEL:")  
 device = verifier\_gpu()  
  
 print("\n🤖 SERVEUR LLM:")  
 ollama\_ok = verifier\_ollama()  
  
 print("\n" + "=" \* 60)  
 print("RÉSUMÉ")  
 print("=" \* 60)  
  
 nb\_ok = sum(resultats.values())  
 nb\_total = len(bibliotheques)  
  
 print(f"\nBibliothèques: {nb\_ok}/{nb\_total} installées")  
 print(f"GPU: {device.upper() if device else 'NON DÉTECTÉ'}")  
 print(f"Ollama: {'✓ OK' if ollama\_ok else '✗ NON ACCESSIBLE'}")  
  
 if nb\_ok == nb\_total and device and ollama\_ok:  
 print("\n🎉 Installation complète et fonctionnelle !")  
 return 0  
 else:  
 print("\n⚠️ Installation incomplète, vérifier les erreurs ci-dessus")  
 return 1  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 sys.exit(main())  
EOF  
  
# Rendre exécutable  
chmod +x verif\_installation.py  
  
# Exécuter  
python verif\_installation.py

**Sortie attendue (installation complète)** :

============================================================  
VÉRIFICATION DE L'INSTALLATION - IA LOCALE  
============================================================  
  
📦 BIBLIOTHÈQUES PYTHON:  
✓ PyTorch: 2.1.2+cu121  
✓ Transformers: 4.36.2  
✓ Sentence-Transformers: 2.2.2  
✓ FAISS: 1.7.4  
✓ ChromaDB: 0.4.18  
✓ LangChain: 0.1.0  
✓ PEFT: 0.7.1  
✓ BitsAndBytes: 0.41.3  
✓ FastAPI: 0.108.0  
✓ Requests: 2.31.0  
  
🖥️ MATÉRIEL:  
  
✓ GPU NVIDIA détecté: NVIDIA GeForce RTX 3090  
 VRAM disponible: 24.00 Go  
 CUDA version: 12.1  
  
🤖 SERVEUR LLM:  
  
✓ Ollama est actif  
 Modèles installés: 2  
 - llama3.1:8b  
 - mistral:7b  
  
============================================================  
RÉSUMÉ  
============================================================  
  
Bibliothèques: 10/10 installées  
GPU: CUDA  
Ollama: ✓ OK  
  
🎉 Installation complète et fonctionnelle !

## 3.6 Structure de Projet Recommandée

Créons la structure de dossiers pour le projet.

# Créer la structure  
mkdir -p ~/projets/ia-locale  
cd ~/projets/ia-locale  
  
# Créer les sous-dossiers  
mkdir -p {data/{raw,processed,chunks},models/{embeddings,llm,adapters},index/{faiss,chroma},scripts,tests,config,logs,docs}  
  
# Créer les fichiers de base  
touch config/config.yaml  
touch scripts/\_\_init\_\_.py  
touch tests/\_\_init\_\_.py  
touch README.md  
touch requirements.txt  
touch .gitignore  
  
# Structure finale  
tree -L 2

**Structure** :

ia-locale/  
├── config/  
│ └── config.yaml # Configuration centrale  
├── data/  
│ ├── raw/ # Données brutes  
│ ├── processed/ # Données nettoyées  
│ └── chunks/ # Documents segmentés  
├── models/  
│ ├── embeddings/ # Modèles d'embeddings  
│ ├── llm/ # LLMs locaux  
│ └── adapters/ # Adaptateurs LoRA  
├── index/  
│ ├── faiss/ # Index FAISS  
│ └── chroma/ # Base Chroma  
├── scripts/  
│ ├── \_\_init\_\_.py  
│ ├── 01\_preparation\_donnees.py  
│ ├── 02\_indexation.py  
│ ├── 03\_rag.py  
│ ├── 04\_fine\_tuning.py  
│ └── 05\_api\_serveur.py  
├── tests/  
│ ├── \_\_init\_\_.py  
│ ├── test\_rag.py  
│ └── test\_qualite.py  
├── logs/  
├── docs/  
├── README.md  
├── requirements.txt  
└── .gitignore

### Fichier requirements.txt

cat > requirements.txt << 'EOF'  
# PyTorch (installer séparément avec la bonne version CUDA/CPU/MPS)  
# torch>=2.1.0  
# torchvision>=0.16.0  
# torchaudio>=2.1.0  
  
# Transformers et modèles  
transformers>=4.36.0  
accelerate>=0.25.0  
datasets>=2.16.0  
sentencepiece>=0.1.99  
tokenizers>=0.15.0  
  
# RAG et embeddings  
sentence-transformers>=2.2.2  
faiss-cpu>=1.7.4 # ou faiss-gpu  
chromadb>=0.4.18  
langchain>=0.1.0  
langchain-community>=0.0.10  
  
# Fine-tuning  
peft>=0.7.0  
bitsandbytes>=0.41.0  
trl>=0.7.0  
  
# API et serveur  
fastapi>=0.108.0  
uvicorn[standard]>=0.25.0  
python-multipart>=0.0.6  
requests>=2.31.0  
httpx>=0.25.0  
  
# Manipulation de données  
pandas>=2.1.0  
numpy>=1.24.0  
scipy>=1.11.0  
  
# Lecture de documents  
pypdf>=3.17.0  
python-docx>=1.1.0  
python-pptx>=0.6.23  
openpyxl>=3.1.0  
  
# Utilitaires  
pyyaml>=6.0.1  
python-dotenv>=1.0.0  
tqdm>=4.66.0  
loguru>=0.7.0  
  
# Validation  
pydantic>=2.5.0  
  
# Tests  
pytest>=7.4.0  
pytest-cov>=4.1.0  
EOF

### Fichier .gitignore

cat > .gitignore << 'EOF'  
# Python  
\_\_pycache\_\_/  
\*.py[cod]  
\*$py.class  
\*.so  
.Python  
env/  
venv/  
ENV/  
.venv/  
  
# Données et modèles (trop gros pour git)  
data/raw/\*  
data/processed/\*  
data/chunks/\*  
models/embeddings/\*  
models/llm/\*  
models/adapters/\*  
index/faiss/\*  
index/chroma/\*  
\*.bin  
\*.safetensors  
\*.gguf  
\*.pkl  
\*.parquet  
  
# Logs  
logs/\*  
\*.log  
  
# Configuration locale  
.env  
config/local.yaml  
  
# IDE  
.vscode/  
.idea/  
\*.swp  
\*.swo  
.DS\_Store  
  
# Jupyter  
.ipynb\_checkpoints/  
\*.ipynb  
  
# Exceptions (fichiers à garder)  
!data/raw/.gitkeep  
!data/processed/.gitkeep  
!models/embeddings/.gitkeep  
!index/faiss/.gitkeep  
EOF

### Fichier README.md

cat > README.md << 'EOF'  
# Projet IA Locale  
  
Système d'Intelligence Artificielle local avec RAG et fine-tuning.  
  
## Installation  
  
```bash  
# Créer l'environnement  
conda create -n ia-locale python=3.11 -y  
conda activate ia-locale  
  
# Installer PyTorch (adapter selon votre GPU)  
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121  
  
# Installer les dépendances  
pip install -r requirements.txt

## Démarrage Rapide

# 1. Préparer les données  
python scripts/01\_preparation\_donnees.py  
  
# 2. Créer l'index vectoriel  
python scripts/02\_indexation.py  
  
# 3. Lancer le serveur RAG  
python scripts/05\_api\_serveur.py

## Documentation

Voir le guide complet dans docs/guide\_technique\_detaille.md EOF

### Configuration de Base  
  
```bash  
cat > config/config.yaml << 'EOF'  
# Configuration du système IA Locale  
  
# Chemins  
paths:  
 data\_raw: "data/raw"  
 data\_processed: "data/processed"  
 data\_chunks: "data/chunks"  
 models: "models"  
 index: "index"  
 logs: "logs"  
  
# Embeddings  
embeddings:  
 model\_name: "sentence-transformers/all-MiniLM-L6-v2"  
 dimension: 384  
 device: "cuda" # ou "cpu", "mps"  
 batch\_size: 32  
  
# Base vectorielle  
vectordb:  
 type: "faiss" # ou "chroma"  
 faiss\_index\_type: "Flat" # ou "IVFFlat", "HNSW"  
 persist\_path: "index/faiss/index.bin"  
  
# LLM  
llm:  
 provider: "ollama" # ou "llamacpp", "vllm"  
 model\_name: "llama3.1:8b"  
 api\_url: "http://localhost:11434"  
 temperature: 0.7  
 max\_tokens: 512  
 top\_p: 0.9  
 top\_k: 40  
  
# Chunking  
chunking:  
 chunk\_size: 1000  
 chunk\_overlap: 200  
 separators: ["\n\n", "\n", ". ", "! ", "? ", " "]  
  
# RAG  
rag:  
 top\_k: 4  
 use\_reranker: false  
 reranker\_model: "cross-encoder/ms-marco-MiniLM-L-6-v2"  
 reranker\_top\_k: 2  
  
# Fine-tuning  
fine\_tuning:  
 lora\_r: 16  
 lora\_alpha: 32  
 lora\_dropout: 0.05  
 learning\_rate: 2e-4  
 num\_epochs: 3  
 batch\_size: 4  
 gradient\_accumulation\_steps: 4  
  
# API  
api:  
 host: "0.0.0.0"  
 port: 8000  
 reload: false  
 log\_level: "info"  
  
# Logging  
logging:  
 level: "INFO" # DEBUG, INFO, WARNING, ERROR  
 format: "%(asctime)s - %(name)s - %(levelname)s - %(message)s"  
 file: "logs/ia-locale.log"  
EOF

## 3.7 Commandes Utiles de Gestion

### Gestion de l’Environnement Conda

# Lister les environnements  
conda env list  
  
# Activer  
conda activate ia-locale  
  
# Désactiver  
conda deactivate  
  
# Exporter l'environnement (pour partager)  
conda env export > environment.yml  
  
# Créer depuis un fichier  
conda env create -f environment.yml  
  
# Supprimer l'environnement  
conda env remove -n ia-locale  
  
# Mettre à jour conda  
conda update conda  
  
# Nettoyer le cache  
conda clean --all

### Gestion des Packages pip

# Lister les packages installés  
pip list  
  
# Vérifier les packages obsolètes  
pip list --outdated  
  
# Mettre à jour un package  
pip install --upgrade transformers  
  
# Mettre à jour tous les packages (attention !)  
pip list --outdated --format=freeze | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip install -U  
  
# Installer depuis requirements.txt  
pip install -r requirements.txt  
  
# Générer requirements.txt des packages installés  
pip freeze > requirements\_freeze.txt  
  
# Désinstaller un package  
pip uninstall transformers  
  
# Voir les infos d'un package  
pip show transformers

### Gestion d’Ollama

# Lister les modèles installés  
ollama list  
  
# Télécharger un modèle  
ollama pull llama3.1:8b  
  
# Supprimer un modèle  
ollama rm llama3.1:8b  
  
# Afficher les infos d'un modèle  
ollama show llama3.1:8b  
  
# Copier un modèle (créer une variante)  
ollama cp llama3.1:8b mon-llama:latest  
  
# Voir les modèles disponibles en ligne  
curl https://ollama.com/api/tags  
  
# Démarrer le serveur  
ollama serve  
  
# Arrêter le serveur  
pkill ollama  
  
# Logs du serveur  
tail -f ~/.ollama/logs/server.log  
  
# Emplacement des modèles  
ls -lh ~/.ollama/models/

# 4. Définition du Problème et Choix de l’Approche

## 4.1 Typologie des Problèmes d’IA

### Questions à Se Poser

Avant de choisir une approche technique, il est crucial de bien définir le problème à résoudre.

**1. Quel est l’objectif métier ?** - Répondre à des questions (Q&A interne) - Résumer des documents - Extraire des informations structurées - Générer du contenu (emails, rapports) - Analyser du sentiment - Classifier des documents

**2. Quelles données ai-je à disposition ?** - Volume : < 1 000 docs, 1 000-10 000 docs, > 10 000 docs - Format : PDF, Word, emails, bases de données - Qualité : propres, bruitées, multilingues - Évolution : statiques ou mises à jour fréquentes

**3. Quelles sont mes contraintes ?** - Latence : temps réel (< 1s), acceptable (1-5s), batch - Confidentialité : critique, importante, secondaire - Budget : limité, moyen, élevé - Compétences : équipe ML expérimentée ou non

## 4.2 Arbre de Décision Technique

graph TD  
 A[Définir le besoin] --> B{Type de tâche ?}  
  
 B -->|Q&A sur documents| C{Volume de documents ?}  
 B -->|Classification| D{Données d'entraînement ?}  
 B -->|Génération| E{Style spécifique requis ?}  
  
 C -->|< 10k docs| F[RAG Simple<br/>FAISS + Ollama]  
 C -->|> 10k docs| G[RAG Avancé<br/>Qdrant + Reranker]  
  
 D -->|Oui, beaucoup| H[Fine-tuning<br/>LoRA/QLoRA]  
 D -->|Non, peu| I[Few-shot Learning<br/>Prompting]  
  
 E -->|Oui| J[Fine-tuning Style<br/>LoRA sur exemples]  
 E -->|Non| K[LLM Standard<br/>Prompting]  
  
 style F fill:#27AE60,color:#fff  
 style G fill:#F39C12,color:#fff  
 style H fill:#E74C3C,color:#fff

### Recommandations par Cas d’Usage

| Cas d’Usage | Approche | Complexité | Temps | Coût |
| --- | --- | --- | --- | --- |
| **Assistant documentation** | RAG | ⭐⭐ | 2-4 sem | 15-25 k€ |
| **Résumeur documents** | Prompting LLM | ⭐ | 1 sem | 5-10 k€ |
| **Chatbot support client** | RAG + Fine-tuning | ⭐⭐⭐ | 6-8 sem | 40-60 k€ |
| **Extracteur d’infos** | Few-shot + Regex | ⭐⭐ | 2-3 sem | 10-20 k€ |
| **Générateur rapports** | Fine-tuning | ⭐⭐⭐ | 4-6 sem | 30-50 k€ |
| **Analyseur sentiment** | Classification | ⭐⭐ | 3-4 sem | 15-30 k€ |

## 4.3 Matrice de Décision RAG vs Fine-tuning

### Critères de Choix

| Critère | RAG | Fine-tuning |
| --- | --- | --- |
| **Données évoluent souvent** | ✅ Excellent | ❌ Nécessite ré-entraînement |
| **Besoin de sources citées** | ✅ Natif | ⚠️ Possible mais complexe |
| **Style d’écriture spécifique** | ⚠️ Limité | ✅ Excellent |
| **Données d’entraînement** | ⚠️ Peu nécessaires | ❌ Beaucoup (1000+) |
| **Time-to-market** | ✅ Rapide (2-4 sem) | ⚠️ Long (6-8 sem) |
| **Coût** | ✅ Faible (10-25 k€) | ⚠️ Élevé (40-80 k€) |
| **Expertise ML requise** | ⚠️ Moyenne | ❌ Élevée |
| **Maintenabilité** | ✅ Simple | ⚠️ Complexe |

### Décision Recommandée

# Algorithme de décision simplifié  
def choisir\_approche(cas\_usage):  
 """  
 Détermine l'approche optimale selon le cas d'usage.  
  
 Args:  
 cas\_usage: Dictionnaire avec les caractéristiques du projet  
  
 Returns:  
 Approche recommandée  
 """  
 # Vérifier si RAG est approprié  
 if (cas\_usage['type'] == 'qa' and  
 cas\_usage['volume\_docs'] > 100 and  
 cas\_usage['donnees\_evoluent']):  
 return "RAG"  
  
 # Vérifier si fine-tuning est nécessaire  
 if (cas\_usage['style\_specifique'] and  
 cas\_usage['exemples\_disponibles'] > 1000):  
 return "Fine-tuning (LoRA/QLoRA)"  
  
 # Vérifier si prompting suffit  
 if (cas\_usage['type'] in ['resume', 'extraction'] and  
 cas\_usage['volume'] < 1000):  
 return "Prompting simple"  
  
 # Par défaut : RAG (le plus versatile)  
 return "RAG (approche sûre)"  
  
# Exemple d'utilisation  
projet = {  
 'type': 'qa',  
 'volume\_docs': 5000,  
 'donnees\_evoluent': True,  
 'style\_specifique': False,  
 'exemples\_disponibles': 0  
}  
  
print(choisir\_approche(projet))  
# Output: "RAG"

# 5. Préparation et Nettoyage des Données

## 5.1 Sources de Données

### Données Internes

# scripts/collecte\_donnees.py  
"""  
Script de collecte de données depuis diverses sources internes.  
"""  
  
import os  
from pathlib import Path  
from typing import List, Dict  
import shutil  
  
def collecter\_pdf(repertoire\_source: str, repertoire\_dest: str) -> List[Path]:  
 """  
 Collecte tous les fichiers PDF depuis un répertoire source.  
  
 Args:  
 repertoire\_source: Répertoire contenant les PDFs  
 repertoire\_dest: Répertoire de destination  
  
 Returns:  
 Liste des fichiers copiés  
 """  
 source = Path(repertoire\_source)  
 dest = Path(repertoire\_dest)  
 dest.mkdir(parents=True, exist\_ok=True)  
  
 fichiers\_copies = []  
  
 # Parcourir récursivement  
 for fichier\_pdf in source.rglob("\*.pdf"):  
 # Créer un nom unique pour éviter les collisions  
 nom\_relatif = fichier\_pdf.relative\_to(source)  
 fichier\_dest = dest / nom\_relatif  
  
 # Créer les sous-répertoires si nécessaire  
 fichier\_dest.parent.mkdir(parents=True, exist\_ok=True)  
  
 # Copier le fichier  
 shutil.copy2(fichier\_pdf, fichier\_dest)  
 fichiers\_copies.append(fichier\_dest)  
  
 print(f"✓ {len(fichiers\_copies)} fichiers PDF collectés")  
 return fichiers\_copies  
  
# Exemple d'utilisation  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Collecter depuis plusieurs sources  
 sources = [  
 "/chemin/vers/documentation",  
 "/chemin/vers/rapports",  
 "/chemin/vers/procedures"  
 ]  
  
 for source in sources:  
 if os.path.exists(source):  
 collecter\_pdf(source, "data/raw/pdfs")

### Lecture et Extraction

# scripts/extraction\_texte.py  
"""  
Extraction de texte depuis différents formats de fichiers.  
"""  
  
from pathlib import Path  
from typing import Dict, Optional  
import pypdf  
from docx import Document  
import json  
  
  
def extraire\_pdf(chemin\_fichier: str) -> Dict[str, any]:  
 """  
 Extrait le texte d'un fichier PDF.  
  
 Args:  
 chemin\_fichier: Chemin vers le PDF  
  
 Returns:  
 Dictionnaire avec texte et métadonnées  
 """  
 texte\_complet = []  
 metadata = {}  
  
 with open(chemin\_fichier, 'rb') as f:  
 lecteur = pypdf.PdfReader(f)  
  
 # Extraire les métadonnées  
 if lecteur.metadata:  
 metadata = {  
 'titre': lecteur.metadata.get('/Title', ''),  
 'auteur': lecteur.metadata.get('/Author', ''),  
 'sujet': lecteur.metadata.get('/Subject', ''),  
 'date\_creation': str(lecteur.metadata.get('/CreationDate', ''))  
 }  
  
 # Extraire le texte de chaque page  
 for numero\_page, page in enumerate(lecteur.pages):  
 texte\_page = page.extract\_text()  
 if texte\_page:  
 texte\_complet.append(f"--- Page {numero\_page + 1} ---\n{texte\_page}")  
  
 return {  
 'fichier': chemin\_fichier,  
 'texte': '\n\n'.join(texte\_complet),  
 'metadata': metadata,  
 'nombre\_pages': len(lecteur.pages)  
 }  
  
  
def extraire\_docx(chemin\_fichier: str) -> Dict[str, any]:  
 """  
 Extrait le texte d'un fichier Word.  
  
 Args:  
 chemin\_fichier: Chemin vers le DOCX  
  
 Returns:  
 Dictionnaire avec texte et métadonnées  
 """  
 doc = Document(chemin\_fichier)  
  
 # Extraire les paragraphes  
 paragraphes = [para.text for para in doc.paragraphs if para.text.strip()]  
  
 # Extraire les propriétés du document  
 metadata = {  
 'titre': doc.core\_properties.title or '',  
 'auteur': doc.core\_properties.author or '',  
 'sujet': doc.core\_properties.subject or '',  
 'date\_creation': str(doc.core\_properties.created) if doc.core\_properties.created else ''  
 }  
  
 return {  
 'fichier': chemin\_fichier,  
 'texte': '\n\n'.join(paragraphes),  
 'metadata': metadata,  
 'nombre\_paragraphes': len(paragraphes)  
 }  
  
  
def extraire\_texte\_fichier(chemin\_fichier: str) -> Optional[Dict]:  
 """  
 Extrait le texte d'un fichier selon son extension.  
  
 Args:  
 chemin\_fichier: Chemin vers le fichier  
  
 Returns:  
 Dictionnaire avec le contenu extrait ou None  
 """  
 fichier = Path(chemin\_fichier)  
  
 if not fichier.exists():  
 print(f"✗ Fichier introuvable: {chemin\_fichier}")  
 return None  
  
 try:  
 if fichier.suffix.lower() == '.pdf':  
 return extraire\_pdf(str(fichier))  
 elif fichier.suffix.lower() in ['.docx', '.doc']:  
 return extraire\_docx(str(fichier))  
 elif fichier.suffix.lower() in ['.txt', '.md']:  
 with open(fichier, 'r', encoding='utf-8') as f:  
 return {  
 'fichier': str(fichier),  
 'texte': f.read(),  
 'metadata': {},  
 'type': 'texte\_brut'  
 }  
 else:  
 print(f"⚠ Format non supporté: {fichier.suffix}")  
 return None  
  
 except Exception as e:  
 print(f"✗ Erreur lors de l'extraction de {fichier.name}: {e}")  
 return None  
  
  
# Traitement batch  
def extraire\_tous\_fichiers(repertoire: str, formats: List[str] = None) -> List[Dict]:  
 """  
 Extrait le texte de tous les fichiers d'un répertoire.  
  
 Args:  
 repertoire: Répertoire contenant les fichiers  
 formats: Liste des extensions à traiter (None = tous)  
  
 Returns:  
 Liste des contenus extraits  
 """  
 if formats is None:  
 formats = ['.pdf', '.docx', '.txt', '.md']  
  
 dossier = Path(repertoire)  
 contenus = []  
  
 for fichier in dossier.rglob("\*"):  
 if fichier.suffix.lower() in formats and fichier.is\_file():  
 print(f"Extraction: {fichier.name}...")  
 contenu = extraire\_texte\_fichier(str(fichier))  
 if contenu:  
 contenus.append(contenu)  
  
 print(f"\n✓ {len(contenus)} fichiers extraits avec succès")  
 return contenus  
  
  
# Sauvegarde en JSONL  
def sauvegarder\_jsonl(contenus: List[Dict], fichier\_sortie: str):  
 """  
 Sauvegarde les contenus extraits au format JSONL.  
  
 Args:  
 contenus: Liste des contenus  
 fichier\_sortie: Chemin du fichier de sortie  
 """  
 with open(fichier\_sortie, 'w', encoding='utf-8') as f:  
 for contenu in contenus:  
 json.dump(contenu, f, ensure\_ascii=False)  
 f.write('\n')  
  
 print(f"✓ Contenus sauvegardés dans {fichier\_sortie}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Extraire tous les documents  
 contenus = extraire\_tous\_fichiers("data/raw")  
  
 # Sauvegarder  
 sauvegarder\_jsonl(contenus, "data/processed/documents\_extraits.jsonl")

## 5.2 Nettoyage des Données

### Script de Nettoyage Complet

# scripts/nettoyage\_donnees.py  
"""  
Nettoyage et normalisation des données textuelles.  
"""  
  
import re  
import unicodedata  
from typing import List  
from difflib import SequenceMatcher  
  
  
def nettoyer\_texte(texte: str) -> str:  
 """  
 Nettoie un texte en supprimant les éléments indésirables.  
  
 Args:  
 texte: Texte brut à nettoyer  
  
 Returns:  
 Texte nettoyé  
 """  
 # Supprimer les balises HTML/XML  
 texte = re.sub(r'<[^>]+>', '', texte)  
  
 # Supprimer les URLs  
 texte = re.sub(  
 r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-\_@.&+]|[!\*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',  
 '',  
 texte  
 )  
  
 # Normaliser les espaces multiples  
 texte = re.sub(r'\s+', ' ', texte)  
  
 # Supprimer les espaces en début/fin de ligne  
 lignes = [ligne.strip() for ligne in texte.split('\n')]  
 texte = '\n'.join(ligne for ligne in lignes if ligne)  
  
 # Normaliser les caractères Unicode (NFD -> NFC)  
 texte = unicodedata.normalize('NFC', texte)  
  
 return texte.strip()  
  
  
def corriger\_erreurs\_ocr(texte: str) -> str:  
 """  
 Corrige les erreurs courantes d'OCR.  
  
 Args:  
 texte: Texte issu d'OCR  
  
 Returns:  
 Texte corrigé  
 """  
 corrections = {  
 r'\b0\b': 'O', # Zéro confondu avec O  
 r'\bl\b': 'I', # L minuscule confondu avec I  
 r'rn': 'm', # rn confondu avec m  
 r'vv': 'w', # vv confondu avec w  
 }  
  
 for pattern, remplacement in corrections.items():  
 texte = re.sub(pattern, remplacement, texte)  
  
 return texte  
  
  
def dedupliquer\_documents(documents: List[str], seuil: float = 0.95) -> List[str]:  
 """  
 Supprime les documents quasi-identiques.  
  
 Args:  
 documents: Liste de documents  
 seuil: Seuil de similarité (0-1)  
  
 Returns:  
 Liste de documents dédupliqués  
 """  
 documents\_uniques = []  
  
 for doc in documents:  
 est\_doublon = False  
  
 for doc\_unique in documents\_uniques:  
 # Calculer la similarité  
 similarite = SequenceMatcher(None, doc, doc\_unique).ratio()  
  
 if similarite > seuil:  
 est\_doublon = True  
 break  
  
 if not est\_doublon:  
 documents\_uniques.append(doc)  
  
 print(f"Déduplication: {len(documents)} → {len(documents\_uniques)} documents")  
 return documents\_uniques  
  
  
def pipeline\_nettoyage(texte: str, corriger\_ocr: bool = False) -> str:  
 """  
 Pipeline complet de nettoyage.  
  
 Args:  
 texte: Texte brut  
 corriger\_ocr: Appliquer les corrections OCR  
  
 Returns:  
 Texte nettoyé  
 """  
 # 1. Nettoyage de base  
 texte = nettoyer\_texte(texte)  
  
 # 2. Correction OCR si demandé  
 if corriger\_ocr:  
 texte = corriger\_erreurs\_ocr(texte)  
  
 # 3. Suppression lignes vides multiples  
 texte = re.sub(r'\n\n+', '\n\n', texte)  
  
 return texte  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Test  
 texte\_test = """  
 <html><p>Ceci est un exemple de texte.</p></html>  
  
  
 Avec des espaces multiples et des lignes vides.  
 http://example.com/lien-inutile  
 """  
  
 texte\_propre = pipeline\_nettoyage(texte\_test)  
 print("Texte nettoyé:")  
 print(texte\_propre)

**Commandes d’exécution** :

# Nettoyer tous les documents  
python scripts/nettoyage\_donnees.py  
  
# Avec logs détaillés  
python scripts/nettoyage\_donnees.py --verbose

*Le guide continue avec les sections suivantes…*

## 6. Algorithmes Clés : RAG, LoRA et Embeddings

Cette section explique en détail les algorithmes fondamentaux pour créer une IA locale.

### 6.1 RAG (Retrieval-Augmented Generation)

Le [RAG](#def-rag) combine la recherche de documents avec la génération de texte pour créer des réponses précises et contextualisées.

#### Architecture RAG

graph TB  
 A[Question utilisateur] --> B[Création embedding question]  
 B --> C[Recherche similarité vectorielle]  
 C --> D[Base de documents vectorisés]  
 D --> E[Top K documents pertinents]  
 E --> F[Construction du contexte]  
 F --> G[Prompt augmenté]  
 G --> H[LLM Local Ollama]  
 H --> I[Réponse générée]  
  
 style A fill:#5EA8A7  
 style I fill:#FE4447  
 style H fill:#F4D03F

#### Composants du RAG

**1. Embedding Model** : Transforme le texte en vecteurs numériques

| Modèle | Dimensions | Performance | Taille | Cas d’usage |
| --- | --- | --- | --- | --- |
| **all-MiniLM-L6-v2** | 384 | Rapide ⚡ | 80 MB | Documents courts, prototypage |
| **all-mpnet-base-v2** | 768 | Équilibré 🎯 | 420 MB | Usage général recommandé |
| **multilingual-e5-large** | 1024 | Multilingue 🌍 | 2.24 GB | Documents français/multilingues |
| **text-embedding-ada-002** | 1536 | Haute qualité 💎 | API OpenAI | Production avec API |

**2. Vector Database** : Stocke et recherche les embeddings

| Base vectorielle | Vitesse | Scalabilité | Persistance | Complexité |
| --- | --- | --- | --- | --- |
| **FAISS** | ⚡⚡⚡ Très rapide | Millions de vecteurs | Fichiers locaux | Moyenne |
| **Chroma** | ⚡⚡ Rapide | Centaines de milliers | SQLite intégré | Faible ✅ |
| **Milvus** | ⚡⚡ Rapide | Milliards de vecteurs | Distribué | Élevée |
| **Qdrant** | ⚡⚡⚡ Très rapide | Millions de vecteurs | Distribué | Moyenne |

**Recommandation** : Pour débuter, utilisez **Chroma** (simple) ou **FAISS** (performant).

**3. LLM** : Génère la réponse finale

#### Code RAG complet avec LangChain

"""  
Script complet d'implémentation RAG avec Ollama et Chroma.  
Nécessite : pip install langchain chromadb sentence-transformers ollama  
"""  
  
from langchain\_community.document\_loaders import DirectoryLoader, TextLoader  
from langchain.text\_splitter import RecursiveCharacterTextSplitter  
from langchain\_community.embeddings import HuggingFaceEmbeddings  
from langchain\_community.vectorstores import Chroma  
from langchain\_community.llms import Ollama  
from langchain.chains import RetrievalQA  
from langchain.prompts import PromptTemplate  
import os  
  
  
# ═══════════════════════════════════════════════════════════  
# ÉTAPE 1 : Chargement et découpage des documents  
# ═══════════════════════════════════════════════════════════  
  
def charger\_et\_decouper\_documents(repertoire\_docs: str):  
 """  
 Charge tous les documents texte d'un répertoire et les découpe en chunks.  
  
 Args:  
 repertoire\_docs: Chemin vers le dossier contenant les documents  
  
 Returns:  
 Liste de documents découpés  
 """  
 print(f"📂 Chargement des documents depuis {repertoire\_docs}...")  
  
 # Charger tous les fichiers .txt  
 loader = DirectoryLoader(  
 repertoire\_docs,  
 glob="\*\*/\*.txt",  
 loader\_cls=TextLoader,  
 loader\_kwargs={'encoding': 'utf-8'}  
 )  
 documents = loader.load()  
  
 print(f"✅ {len(documents)} documents chargés")  
  
 # Découper en chunks de 500 caractères avec overlap de 50  
 text\_splitter = RecursiveCharacterTextSplitter(  
 chunk\_size=500, # Taille de chaque morceau  
 chunk\_overlap=50, # Chevauchement entre morceaux  
 length\_function=len,  
 separators=["\n\n", "\n", ". ", " ", ""]  
 )  
  
 chunks = text\_splitter.split\_documents(documents)  
 print(f"✂️ Documents découpés en {len(chunks)} chunks")  
  
 return chunks  
  
  
# ═══════════════════════════════════════════════════════════  
# ÉTAPE 2 : Création de la base vectorielle  
# ═══════════════════════════════════════════════════════════  
  
def creer\_base\_vectorielle(chunks, persist\_directory: str = "./chroma\_db"):  
 """  
 Crée ou charge une base vectorielle Chroma avec les embeddings.  
  
 Args:  
 chunks: Liste de documents découpés  
 persist\_directory: Répertoire de persistance Chroma  
  
 Returns:  
 Base vectorielle Chroma  
 """  
 print("🧮 Création des embeddings avec all-mpnet-base-v2...")  
  
 # Modèle d'embeddings (français + anglais)  
 embeddings = HuggingFaceEmbeddings(  
 model\_name="sentence-transformers/all-mpnet-base-v2",  
 model\_kwargs={'device': 'cpu'}, # Utiliser 'cuda' si GPU disponible  
 encode\_kwargs={'normalize\_embeddings': True}  
 )  
  
 # Vérifier si la base existe déjà  
 if os.path.exists(persist\_directory):  
 print(f"📦 Chargement de la base existante depuis {persist\_directory}")  
 vectorstore = Chroma(  
 persist\_directory=persist\_directory,  
 embedding\_function=embeddings  
 )  
 else:  
 print(f"🔨 Création de la base vectorielle...")  
 vectorstore = Chroma.from\_documents(  
 documents=chunks,  
 embedding=embeddings,  
 persist\_directory=persist\_directory  
 )  
 vectorstore.persist()  
 print(f"💾 Base sauvegardée dans {persist\_directory}")  
  
 return vectorstore  
  
  
# ═══════════════════════════════════════════════════════════  
# ÉTAPE 3 : Configuration du LLM Ollama  
# ═══════════════════════════════════════════════════════════  
  
def configurer\_llm():  
 """  
 Configure le modèle Ollama local.  
  
 Returns:  
 Instance Ollama configurée  
 """  
 print("🦙 Configuration d'Ollama (llama3.1:8b)...")  
  
 llm = Ollama(  
 model="llama3.1:8b",  
 temperature=0.2, # Faible = réponses plus déterministes  
 num\_ctx=4096, # Contexte de 4096 tokens  
 num\_predict=512 # Maximum 512 tokens de réponse  
 )  
  
 return llm  
  
  
# ═══════════════════════════════════════════════════════════  
# ÉTAPE 4 : Création du prompt personnalisé  
# ═══════════════════════════════════════════════════════════  
  
def creer\_prompt\_template():  
 """  
 Crée un template de prompt optimisé pour le RAG.  
  
 Returns:  
 PromptTemplate configuré  
 """  
 template = """Tu es un assistant IA qui répond uniquement en te basant sur le contexte fourni.  
  
CONTEXTE :  
{context}  
  
QUESTION : {question}  
  
INSTRUCTIONS :  
- Réponds uniquement avec les informations du contexte ci-dessus  
- Si la réponse n'est pas dans le contexte, dis "Je n'ai pas trouvé cette information dans mes documents"  
- Sois précis et concis  
- Cite les sources si possible  
  
RÉPONSE :"""  
  
 prompt = PromptTemplate(  
 template=template,  
 input\_variables=["context", "question"]  
 )  
  
 return prompt  
  
  
# ═══════════════════════════════════════════════════════════  
# ÉTAPE 5 : Assemblage de la chaîne RAG  
# ═══════════════════════════════════════════════════════════  
  
def creer\_chaine\_rag(vectorstore, llm, prompt):  
 """  
 Assemble tous les composants en une chaîne RAG complète.  
  
 Args:  
 vectorstore: Base vectorielle Chroma  
 llm: Modèle Ollama  
 prompt: Template de prompt  
  
 Returns:  
 Chaîne RetrievalQA complète  
 """  
 print("🔗 Assemblage de la chaîne RAG...")  
  
 # Créer le retriever (récupérateur de documents)  
 retriever = vectorstore.as\_retriever(  
 search\_type="similarity", # Recherche par similarité  
 search\_kwargs={"k": 3} # Récupérer les 3 documents les plus similaires  
 )  
  
 # Créer la chaîne complète  
 qa\_chain = RetrievalQA.from\_chain\_type(  
 llm=llm,  
 chain\_type="stuff", # "stuff" = injecter tous les docs dans le prompt  
 retriever=retriever,  
 return\_source\_documents=True, # Retourner les sources  
 chain\_type\_kwargs={"prompt": prompt}  
 )  
  
 print("✅ Chaîne RAG prête !")  
 return qa\_chain  
  
  
# ═══════════════════════════════════════════════════════════  
# FONCTION PRINCIPALE  
# ═══════════════════════════════════════════════════════════  
  
def main():  
 """Point d'entrée principal du script RAG."""  
  
 # Configuration  
 REPERTOIRE\_DOCS = "./mes\_documents"  
 PERSIST\_DIR = "./chroma\_db"  
  
 # 1. Charger et découper les documents  
 chunks = charger\_et\_decouper\_documents(REPERTOIRE\_DOCS)  
  
 # 2. Créer la base vectorielle  
 vectorstore = creer\_base\_vectorielle(chunks, PERSIST\_DIR)  
  
 # 3. Configurer le LLM  
 llm = configurer\_llm()  
  
 # 4. Créer le prompt  
 prompt = creer\_prompt\_template()  
  
 # 5. Assembler la chaîne RAG  
 qa\_chain = creer\_chaine\_rag(vectorstore, llm, prompt)  
  
 # 6. Boucle interactive  
 print("\n" + "="\*60)  
 print("🤖 Système RAG prêt ! Posez vos questions.")  
 print(" (tapez 'quit' pour quitter)")  
 print("="\*60 + "\n")  
  
 while True:  
 question = input("\n❓ Question : ").strip()  
  
 if question.lower() in ['quit', 'exit', 'q']:  
 print("👋 Au revoir !")  
 break  
  
 if not question:  
 continue  
  
 # Exécuter la requête  
 print("\n🔍 Recherche en cours...")  
 resultat = qa\_chain.invoke({"query": question})  
  
 # Afficher la réponse  
 print("\n💬 Réponse :")  
 print("-" \* 60)  
 print(resultat['result'])  
 print("-" \* 60)  
  
 # Afficher les sources  
 if resultat.get('source\_documents'):  
 print("\n📚 Sources utilisées :")  
 for i, doc in enumerate(resultat['source\_documents'], 1):  
 source = doc.metadata.get('source', 'Inconnu')  
 print(f" {i}. {source}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Commandes d’exécution** :

# 1. Installer les dépendances  
pip install langchain chromadb sentence-transformers ollama  
  
# 2. Créer le dossier de documents  
mkdir mes\_documents  
echo "L'intelligence artificielle locale permet..." > mes\_documents/doc1.txt  
echo "Les modèles de langage comme Llama..." > mes\_documents/doc2.txt  
  
# 3. S'assurer qu'Ollama est lancé avec un modèle  
ollama run llama3.1:8b  
  
# 4. Lancer le script RAG  
python rag\_complet.py

### 6.2 LoRA et QLoRA (Fine-tuning efficace)

Le [LoRA](#def-lora) (Low-Rank Adaptation) permet d’adapter un modèle pré-entraîné avec très peu de paramètres supplémentaires.

#### Principe de LoRA

Au lieu de modifier tous les poids du modèle (plusieurs milliards de paramètres), LoRA :

1. **Gèle les poids originaux** du modèle (pas de modification)
2. **Ajoute des matrices de faible rang** (quelques millions de paramètres)
3. **Entraîne uniquement ces nouvelles matrices**

graph LR  
 A[Modèle pré-entraîné<br/>7B paramètres gelés] --> B{LoRA Adapter<br/>4M paramètres}  
 B --> C[Modèle adapté]  
  
 D[Vos données<br/>d'entraînement] --> B  
  
 style A fill:#E8F4F8  
 style B fill:#FFF4E6  
 style C fill:#E8F8F5  
 style D fill:#FEF5E7

**Avantages** : - 🎯 **99% moins de paramètres à entraîner** (4M vs 7B) - ⚡ **3-5x plus rapide** que le full fine-tuning - 💾 **Économie mémoire** : 8 GB au lieu de 80 GB - 🔄 **Modularité** : basculer entre plusieurs adapters

#### QLoRA : LoRA + Quantification

[QLoRA](#def-qlora) va encore plus loin en quantifiant le modèle de base :

| Caractéristique | Full Fine-tuning | LoRA | QLoRA |
| --- | --- | --- | --- |
| **Poids entraînés** | 100% (7B params) | 0.1% (4M params) | 0.1% (4M params) |
| **Mémoire GPU** | ~80 GB | ~24 GB | **~8 GB** ⭐ |
| **VRAM minimale** | 80 GB (A100) | 24 GB (RTX 3090) | **8 GB (RTX 4060)** |
| **Durée (1000 ex)** | 48h | 12h | **6h** |
| **Qualité** | 100% | 95-98% | 92-95% |

**Configuration matérielle recommandée pour QLoRA** :

* ✅ **GPU 8 GB** : RTX 4060, RTX 3060 12GB
* ✅ **GPU 12-16 GB** : RTX 3080, RTX 4070, RTX 4080
* ✅ **GPU 24 GB** : RTX 3090, RTX 4090 (modèles 13B-30B possibles)

#### Code QLoRA avec Unsloth (optimisé)

"""  
Fine-tuning QLoRA optimisé avec Unsloth pour Llama 3.1.  
Nécessite : pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"  
"""  
  
from unsloth import FastLanguageModel  
import torch  
from datasets import load\_dataset  
from trl import SFTTrainer  
from transformers import TrainingArguments  
  
  
# ═══════════════════════════════════════════════════════════  
# CONFIGURATION DU MODÈLE  
# ═══════════════════════════════════════════════════════════  
  
# Paramètres du modèle  
max\_seq\_length = 2048 # Longueur maximale des séquences  
dtype = None # Auto-détection (Float16 pour Tesla T4, V100, Bfloat16 pour Ampere+)  
load\_in\_4bit = True # Quantification 4-bit pour économiser la mémoire  
  
# Charger le modèle avec quantification 4-bit  
model, tokenizer = FastLanguageModel.from\_pretrained(  
 model\_name="unsloth/Meta-Llama-3.1-8B-bnb-4bit", # Modèle pré-quantifié  
 max\_seq\_length=max\_seq\_length,  
 dtype=dtype,  
 load\_in\_4bit=load\_in\_4bit,  
)  
  
print("✅ Modèle Llama 3.1 8B chargé en 4-bit")  
  
  
# ═══════════════════════════════════════════════════════════  
# CONFIGURATION LORA  
# ═══════════════════════════════════════════════════════════  
  
model = FastLanguageModel.get\_peft\_model(  
 model,  
 r=16, # Rang des matrices LoRA (plus élevé = plus de paramètres)  
 target\_modules=[ # Modules à adapter  
 "q\_proj", "k\_proj", "v\_proj", "o\_proj", # Attention  
 "gate\_proj", "up\_proj", "down\_proj", # MLP  
 ],  
 lora\_alpha=16, # Facteur d'échelle LoRA  
 lora\_dropout=0.0, # Dropout (0 = plus rapide)  
 bias="none", # Type de biais  
 use\_gradient\_checkpointing="unsloth", # Économie mémoire  
 random\_state=3407,  
 use\_rslora=False, # RSLoRA (expérimental)  
 loftq\_config=None,  
)  
  
print(f"✅ LoRA configuré : r={16}, alpha={16}")  
  
  
# ═══════════════════════════════════════════════════════════  
# PRÉPARATION DES DONNÉES  
# ═══════════════════════════════════════════════════════════  
  
# Format de prompt pour Llama 3.1 Instruct  
alpaca\_prompt = """Instruction :  
{}  
  
Entrée :  
{}  
  
Réponse :  
{}"""  
  
def formater\_exemples(examples):  
 """  
 Formate les exemples au format Llama 3.1 Instruct.  
  
 Args:  
 examples: Batch d'exemples du dataset  
  
 Returns:  
 Exemples formatés  
 """  
 instructions = examples["instruction"]  
 inputs = examples["input"]  
 outputs = examples["output"]  
  
 texts = []  
 for instruction, input\_text, output in zip(instructions, inputs, outputs):  
 # Créer le prompt complet  
 text = alpaca\_prompt.format(instruction, input\_text, output)  
 texts.append(text)  
  
 return {"text": texts}  
  
  
# Charger un dataset (exemple avec Alpaca français)  
# Remplacez par votre propre dataset JSONL  
dataset = load\_dataset("json", data\_files="mes\_donnees\_entrainement.jsonl", split="train")  
  
# Format attendu du JSONL :  
# {"instruction": "Résume ce texte", "input": "Lorem ipsum...", "output": "Résumé..."}  
# {"instruction": "Traduis en anglais", "input": "Bonjour", "output": "Hello"}  
  
# Formater le dataset  
dataset = dataset.map(formater\_exemples, batched=True)  
  
print(f"✅ Dataset préparé : {len(dataset)} exemples")  
  
  
# ═══════════════════════════════════════════════════════════  
# CONFIGURATION DE L'ENTRAÎNEMENT  
# ═══════════════════════════════════════════════════════════  
  
trainer = SFTTrainer(  
 model=model,  
 tokenizer=tokenizer,  
 train\_dataset=dataset,  
 dataset\_text\_field="text", # Champ contenant le texte  
 max\_seq\_length=max\_seq\_length,  
 dataset\_num\_proc=2,  
 packing=False, # Concaténer plusieurs exemples (False = plus simple)  
 args=TrainingArguments(  
 # Paramètres de sortie  
 output\_dir="./resultats\_qlora",  
  
 # Paramètres d'entraînement  
 per\_device\_train\_batch\_size=2, # Batch size (augmenter si GPU > 8GB)  
 gradient\_accumulation\_steps=4, # Accumuler 4 batchs = batch effectif de 8  
 warmup\_steps=5, # Steps de warmup  
 num\_train\_epochs=3, # Nombre d'époques  
 learning\_rate=2e-4, # Taux d'apprentissage  
  
 # Optimisation mémoire  
 fp16=not torch.cuda.is\_bf16\_supported(), # Mixed precision  
 bf16=torch.cuda.is\_bf16\_supported(),  
 optim="adamw\_8bit", # Optimiseur 8-bit  
  
 # Logging et sauvegarde  
 logging\_steps=10,  
 save\_strategy="epoch",  
 save\_total\_limit=2, # Garder seulement 2 checkpoints  
  
 # Autres  
 seed=3407,  
 ),  
)  
  
print("✅ Trainer configuré")  
  
  
# ═══════════════════════════════════════════════════════════  
# LANCER L'ENTRAÎNEMENT  
# ═══════════════════════════════════════════════════════════  
  
print("\n🚀 Début de l'entraînement QLoRA...\n")  
  
# Statistiques avant entraînement  
gpu\_stats = torch.cuda.get\_device\_properties(0)  
start\_gpu\_memory = round(torch.cuda.max\_memory\_reserved() / 1024 / 1024 / 1024, 3)  
max\_memory = round(gpu\_stats.total\_memory / 1024 / 1024 / 1024, 3)  
print(f"💾 Mémoire GPU utilisée : {start\_gpu\_memory} GB / {max\_memory} GB ({start\_gpu\_memory/max\_memory\*100:.1f}%)")  
  
# Entraînement  
trainer\_stats = trainer.train()  
  
# Statistiques après entraînement  
used\_memory = round(torch.cuda.max\_memory\_reserved() / 1024 / 1024 / 1024, 3)  
used\_memory\_percent = round(used\_memory / max\_memory \* 100, 3)  
print(f"\n✅ Entraînement terminé !")  
print(f"💾 Mémoire GPU max utilisée : {used\_memory} GB / {max\_memory} GB ({used\_memory\_percent}%)")  
print(f"⏱️ Temps total : {trainer\_stats.metrics['train\_runtime']:.2f}s")  
  
  
# ═══════════════════════════════════════════════════════════  
# SAUVEGARDER LE MODÈLE  
# ═══════════════════════════════════════════════════════════  
  
# Sauvegarder uniquement l'adapter LoRA (léger, quelques MB)  
model.save\_pretrained("mon\_modele\_lora")  
tokenizer.save\_pretrained("mon\_modele\_lora")  
  
print("💾 Adapter LoRA sauvegardé dans ./mon\_modele\_lora")  
  
# Optionnel : Sauvegarder le modèle fusionné complet (lourd, plusieurs GB)  
# model.save\_pretrained\_merged("mon\_modele\_complet", tokenizer, save\_method="merged\_16bit")  
# print("💾 Modèle complet sauvegardé dans ./mon\_modele\_complet")  
  
  
# ═══════════════════════════════════════════════════════════  
# INFÉRENCE AVEC LE MODÈLE FINE-TUNÉ  
# ═══════════════════════════════════════════════════════════  
  
# Activer le mode inférence rapide  
FastLanguageModel.for\_inference(model)  
  
# Tester le modèle  
instruction = "Explique ce qu'est le machine learning"  
input\_text = ""  
  
prompt = alpaca\_prompt.format(instruction, input\_text, "")  
inputs = tokenizer(prompt, return\_tensors="pt").to("cuda")  
  
# Générer  
outputs = model.generate(  
 \*\*inputs,  
 max\_new\_tokens=256,  
 temperature=0.7,  
 top\_p=0.9,  
 do\_sample=True  
)  
  
reponse = tokenizer.decode(outputs[0], skip\_special\_tokens=True)  
print("\n📝 Test du modèle fine-tuné :")  
print("="\*60)  
print(reponse)  
print("="\*60)

**Fichier de données d’entraînement** (mes\_donnees\_entrainement.jsonl) :

{"instruction": "Résume ce document", "input": "L'intelligence artificielle est un domaine de l'informatique qui vise à créer des machines capables de réaliser des tâches nécessitant normalement l'intelligence humaine.", "output": "L'IA crée des machines intelligentes pour automatiser des tâches cognitives."}  
{"instruction": "Traduis en anglais", "input": "Bonjour, comment allez-vous ?", "output": "Hello, how are you?"}  
{"instruction": "Corrige les fautes", "input": "Les chein son partis en vacances", "output": "Les chiens sont partis en vacances"}

### 6.3 Embeddings et similarité vectorielle

Les [embeddings](#def-embeddings) transforment du texte en vecteurs numériques capturant le sens sémantique.

#### Principe des embeddings

graph LR  
 A[" 'intelligence artificielle' "] --> B[Modèle d'embedding]  
 B --> C["[0.23, -0.45, 0.78, ..., 0.12]<br/>768 dimensions"]  
  
 D[" 'machine learning' "] --> B  
 B --> E["[0.25, -0.42, 0.81, ..., 0.15]<br/>768 dimensions"]  
  
 C -.similarité cosinus: 0.89.-> E  
  
 style C fill:#E8F8F5  
 style E fill:#FEF5E7

**Deux phrases similaires = vecteurs proches** → Permet la recherche sémantique !

#### Comparaison des modèles d’embeddings

| Modèle | Dimensions | Taille | Multilingue | Performance FR | Vitesse | Recommandation |
| --- | --- | --- | --- | --- | --- | --- |
| **all-MiniLM-L6-v2** | 384 | 80 MB | ❌ Anglais | ⭐⭐ Faible | ⚡⚡⚡ | Prototypage rapide EN |
| **all-mpnet-base-v2** | 768 | 420 MB | ❌ Anglais | ⭐⭐⭐ Moyenne | ⚡⚡ | **Bon choix général EN** ✅ |
| **paraphrase-multilingual-mpnet-base-v2** | 768 | 970 MB | ✅ 50+ langues | ⭐⭐⭐⭐ Bonne | ⚡⚡ | **Recommandé FR** ✅ |
| **multilingual-e5-large** | 1024 | 2.24 GB | ✅ 100 langues | ⭐⭐⭐⭐⭐ Excellente | ⚡ | **Meilleur FR** ⭐ |
| **sentence-t5-xxl** | 768 | 4.8 GB | ❌ Anglais | ⭐⭐⭐⭐⭐ | ⚡ | Très haute qualité EN |

**Pour des documents français, utilisez** : 1. **multilingual-e5-large** (meilleur qualité) si vous avez la RAM 2. **paraphrase-multilingual-mpnet-base-v2** (bon compromis)

#### Code de comparaison d’embeddings

"""  
Comparaison de différents modèles d'embeddings pour documents français.  
"""  
  
from sentence\_transformers import SentenceTransformer  
from sklearn.metrics.pairwise import cosine\_similarity  
import numpy as np  
import time  
  
  
def tester\_modele\_embeddings(nom\_modele: str, phrases: list):  
 """  
 Teste un modèle d'embeddings sur des phrases françaises.  
  
 Args:  
 nom\_modele: Nom du modèle sur HuggingFace  
 phrases: Liste de phrases à encoder  
 """  
 print(f"\n{'='\*60}")  
 print(f"🧪 Test du modèle : {nom\_modele}")  
 print(f"{'='\*60}")  
  
 # Charger le modèle  
 debut = time.time()  
 model = SentenceTransformer(nom\_modele)  
 temps\_chargement = time.time() - debut  
 print(f"⏱️ Temps de chargement : {temps\_chargement:.2f}s")  
  
 # Encoder les phrases  
 debut = time.time()  
 embeddings = model.encode(phrases, show\_progress\_bar=False)  
 temps\_encodage = time.time() - debut  
  
 print(f"📊 Dimensions : {embeddings.shape}")  
 print(f"⏱️ Temps d'encodage : {temps\_encodage:.3f}s ({len(phrases)} phrases)")  
 print(f"⚡ Vitesse : {len(phrases)/temps\_encodage:.1f} phrases/s")  
  
 # Calculer la similarité cosinus entre toutes les paires  
 print(f"\n📐 Matrice de similarité cosinus :")  
 similarities = cosine\_similarity(embeddings)  
  
 # Afficher la matrice  
 print("\n ", end="")  
 for i in range(len(phrases)):  
 print(f"P{i+1:2d} ", end="")  
 print()  
  
 for i, row in enumerate(similarities):  
 print(f"P{i+1:2d} ", end="")  
 for sim in row:  
 print(f"{sim:5.3f} ", end="")  
 print()  
  
 # Trouver les paires les plus similaires (excluant les auto-similarités)  
 print(f"\n🔗 Paires les plus similaires :")  
 paires\_similaires = []  
 for i in range(len(phrases)):  
 for j in range(i+1, len(phrases)):  
 paires\_similaires.append((i, j, similarities[i][j]))  
  
 paires\_similaires.sort(key=lambda x: x[2], reverse=True)  
  
 for i, j, sim in paires\_similaires[:3]:  
 print(f" P{i+1} ↔ P{j+1} : {sim:.3f}")  
 print(f" '{phrases[i][:50]}...'")  
 print(f" '{phrases[j][:50]}...'")  
 print()  
  
 return embeddings, similarities  
  
  
# ═══════════════════════════════════════════════════════════  
# PHRASES DE TEST (français)  
# ═══════════════════════════════════════════════════════════  
  
phrases\_test = [  
 "L'intelligence artificielle transforme notre quotidien",  
 "Les modèles de langage comme GPT révolutionnent le NLP",  
 "Le deep learning nécessite beaucoup de données d'entraînement",  
 "J'aime les pizzas et les pâtes italiennes",  
 "La météo est ensoleillée aujourd'hui",  
 "L'IA et le machine learning sont des technologies clés",  
]  
  
print("📝 Phrases de test :")  
for i, phrase in enumerate(phrases\_test, 1):  
 print(f" P{i} : {phrase}")  
  
  
# ═══════════════════════════════════════════════════════════  
# TESTER PLUSIEURS MODÈLES  
# ═══════════════════════════════════════════════════════════  
  
modeles\_a\_tester = [  
 "sentence-transformers/all-MiniLM-L6-v2", # Rapide, anglais  
 "sentence-transformers/paraphrase-multilingual-mpnet-base-v2", # Multilingue  
 "intfloat/multilingual-e5-large", # Meilleur FR  
]  
  
resultats = {}  
  
for modele in modeles\_a\_tester:  
 try:  
 embeddings, similarities = tester\_modele\_embeddings(modele, phrases\_test)  
 resultats[modele] = (embeddings, similarities)  
 except Exception as e:  
 print(f"❌ Erreur avec {modele} : {e}")  
  
print("\n" + "="\*60)  
print("✅ Tests terminés !")  
print("="\*60)

**Sortie attendue** :

📐 Matrice de similarité cosinus :  
  
 P 1 P 2 P 3 P 4 P 5 P 6  
P 1 1.000 0.687 0.521 0.102 0.089 0.821  
P 2 0.687 1.000 0.698 0.095 0.112 0.734  
P 3 0.521 0.698 1.000 0.087 0.098 0.612  
P 4 0.102 0.095 0.087 1.000 0.234 0.098  
P 5 0.089 0.112 0.098 0.234 1.000 0.091  
P 6 0.821 0.734 0.612 0.098 0.091 1.000  
  
🔗 Paires les plus similaires :  
 P1 ↔ P6 : 0.821  
 'L'intelligence artificielle transforme notre quotidien...'  
 'L'IA et le machine learning sont des technologies clés...'

**Observations** : - P1 et P6 sont très similaires (IA/ML) - P4 et P5 sont dissimilaires des autres (pizza, météo)

### 6.4 Choix de l’algorithme : Tableau récapitulatif

| Critère | RAG | Fine-tuning (LoRA/QLoRA) | Embeddings seuls |
| --- | --- | --- | --- |
| **Cas d’usage principal** | QA sur documents évolutifs | Adaptation style/domaine spécifique | Recherche sémantique |
| **Données nécessaires** | Documents (50+) | Exemples entraînement (500+) | Documents à indexer |
| **Temps de mise en place** | 1-2 jours | 3-7 jours | Quelques heures |
| **GPU requis** | ❌ Non (CPU OK) | ✅ Oui (8 GB min) | ❌ Non (CPU OK) |
| **Mémoire nécessaire** | 4-8 GB RAM | 12-24 GB RAM + 8 GB VRAM | 2-4 GB RAM |
| **Données évolutives** | ✅ Ajout facile | ❌ Ré-entraînement nécessaire | ✅ Ajout facile |
| **Précision réponses** | ⭐⭐⭐⭐ Excellente (sources) | ⭐⭐⭐⭐⭐ Très haute | ⭐⭐⭐ Bonne |
| **Coût maintenance** | Faible | Moyen-élevé | Très faible |
| **Recommandation** | **Commencer ici** ✅ | Si style spécifique nécessaire | Pour recherche uniquement |

## 7. Implémentation RAG Avancée : Application Complète

Cette section présente une architecture complète d’application [RAG](#def-rag) en production avec interface web, gestion multi-formats et optimisations.

### 7.1 Architecture de l’application

graph TB  
 subgraph "Interface Utilisateur"  
 A[Interface Web Gradio]  
 end  
  
 subgraph "Backend RAG"  
 B[API FastAPI]  
 C[Gestionnaire de Documents]  
 D[Pipeline d'Indexation]  
 end  
  
 subgraph "Stockage"  
 E[(Base Vectorielle<br/>Chroma)]  
 F[(Métadonnées<br/>SQLite)]  
 G[Fichiers Sources]  
 end  
  
 subgraph "Modèles IA"  
 H[Embeddings<br/>HuggingFace]  
 I[LLM Ollama<br/>Llama 3.1]  
 end  
  
 A -->|Requêtes| B  
 B -->|Recherche| E  
 B -->|Métadonnées| F  
 C -->|Indexation| D  
 D -->|Embeddings| H  
 D -->|Stockage vecteurs| E  
 D -->|Métadonnées| F  
 G -->|Documents| C  
 E -->|Contexte| B  
 B -->|Génération| I  
 I -->|Réponse| B  
 B -->|Affichage| A  
  
 style A fill:#5EA8A7  
 style I fill:#FE4447  
 style E fill:#F4D03F  
 style H fill:#85C1E9

### 7.2 Structure du projet

mon\_rag\_app/  
├── src/  
│ ├── \_\_init\_\_.py  
│ ├── config.py # Configuration centralisée  
│ ├── document\_loader.py # Chargement multi-formats  
│ ├── embeddings\_manager.py # Gestion des embeddings  
│ ├── vector\_store.py # Interface Chroma  
│ ├── llm\_manager.py # Gestion Ollama  
│ ├── rag\_pipeline.py # Pipeline RAG complet  
│ └── api.py # API FastAPI  
├── ui/  
│ └── gradio\_app.py # Interface Gradio  
├── data/  
│ ├── documents/ # Documents sources  
│ ├── chroma\_db/ # Base vectorielle  
│ └── metadata.db # Base métadonnées  
├── tests/  
│ └── test\_rag.py # Tests unitaires  
├── requirements.txt  
├── docker-compose.yml # Déploiement Docker  
└── README.md

### 7.3 Code complet : Configuration

**Fichier src/config.py** :

"""  
Configuration centralisée de l'application RAG.  
"""  
  
from pathlib import Path  
from typing import Literal  
import os  
  
  
class Config:  
 """Configuration globale de l'application."""  
  
 # Chemins de base  
 BASE\_DIR = Path(\_\_file\_\_).parent.parent  
 DATA\_DIR = BASE\_DIR / "data"  
 DOCUMENTS\_DIR = DATA\_DIR / "documents"  
 CHROMA\_DIR = DATA\_DIR / "chroma\_db"  
 METADATA\_DB = DATA\_DIR / "metadata.db"  
  
 # Modèles  
 EMBEDDING\_MODEL = "sentence-transformers/paraphrase-multilingual-mpnet-base-v2"  
 EMBEDDING\_DEVICE = "cpu" # ou "cuda" si GPU disponible  
 LLM\_MODEL = "llama3.1:8b"  
 LLM\_BASE\_URL = "http://localhost:11434" # URL Ollama  
  
 # Paramètres RAG  
 CHUNK\_SIZE = 500  
 CHUNK\_OVERLAP = 50  
 TOP\_K\_RESULTS = 3 # Nombre de documents à récupérer  
 SEARCH\_TYPE: Literal["similarity", "mmr"] = "similarity"  
  
 # Paramètres LLM  
 TEMPERATURE = 0.2  
 MAX\_TOKENS = 512  
 CONTEXT\_WINDOW = 4096  
  
 # API  
 API\_HOST = "0.0.0.0"  
 API\_PORT = 8000  
 CORS\_ORIGINS = ["http://localhost:7860"] # Pour Gradio  
  
 # Logging  
 LOG\_LEVEL = "INFO"  
 LOG\_FILE = BASE\_DIR / "rag\_app.log"  
  
 # Formats de documents supportés  
 SUPPORTED\_FORMATS = {  
 ".txt": "text",  
 ".pdf": "pdf",  
 ".docx": "docx",  
 ".md": "markdown",  
 ".html": "html",  
 }  
  
 @classmethod  
 def create\_directories(cls):  
 """Crée les répertoires nécessaires s'ils n'existent pas."""  
 cls.DATA\_DIR.mkdir(exist\_ok=True)  
 cls.DOCUMENTS\_DIR.mkdir(exist\_ok=True)  
 cls.CHROMA\_DIR.mkdir(exist\_ok=True)  
  
 @classmethod  
 def validate(cls):  
 """Valide la configuration."""  
 errors = []  
  
 # Vérifier Ollama  
 try:  
 import requests  
 response = requests.get(f"{cls.LLM\_BASE\_URL}/api/tags", timeout=5)  
 if response.status\_code != 200:  
 errors.append(f"Ollama non accessible sur {cls.LLM\_BASE\_URL}")  
 except Exception as e:  
 errors.append(f"Erreur connexion Ollama : {e}")  
  
 # Vérifier le modèle Ollama  
 try:  
 import requests  
 response = requests.get(f"{cls.LLM\_BASE\_URL}/api/tags")  
 models = [m["name"] for m in response.json().get("models", [])]  
 if cls.LLM\_MODEL not in models:  
 errors.append(f"Modèle {cls.LLM\_MODEL} non installé dans Ollama")  
 except:  
 pass  
  
 if errors:  
 raise ValueError("\n".join(errors))  
  
 print("✅ Configuration validée")  
  
  
# Créer les répertoires au chargement du module  
Config.create\_directories()

### 7.4 Chargeur de documents multi-formats

**Fichier src/document\_loader.py** :

"""  
Chargeur de documents supportant multiples formats.  
"""  
  
from pathlib import Path  
from typing import List, Dict, Any  
import logging  
from datetime import datetime  
  
from langchain\_community.document\_loaders import (  
 TextLoader,  
 PyPDFLoader,  
 Docx2txtLoader,  
 UnstructuredMarkdownLoader,  
 UnstructuredHTMLLoader,  
)  
from langchain.text\_splitter import RecursiveCharacterTextSplitter  
from langchain.schema import Document  
  
from .config import Config  
  
logger = logging.getLogger(\_\_name\_\_)  
  
  
class DocumentLoader:  
 """Gestionnaire de chargement de documents multi-formats."""  
  
 def \_\_init\_\_(self):  
 self.text\_splitter = RecursiveCharacterTextSplitter(  
 chunk\_size=Config.CHUNK\_SIZE,  
 chunk\_overlap=Config.CHUNK\_OVERLAP,  
 length\_function=len,  
 separators=["\n\n", "\n", ". ", " ", ""]  
 )  
  
 # Mappage extension -> classe de loader  
 self.loader\_mapping = {  
 ".txt": TextLoader,  
 ".pdf": PyPDFLoader,  
 ".docx": Docx2txtLoader,  
 ".md": UnstructuredMarkdownLoader,  
 ".html": UnstructuredHTMLLoader,  
 }  
  
 def load\_document(self, file\_path: Path) -> List[Document]:  
 """  
 Charge un document et le découpe en chunks.  
  
 Args:  
 file\_path: Chemin vers le document  
  
 Returns:  
 Liste de chunks avec métadonnées  
 """  
 if not file\_path.exists():  
 raise FileNotFoundError(f"Fichier introuvable : {file\_path}")  
  
 extension = file\_path.suffix.lower()  
 if extension not in self.loader\_mapping:  
 raise ValueError(  
 f"Format {extension} non supporté. "  
 f"Formats acceptés : {list(self.loader\_mapping.keys())}"  
 )  
  
 logger.info(f"📄 Chargement de {file\_path.name}...")  
  
 try:  
 # Charger le document  
 LoaderClass = self.loader\_mapping[extension]  
 loader = LoaderClass(str(file\_path))  
 documents = loader.load()  
  
 # Ajouter des métadonnées  
 for doc in documents:  
 doc.metadata.update({  
 "source": str(file\_path),  
 "filename": file\_path.name,  
 "file\_type": extension,  
 "indexed\_at": datetime.now().isoformat(),  
 "file\_size": file\_path.stat().st\_size,  
 })  
  
 # Découper en chunks  
 chunks = self.text\_splitter.split\_documents(documents)  
  
 logger.info(  
 f"✅ {file\_path.name} : {len(documents)} docs → "  
 f"{len(chunks)} chunks"  
 )  
  
 return chunks  
  
 except Exception as e:  
 logger.error(f"❌ Erreur lors du chargement de {file\_path.name} : {e}")  
 raise  
  
 def load\_directory(self, directory: Path) -> List[Document]:  
 """  
 Charge tous les documents d'un répertoire récursivement.  
  
 Args:  
 directory: Répertoire contenant les documents  
  
 Returns:  
 Liste de tous les chunks  
 """  
 all\_chunks = []  
 file\_count = 0  
 error\_count = 0  
  
 logger.info(f"📂 Scan du répertoire {directory}...")  
  
 # Parcourir récursivement  
 for file\_path in directory.rglob("\*"):  
 if file\_path.is\_file() and file\_path.suffix in self.loader\_mapping:  
 try:  
 chunks = self.load\_document(file\_path)  
 all\_chunks.extend(chunks)  
 file\_count += 1  
 except Exception as e:  
 logger.error(f"Erreur avec {file\_path.name} : {e}")  
 error\_count += 1  
  
 logger.info(  
 f"✅ Chargement terminé : {file\_count} fichiers, "  
 f"{len(all\_chunks)} chunks, {error\_count} erreurs"  
 )  
  
 return all\_chunks  
  
 def get\_document\_info(self, chunks: List[Document]) -> Dict[str, Any]:  
 """  
 Extrait des statistiques sur les documents chargés.  
  
 Args:  
 chunks: Liste de chunks  
  
 Returns:  
 Dictionnaire de statistiques  
 """  
 if not chunks:  
 return {}  
  
 # Compter par type de fichier  
 file\_types = {}  
 sources = set()  
  
 for chunk in chunks:  
 file\_type = chunk.metadata.get("file\_type", "unknown")  
 file\_types[file\_type] = file\_types.get(file\_type, 0) + 1  
 sources.add(chunk.metadata.get("source", "unknown"))  
  
 # Calculer la longueur moyenne  
 avg\_length = sum(len(chunk.page\_content) for chunk in chunks) / len(chunks)  
  
 return {  
 "total\_chunks": len(chunks),  
 "total\_files": len(sources),  
 "file\_types": file\_types,  
 "avg\_chunk\_length": int(avg\_length),  
 "sources": list(sources),  
 }  
  
  
# Test du module  
if \_\_name\_\_ == "\_\_main\_\_":  
 logging.basicConfig(level=logging.INFO)  
  
 loader = DocumentLoader()  
  
 # Tester avec un fichier  
 test\_file = Config.DOCUMENTS\_DIR / "test.txt"  
 if test\_file.exists():  
 chunks = loader.load\_document(test\_file)  
 info = loader.get\_document\_info(chunks)  
 print(f"\n📊 Statistiques : {info}")

### 7.5 Gestionnaire de base vectorielle

**Fichier src/vector\_store.py** :

"""  
Gestionnaire de la base vectorielle Chroma.  
"""  
  
from typing import List, Optional, Dict, Any  
import logging  
from pathlib import Path  
  
from langchain\_community.embeddings import HuggingFaceEmbeddings  
from langchain\_community.vectorstores import Chroma  
from langchain.schema import Document  
  
from .config import Config  
  
logger = logging.getLogger(\_\_name\_\_)  
  
  
class VectorStoreManager:  
 """Gestionnaire de la base vectorielle."""  
  
 def \_\_init\_\_(self):  
 self.embeddings = self.\_initialize\_embeddings()  
 self.vectorstore = None  
  
 def \_initialize\_embeddings(self) -> HuggingFaceEmbeddings:  
 """  
 Initialise le modèle d'embeddings.  
  
 Returns:  
 Modèle d'embeddings configuré  
 """  
 logger.info(f"🧮 Chargement du modèle d'embeddings {Config.EMBEDDING\_MODEL}...")  
  
 embeddings = HuggingFaceEmbeddings(  
 model\_name=Config.EMBEDDING\_MODEL,  
 model\_kwargs={"device": Config.EMBEDDING\_DEVICE},  
 encode\_kwargs={"normalize\_embeddings": True}  
 )  
  
 logger.info("✅ Modèle d'embeddings chargé")  
 return embeddings  
  
 def create\_or\_load(self, force\_recreate: bool = False) -> Chroma:  
 """  
 Crée ou charge la base vectorielle.  
  
 Args:  
 force\_recreate: Force la recréation de la base  
  
 Returns:  
 Instance Chroma  
 """  
 if force\_recreate and Config.CHROMA\_DIR.exists():  
 logger.warning("⚠️ Suppression de la base existante...")  
 import shutil  
 shutil.rmtree(Config.CHROMA\_DIR)  
  
 if Config.CHROMA\_DIR.exists() and not force\_recreate:  
 logger.info(f"📦 Chargement de la base depuis {Config.CHROMA\_DIR}")  
 self.vectorstore = Chroma(  
 persist\_directory=str(Config.CHROMA\_DIR),  
 embedding\_function=self.embeddings  
 )  
 else:  
 logger.info("🔨 Création d'une nouvelle base vectorielle")  
 self.vectorstore = Chroma(  
 persist\_directory=str(Config.CHROMA\_DIR),  
 embedding\_function=self.embeddings  
 )  
  
 return self.vectorstore  
  
 def add\_documents(self, documents: List[Document]) -> None:  
 """  
 Ajoute des documents à la base vectorielle.  
  
 Args:  
 documents: Liste de documents à indexer  
 """  
 if not self.vectorstore:  
 self.create\_or\_load()  
  
 logger.info(f"➕ Ajout de {len(documents)} documents...")  
  
 # Ajouter par batch de 100 pour éviter les problèmes de mémoire  
 batch\_size = 100  
 for i in range(0, len(documents), batch\_size):  
 batch = documents[i:i+batch\_size]  
 self.vectorstore.add\_documents(batch)  
 logger.debug(f" Batch {i//batch\_size + 1} ajouté ({len(batch)} docs)")  
  
 # Persister  
 self.vectorstore.persist()  
 logger.info("✅ Documents ajoutés et base persistée")  
  
 def search(  
 self,  
 query: str,  
 k: int = Config.TOP\_K\_RESULTS,  
 filter\_dict: Optional[Dict[str, Any]] = None  
 ) -> List[Document]:  
 """  
 Recherche dans la base vectorielle.  
  
 Args:  
 query: Question de l'utilisateur  
 k: Nombre de résultats à retourner  
 filter\_dict: Filtre sur les métadonnées (ex: {"file\_type": ".pdf"})  
  
 Returns:  
 Liste de documents pertinents  
 """  
 if not self.vectorstore:  
 self.create\_or\_load()  
  
 logger.debug(f"🔍 Recherche : '{query[:50]}...' (top {k})")  
  
 if Config.SEARCH\_TYPE == "similarity":  
 results = self.vectorstore.similarity\_search(  
 query,  
 k=k,  
 filter=filter\_dict  
 )  
 elif Config.SEARCH\_TYPE == "mmr":  
 # Maximum Marginal Relevance : diversifie les résultats  
 results = self.vectorstore.max\_marginal\_relevance\_search(  
 query,  
 k=k,  
 filter=filter\_dict,  
 fetch\_k=k\*3 # Récupérer 3x plus puis filtrer  
 )  
  
 logger.debug(f"✅ {len(results)} documents trouvés")  
 return results  
  
 def search\_with\_scores(  
 self,  
 query: str,  
 k: int = Config.TOP\_K\_RESULTS  
 ) -> List[tuple]:  
 """  
 Recherche avec scores de similarité.  
  
 Args:  
 query: Question  
 k: Nombre de résultats  
  
 Returns:  
 Liste de tuples (document, score)  
 """  
 if not self.vectorstore:  
 self.create\_or\_load()  
  
 results = self.vectorstore.similarity\_search\_with\_score(query, k=k)  
 return results  
  
 def get\_stats(self) -> Dict[str, Any]:  
 """  
 Récupère les statistiques de la base.  
  
 Returns:  
 Dictionnaire de statistiques  
 """  
 if not self.vectorstore:  
 self.create\_or\_load()  
  
 # Chroma n'a pas de méthode count() directe, on doit récupérer la collection  
 collection = self.vectorstore.\_collection  
 count = collection.count()  
  
 return {  
 "total\_documents": count,  
 "embedding\_model": Config.EMBEDDING\_MODEL,  
 "persist\_directory": str(Config.CHROMA\_DIR),  
 }  
  
 def delete\_by\_source(self, source\_path: str) -> None:  
 """  
 Supprime tous les documents d'une source donnée.  
  
 Args:  
 source\_path: Chemin source à supprimer  
 """  
 if not self.vectorstore:  
 self.create\_or\_load()  
  
 logger.info(f"🗑️ Suppression des documents de {source\_path}")  
  
 # Récupérer tous les IDs avec ce source  
 collection = self.vectorstore.\_collection  
 results = collection.get(where={"source": source\_path})  
  
 if results and results["ids"]:  
 collection.delete(ids=results["ids"])  
 logger.info(f"✅ {len(results['ids'])} documents supprimés")  
 else:  
 logger.info("Aucun document à supprimer")  
  
  
# Test du module  
if \_\_name\_\_ == "\_\_main\_\_":  
 logging.basicConfig(level=logging.INFO)  
  
 manager = VectorStoreManager()  
 manager.create\_or\_load()  
  
 stats = manager.get\_stats()  
 print(f"\n📊 Statistiques : {stats}")

### 7.6 Pipeline RAG complet

**Fichier src/rag\_pipeline.py** :

"""  
Pipeline RAG complet avec génération de réponses.  
"""  
  
from typing import List, Dict, Any, Optional  
import logging  
  
from langchain\_community.llms import Ollama  
from langchain.chains import RetrievalQA  
from langchain.prompts import PromptTemplate  
from langchain.schema import Document  
  
from .config import Config  
from .vector\_store import VectorStoreManager  
  
logger = logging.getLogger(\_\_name\_\_)  
  
  
class RAGPipeline:  
 """Pipeline RAG complet."""  
  
 def \_\_init\_\_(self):  
 self.vector\_store\_manager = VectorStoreManager()  
 self.vector\_store\_manager.create\_or\_load()  
 self.llm = self.\_initialize\_llm()  
 self.qa\_chain = self.\_create\_qa\_chain()  
  
 def \_initialize\_llm(self) -> Ollama:  
 """  
 Initialise le modèle de langage Ollama.  
  
 Returns:  
 Instance Ollama configurée  
 """  
 logger.info(f"🦙 Configuration d'Ollama ({Config.LLM\_MODEL})...")  
  
 llm = Ollama(  
 model=Config.LLM\_MODEL,  
 base\_url=Config.LLM\_BASE\_URL,  
 temperature=Config.TEMPERATURE,  
 num\_ctx=Config.CONTEXT\_WINDOW,  
 num\_predict=Config.MAX\_TOKENS,  
 )  
  
 logger.info("✅ LLM configuré")  
 return llm  
  
 def \_create\_prompt\_template(self) -> PromptTemplate:  
 """  
 Crée le template de prompt pour le RAG.  
  
 Returns:  
 PromptTemplate configuré  
 """  
 template = """Tu es un assistant IA expert qui répond aux questions en te basant sur le contexte fourni.  
  
CONTEXTE :  
{context}  
  
QUESTION : {question}  
  
INSTRUCTIONS :  
- Réponds UNIQUEMENT avec les informations présentes dans le contexte  
- Si la réponse n'est pas dans le contexte, dis clairement "Je n'ai pas trouvé cette information dans mes documents"  
- Sois précis, structuré et concis  
- Cite les sources si possible (nom de fichier)  
- Utilise des listes à puces ou numérotées si approprié  
- Réponds en français  
  
RÉPONSE :"""  
  
 return PromptTemplate(  
 template=template,  
 input\_variables=["context", "question"]  
 )  
  
 def \_create\_qa\_chain(self) -> RetrievalQA:  
 """  
 Crée la chaîne QA complète.  
  
 Returns:  
 Chaîne RetrievalQA  
 """  
 retriever = self.vector\_store\_manager.vectorstore.as\_retriever(  
 search\_type=Config.SEARCH\_TYPE,  
 search\_kwargs={"k": Config.TOP\_K\_RESULTS}  
 )  
  
 prompt = self.\_create\_prompt\_template()  
  
 qa\_chain = RetrievalQA.from\_chain\_type(  
 llm=self.llm,  
 chain\_type="stuff",  
 retriever=retriever,  
 return\_source\_documents=True,  
 chain\_type\_kwargs={"prompt": prompt}  
 )  
  
 return qa\_chain  
  
 def query(self, question: str) -> Dict[str, Any]:  
 """  
 Traite une question et génère une réponse.  
  
 Args:  
 question: Question de l'utilisateur  
  
 Returns:  
 Dictionnaire avec réponse et métadonnées  
 """  
 logger.info(f"❓ Question : {question[:100]}...")  
  
 try:  
 # Exécuter la requête  
 result = self.qa\_chain.invoke({"query": question})  
  
 # Extraire les sources  
 sources = []  
 for doc in result.get("source\_documents", []):  
 sources.append({  
 "filename": doc.metadata.get("filename", "Inconnu"),  
 "file\_type": doc.metadata.get("file\_type", ""),  
 "excerpt": doc.page\_content[:200] + "...",  
 })  
  
 response = {  
 "question": question,  
 "answer": result["result"],  
 "sources": sources,  
 "num\_sources": len(sources),  
 }  
  
 logger.info(f"✅ Réponse générée ({len(response['answer'])} caractères)")  
 return response  
  
 except Exception as e:  
 logger.error(f"❌ Erreur lors de la requête : {e}")  
 return {  
 "question": question,  
 "answer": f"Erreur : {str(e)}",  
 "sources": [],  
 "num\_sources": 0,  
 }  
  
 def query\_with\_filter(  
 self,  
 question: str,  
 file\_type: Optional[str] = None  
 ) -> Dict[str, Any]:  
 """  
 Requête avec filtre sur le type de fichier.  
  
 Args:  
 question: Question  
 file\_type: Extension (.pdf, .txt, etc.)  
  
 Returns:  
 Réponse avec sources filtrées  
 """  
 # Rechercher avec filtre  
 filter\_dict = {"file\_type": file\_type} if file\_type else None  
 docs = self.vector\_store\_manager.search(  
 question,  
 k=Config.TOP\_K\_RESULTS,  
 filter\_dict=filter\_dict  
 )  
  
 if not docs:  
 return {  
 "question": question,  
 "answer": "Aucun document trouvé avec ce filtre.",  
 "sources": [],  
 "num\_sources": 0,  
 }  
  
 # Construire le contexte  
 context = "\n\n".join([doc.page\_content for doc in docs])  
  
 # Générer la réponse  
 prompt = self.\_create\_prompt\_template()  
 formatted\_prompt = prompt.format(context=context, question=question)  
  
 answer = self.llm.invoke(formatted\_prompt)  
  
 # Formater la réponse  
 sources = [{  
 "filename": doc.metadata.get("filename", "Inconnu"),  
 "file\_type": doc.metadata.get("file\_type", ""),  
 "excerpt": doc.page\_content[:200] + "...",  
 } for doc in docs]  
  
 return {  
 "question": question,  
 "answer": answer,  
 "sources": sources,  
 "num\_sources": len(sources),  
 "filter": file\_type,  
 }  
  
  
# Test du module  
if \_\_name\_\_ == "\_\_main\_\_":  
 logging.basicConfig(level=logging.INFO)  
  
 pipeline = RAGPipeline()  
  
 # Test  
 response = pipeline.query("Qu'est-ce que l'intelligence artificielle ?")  
 print(f"\n💬 Réponse : {response['answer']}")  
 print(f"\n📚 Sources ({response['num\_sources']}) :")  
 for i, source in enumerate(response['sources'], 1):  
 print(f" {i}. {source['filename']}")

**Commandes de test** :

# 1. Vérifier la configuration  
python -c "from src.config import Config; Config.validate()"  
  
# 2. Tester le chargement de documents  
python src/document\_loader.py  
  
# 3. Tester la base vectorielle  
python src/vector\_store.py  
  
# 4. Tester le pipeline RAG  
python src/rag\_pipeline.py

### 7.7 Points clés d’optimisation

| Optimisation | Impact | Complexité | Recommandation |
| --- | --- | --- | --- |
| **Chunking adaptatif** | ⭐⭐⭐ Élevé | Moyenne | Adapter la taille selon le type de doc |
| **Cache des embeddings** | ⭐⭐⭐ Très élevé | Faible | Persister Chroma sur disque ✅ |
| **Batch processing** | ⭐⭐ Moyen | Faible | Traiter par lots de 100 docs ✅ |
| **MMR (diversité)** | ⭐⭐ Moyen | Faible | Évite les doublons dans résultats |
| **Re-ranking** | ⭐⭐⭐ Élevé | Moyenne | Modèle de re-classement (cross-encoder) |
| **GPU pour embeddings** | ⭐⭐⭐ Très élevé | Faible | 10-50x plus rapide |

## 8. Fine-tuning Pratique : De la Préparation au Déploiement

Cette section détaille le processus complet de [fine-tuning](#def-fine-tuning) d’un modèle avec [QLoRA](#def-qlora), de la création du dataset jusqu’au déploiement.

### 8.1 Workflow de fine-tuning

graph TD  
 A[Dataset brut] --> B[Préparation dataset]  
 B --> C[Format JSONL]  
 C --> D[Validation qualité]  
 D --> E{Qualité OK?}  
 E -->|Non| F[Nettoyage/Augmentation]  
 F --> B  
 E -->|Oui| G[Split train/val/test]  
 G --> H[Configuration QLoRA]  
 H --> I[Entraînement]  
 I --> J[Évaluation métriques]  
 J --> K{Performance OK?}  
 K -->|Non| L[Ajuster hyperparamètres]  
 L --> H  
 K -->|Oui| M[Sauvegarder adapter LoRA]  
 M --> N[Test inférence]  
 N --> O[Déploiement]  
  
 style A fill:#FEF5E7  
 style M fill:#E8F8F5  
 style O fill:#5EA8A7

### 8.2 Préparation du dataset

#### 8.2.1 Formats de données

Pour le fine-tuning, plusieurs formats sont possibles :

**Format 1 : Instruction + Entrée + Sortie** (Alpaca)

{"instruction": "Résume le texte suivant", "input": "Le machine learning est...", "output": "Le ML est une branche de l'IA..."}  
{"instruction": "Traduis en anglais", "input": "Bonjour le monde", "output": "Hello world"}

**Format 2 : Conversations** (ChatML)

{"messages": [  
 {"role": "system", "content": "Tu es un assistant IA expert."},  
 {"role": "user", "content": "Qu'est-ce que le RAG ?"},  
 {"role": "assistant", "content": "Le RAG (Retrieval-Augmented Generation) combine..."}  
]}

**Format 3 : Completion** (simple)

{"text": "### Question: Qu'est-ce que Python?\n### Réponse: Python est un langage de programmation..."}

#### 8.2.2 Script de préparation de dataset

"""  
Script de préparation de dataset pour fine-tuning.  
Convertit différents formats en format Alpaca JSONL.  
"""  
  
import json  
from pathlib import Path  
from typing import List, Dict, Any  
import pandas as pd  
from datasets import load\_dataset  
import re  
  
  
def nettoyer\_texte(texte: str) -> str:  
 """  
 Nettoie un texte pour le fine-tuning.  
  
 Args:  
 texte: Texte brut  
  
 Returns:  
 Texte nettoyé  
 """  
 # Supprimer espaces multiples  
 texte = re.sub(r'\s+', ' ', texte)  
  
 # Supprimer caractères de contrôle  
 texte = re.sub(r'[\x00-\x1F\x7F-\x9F]', '', texte)  
  
 # Trim  
 texte = texte.strip()  
  
 return texte  
  
  
def convertir\_csv\_en\_alpaca(  
 csv\_path: Path,  
 col\_instruction: str,  
 col\_input: str,  
 col\_output: str  
) -> List[Dict[str, str]]:  
 """  
 Convertit un fichier CSV en format Alpaca.  
  
 Args:  
 csv\_path: Chemin vers le CSV  
 col\_instruction: Nom de la colonne instruction  
 col\_input: Nom de la colonne input  
 col\_output: Nom de la colonne output  
  
 Returns:  
 Liste d'exemples au format Alpaca  
 """  
 df = pd.read\_csv(csv\_path)  
  
 exemples = []  
 for \_, row in df.iterrows():  
 exemple = {  
 "instruction": nettoyer\_texte(str(row[col\_instruction])),  
 "input": nettoyer\_texte(str(row[col\_input])) if col\_input in row else "",  
 "output": nettoyer\_texte(str(row[col\_output])),  
 }  
 exemples.append(exemple)  
  
 print(f"✅ {len(exemples)} exemples convertis depuis {csv\_path.name}")  
 return exemples  
  
  
def creer\_dataset\_qa\_documents(  
 documents\_path: Path  
) -> List[Dict[str, str]]:  
 """  
 Crée un dataset Q&A à partir de documents.  
 Exemple : génère des paires question-réponse depuis des docs.  
  
 Args:  
 documents\_path: Répertoire contenant les documents  
  
 Returns:  
 Dataset au format Alpaca  
 """  
 exemples = []  
  
 # Exemple simple : extrait titre et premier paragraphe  
 for doc\_file in documents\_path.glob("\*.txt"):  
 with open(doc\_file, 'r', encoding='utf-8') as f:  
 contenu = f.read()  
  
 # Stratégie simple : découper en sections  
 sections = contenu.split('\n\n')  
 for i, section in enumerate(sections):  
 if len(section) > 50: # Ignorer les sections trop courtes  
 # Créer une question générique  
 question = f"Que dit le document sur la section {i+1} ?"  
  
 exemple = {  
 "instruction": question,  
 "input": "",  
 "output": nettoyer\_texte(section),  
 }  
 exemples.append(exemple)  
  
 print(f"✅ {len(exemples)} exemples Q&A générés")  
 return exemples  
  
  
def valider\_dataset(exemples: List[Dict[str, str]]) -> Dict[str, Any]:  
 """  
 Valide la qualité d'un dataset.  
  
 Args:  
 exemples: Liste d'exemples Alpaca  
  
 Returns:  
 Rapport de validation  
 """  
 stats = {  
 "total": len(exemples),  
 "vides": 0,  
 "trop\_courts": 0,  
 "trop\_longs": 0,  
 "longueur\_moyenne\_output": 0,  
 }  
  
 longueurs = []  
  
 for ex in exemples:  
 output = ex.get("output", "")  
 longueur = len(output)  
 longueurs.append(longueur)  
  
 if not output or longueur == 0:  
 stats["vides"] += 1  
 elif longueur < 10:  
 stats["trop\_courts"] += 1  
 elif longueur > 2000:  
 stats["trop\_longs"] += 1  
  
 if longueurs:  
 stats["longueur\_moyenne\_output"] = sum(longueurs) / len(longueurs)  
 stats["longueur\_min"] = min(longueurs)  
 stats["longueur\_max"] = max(longueurs)  
  
 # Calculer le pourcentage de qualité  
 problemes = stats["vides"] + stats["trop\_courts"]  
 stats["qualite\_pct"] = ((stats["total"] - problemes) / stats["total"] \* 100) if stats["total"] > 0 else 0  
  
 return stats  
  
  
def split\_dataset(  
 exemples: List[Dict[str, str]],  
 train\_ratio: float = 0.8,  
 val\_ratio: float = 0.1,  
 test\_ratio: float = 0.1,  
 shuffle: bool = True  
) -> Dict[str, List[Dict[str, str]]]:  
 """  
 Découpe le dataset en train/validation/test.  
  
 Args:  
 exemples: Liste d'exemples  
 train\_ratio: Proportion d'entraînement (0.8 = 80%)  
 val\_ratio: Proportion de validation  
 test\_ratio: Proportion de test  
 shuffle: Mélanger avant split  
  
 Returns:  
 Dict avec 'train', 'val', 'test'  
 """  
 import random  
  
 assert abs(train\_ratio + val\_ratio + test\_ratio - 1.0) < 0.001, "Les ratios doivent sommer à 1.0"  
  
 if shuffle:  
 exemples = exemples.copy()  
 random.shuffle(exemples)  
  
 total = len(exemples)  
 train\_end = int(total \* train\_ratio)  
 val\_end = train\_end + int(total \* val\_ratio)  
  
 splits = {  
 "train": exemples[:train\_end],  
 "val": exemples[train\_end:val\_end],  
 "test": exemples[val\_end:],  
 }  
  
 print(f"📊 Split réalisé :")  
 print(f" Train: {len(splits['train'])} ({len(splits['train'])/total\*100:.1f}%)")  
 print(f" Val: {len(splits['val'])} ({len(splits['val'])/total\*100:.1f}%)")  
 print(f" Test: {len(splits['test'])} ({len(splits['test'])/total\*100:.1f}%)")  
  
 return splits  
  
  
def sauvegarder\_jsonl(exemples: List[Dict[str, str]], output\_path: Path):  
 """  
 Sauvegarde le dataset au format JSONL.  
  
 Args:  
 exemples: Liste d'exemples  
 output\_path: Chemin de sortie  
 """  
 with open(output\_path, 'w', encoding='utf-8') as f:  
 for exemple in exemples:  
 f.write(json.dumps(exemple, ensure\_ascii=False) + '\n')  
  
 print(f"💾 Dataset sauvegardé : {output\_path} ({len(exemples)} exemples)")  
  
  
# ═══════════════════════════════════════════════════════════  
# EXEMPLE COMPLET  
# ═══════════════════════════════════════════════════════════  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 1. Créer un dataset exemple  
 exemples = [  
 {  
 "instruction": "Explique ce qu'est le machine learning",  
 "input": "",  
 "output": "Le machine learning est une branche de l'intelligence artificielle qui permet aux machines d'apprendre à partir de données sans être explicitement programmées.",  
 },  
 {  
 "instruction": "Traduis en anglais",  
 "input": "Bonjour le monde",  
 "output": "Hello world",  
 },  
 {  
 "instruction": "Résume ce texte",  
 "input": "Le RAG (Retrieval-Augmented Generation) combine la recherche de documents avec la génération de texte. Cette approche permet de répondre à des questions en se basant sur une base de connaissances spécifique.",  
 "output": "Le RAG combine recherche documentaire et génération de texte pour répondre aux questions via une base de connaissances.",  
 },  
 ]  
  
 # 2. Valider  
 stats = valider\_dataset(exemples)  
 print(f"\n📊 Statistiques de validation :")  
 print(f" Total: {stats['total']}")  
 print(f" Qualité: {stats['qualite\_pct']:.1f}%")  
 print(f" Longueur moyenne output: {stats['longueur\_moyenne\_output']:.0f} caractères")  
  
 # 3. Split  
 splits = split\_dataset(exemples, train\_ratio=0.7, val\_ratio=0.15, test\_ratio=0.15)  
  
 # 4. Sauvegarder  
 output\_dir = Path("./dataset\_finetuning")  
 output\_dir.mkdir(exist\_ok=True)  
  
 sauvegarder\_jsonl(splits['train'], output\_dir / "train.jsonl")  
 sauvegarder\_jsonl(splits['val'], output\_dir / "val.jsonl")  
 sauvegarder\_jsonl(splits['test'], output\_dir / "test.jsonl")  
  
 print("\n✅ Dataset prêt pour le fine-tuning !")

### 8.3 Entraînement avec monitoring

**Script d’entraînement avancé avec logging** :

"""  
Script de fine-tuning QLoRA avec monitoring et logging détaillé.  
"""  
  
from unsloth import FastLanguageModel  
import torch  
from datasets import load\_dataset  
from trl import SFTTrainer  
from transformers import TrainingArguments, TrainerCallback  
import wandb # Weights & Biases pour le monitoring (optionnel)  
from datetime import datetime  
import json  
  
  
# ═══════════════════════════════════════════════════════════  
# CONFIGURATION  
# ═══════════════════════════════════════════════════════════  
  
class Config:  
 """Configuration du fine-tuning."""  
  
 # Modèle  
 MODEL\_NAME = "unsloth/Meta-Llama-3.1-8B-bnb-4bit"  
 MAX\_SEQ\_LENGTH = 2048  
  
 # LoRA  
 LORA\_R = 16  
 LORA\_ALPHA = 16  
 LORA\_DROPOUT = 0.0  
  
 # Entraînement  
 BATCH\_SIZE = 2  
 GRADIENT\_ACCUMULATION\_STEPS = 4 # Batch effectif = 8  
 LEARNING\_RATE = 2e-4  
 NUM\_EPOCHS = 3  
 WARMUP\_STEPS = 10  
 SAVE\_STEPS = 100  
 LOGGING\_STEPS = 10  
  
 # Datasets  
 TRAIN\_FILE = "dataset\_finetuning/train.jsonl"  
 VAL\_FILE = "dataset\_finetuning/val.jsonl"  
  
 # Output  
 OUTPUT\_DIR = f"./models/llama\_finetuned\_{datetime.now().strftime('%Y%m%d\_%H%M%S')}"  
  
 # Monitoring (optionnel)  
 USE\_WANDB = False  
 WANDB\_PROJECT = "llama-finetuning"  
  
  
# ═══════════════════════════════════════════════════════════  
# CALLBACK PERSONNALISÉ POUR LOGGING  
# ═══════════════════════════════════════════════════════════  
  
class DetailedLoggingCallback(TrainerCallback):  
 """Callback pour logger les détails de l'entraînement."""  
  
 def \_\_init\_\_(self):  
 self.start\_time = None  
 self.logs = []  
  
 def on\_train\_begin(self, args, state, control, \*\*kwargs):  
 """Début de l'entraînement."""  
 self.start\_time = datetime.now()  
 print(f"\n🚀 Début de l'entraînement : {self.start\_time.strftime('%H:%M:%S')}")  
 print(f"📊 Epochs: {args.num\_train\_epochs}")  
 print(f"📦 Batch size: {args.per\_device\_train\_batch\_size}")  
 print(f"📈 Steps par epoch: {state.max\_steps // args.num\_train\_epochs}\n")  
  
 def on\_log(self, args, state, control, logs=None, \*\*kwargs):  
 """À chaque log."""  
 if logs:  
 self.logs.append(logs)  
  
 # Afficher la progression  
 if "loss" in logs:  
 step = state.global\_step  
 loss = logs["loss"]  
 lr = logs.get("learning\_rate", "N/A")  
  
 print(f" Step {step:4d} | Loss: {loss:.4f} | LR: {lr}")  
  
 def on\_epoch\_end(self, args, state, control, \*\*kwargs):  
 """Fin d'une epoch."""  
 epoch = int(state.epoch)  
 print(f"\n✅ Epoch {epoch} terminée\n")  
  
 def on\_train\_end(self, args, state, control, \*\*kwargs):  
 """Fin de l'entraînement."""  
 end\_time = datetime.now()  
 duration = end\_time - self.start\_time  
  
 print(f"\n🎉 Entraînement terminé !")  
 print(f"⏱️ Durée totale: {duration}")  
 print(f"📊 Steps totaux: {state.global\_step}")  
  
 # Sauvegarder les logs  
 log\_file = f"{args.output\_dir}/training\_logs.json"  
 with open(log\_file, 'w') as f:  
 json.dump(self.logs, f, indent=2)  
 print(f"📝 Logs sauvegardés : {log\_file}")  
  
  
# ═══════════════════════════════════════════════════════════  
# FONCTION PRINCIPALE D'ENTRAÎNEMENT  
# ═══════════════════════════════════════════════════════════  
  
def train():  
 """Lance le fine-tuning."""  
  
 # 1. Charger le modèle  
 print("📦 Chargement du modèle...")  
 model, tokenizer = FastLanguageModel.from\_pretrained(  
 model\_name=Config.MODEL\_NAME,  
 max\_seq\_length=Config.MAX\_SEQ\_LENGTH,  
 dtype=None,  
 load\_in\_4bit=True,  
 )  
  
 # 2. Configurer LoRA  
 print("🔧 Configuration LoRA...")  
 model = FastLanguageModel.get\_peft\_model(  
 model,  
 r=Config.LORA\_R,  
 target\_modules=["q\_proj", "k\_proj", "v\_proj", "o\_proj",  
 "gate\_proj", "up\_proj", "down\_proj"],  
 lora\_alpha=Config.LORA\_ALPHA,  
 lora\_dropout=Config.LORA\_DROPOUT,  
 bias="none",  
 use\_gradient\_checkpointing="unsloth",  
 random\_state=3407,  
 )  
  
 # Afficher le nombre de paramètres entraînables  
 trainable\_params = sum(p.numel() for p in model.parameters() if p.requires\_grad)  
 total\_params = sum(p.numel() for p in model.parameters())  
 print(f"🎯 Paramètres entraînables: {trainable\_params:,} / {total\_params:,} "  
 f"({100 \* trainable\_params / total\_params:.2f}%)")  
  
 # 3. Format de prompt  
 alpaca\_prompt = """Instruction :  
{}  
  
Entrée :  
{}  
  
Réponse :  
{}"""  
  
 def formater\_exemples(examples):  
 """Formate les exemples."""  
 instructions = examples["instruction"]  
 inputs = examples["input"]  
 outputs = examples["output"]  
  
 texts = []  
 for instruction, input\_text, output in zip(instructions, inputs, outputs):  
 text = alpaca\_prompt.format(instruction, input\_text, output)  
 texts.append(text)  
  
 return {"text": texts}  
  
 # 4. Charger les datasets  
 print(f"📂 Chargement du dataset d'entraînement...")  
 train\_dataset = load\_dataset("json", data\_files=Config.TRAIN\_FILE, split="train")  
 train\_dataset = train\_dataset.map(formater\_exemples, batched=True)  
  
 print(f"📂 Chargement du dataset de validation...")  
 val\_dataset = load\_dataset("json", data\_files=Config.VAL\_FILE, split="train")  
 val\_dataset = val\_dataset.map(formater\_exemples, batched=True)  
  
 print(f"✅ Train: {len(train\_dataset)} exemples")  
 print(f"✅ Val: {len(val\_dataset)} exemples")  
  
 # 5. Initialiser Weights & Biases (optionnel)  
 if Config.USE\_WANDB:  
 wandb.init(project=Config.WANDB\_PROJECT, config=vars(Config))  
  
 # 6. Configuration du Trainer  
 training\_args = TrainingArguments(  
 output\_dir=Config.OUTPUT\_DIR,  
 per\_device\_train\_batch\_size=Config.BATCH\_SIZE,  
 gradient\_accumulation\_steps=Config.GRADIENT\_ACCUMULATION\_STEPS,  
 warmup\_steps=Config.WARMUP\_STEPS,  
 num\_train\_epochs=Config.NUM\_EPOCHS,  
 learning\_rate=Config.LEARNING\_RATE,  
 fp16=not torch.cuda.is\_bf16\_supported(),  
 bf16=torch.cuda.is\_bf16\_supported(),  
 logging\_steps=Config.LOGGING\_STEPS,  
 optim="adamw\_8bit",  
 weight\_decay=0.01,  
 lr\_scheduler\_type="linear",  
 seed=3407,  
 save\_strategy="steps",  
 save\_steps=Config.SAVE\_STEPS,  
 save\_total\_limit=3,  
 evaluation\_strategy="steps",  
 eval\_steps=Config.SAVE\_STEPS,  
 load\_best\_model\_at\_end=True,  
 metric\_for\_best\_model="eval\_loss",  
 report\_to="wandb" if Config.USE\_WANDB else "none",  
 )  
  
 # 7. Créer le Trainer  
 trainer = SFTTrainer(  
 model=model,  
 tokenizer=tokenizer,  
 train\_dataset=train\_dataset,  
 eval\_dataset=val\_dataset,  
 dataset\_text\_field="text",  
 max\_seq\_length=Config.MAX\_SEQ\_LENGTH,  
 dataset\_num\_proc=2,  
 packing=False,  
 args=training\_args,  
 callbacks=[DetailedLoggingCallback()],  
 )  
  
 # 8. Statistiques GPU avant entraînement  
 if torch.cuda.is\_available():  
 gpu\_stats = torch.cuda.get\_device\_properties(0)  
 start\_memory = torch.cuda.max\_memory\_reserved() / 1024\*\*3  
 max\_memory = gpu\_stats.total\_memory / 1024\*\*3  
 print(f"\n💾 Mémoire GPU initiale: {start\_memory:.2f} GB / {max\_memory:.2f} GB")  
  
 # 9. ENTRAÎNEMENT  
 print("\n" + "="\*60)  
 print("🚀 DÉBUT DE L'ENTRAÎNEMENT")  
 print("="\*60 + "\n")  
  
 trainer\_stats = trainer.train()  
  
 # 10. Statistiques finales  
 if torch.cuda.is\_available():  
 used\_memory = torch.cuda.max\_memory\_reserved() / 1024\*\*3  
 print(f"\n💾 Mémoire GPU maximale utilisée: {used\_memory:.2f} GB / {max\_memory:.2f} GB "  
 f"({used\_memory/max\_memory\*100:.1f}%)")  
  
 print(f"\n⏱️ Temps d'entraînement: {trainer\_stats.metrics['train\_runtime']:.2f}s")  
 print(f"📉 Loss finale: {trainer\_stats.metrics['train\_loss']:.4f}")  
  
 # 11. Sauvegarder le modèle final  
 print(f"\n💾 Sauvegarde du modèle dans {Config.OUTPUT\_DIR}...")  
 model.save\_pretrained(Config.OUTPUT\_DIR)  
 tokenizer.save\_pretrained(Config.OUTPUT\_DIR)  
  
 print("\n✅ FINE-TUNING TERMINÉ !")  
 print(f"📁 Modèle sauvegardé : {Config.OUTPUT\_DIR}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 train()

### 8.4 Évaluation du modèle fine-tuné

**Script d’évaluation** :

"""  
Évaluation d'un modèle fine-tuné.  
"""  
  
from unsloth import FastLanguageModel  
import torch  
from datasets import load\_dataset  
import json  
from typing import List, Dict  
from pathlib import Path  
  
  
def charger\_modele\_finetuned(model\_path: str):  
 """  
 Charge un modèle fine-tuné.  
  
 Args:  
 model\_path: Chemin vers le modèle  
  
 Returns:  
 Modèle et tokenizer  
 """  
 print(f"📦 Chargement du modèle depuis {model\_path}...")  
  
 model, tokenizer = FastLanguageModel.from\_pretrained(  
 model\_name=model\_path,  
 max\_seq\_length=2048,  
 dtype=None,  
 load\_in\_4bit=True,  
 )  
  
 # Activer le mode inférence rapide  
 FastLanguageModel.for\_inference(model)  
  
 print("✅ Modèle chargé")  
 return model, tokenizer  
  
  
def generer\_reponse(  
 model,  
 tokenizer,  
 instruction: str,  
 input\_text: str = "",  
 max\_tokens: int = 256  
) -> str:  
 """  
 Génère une réponse avec le modèle.  
  
 Args:  
 model: Modèle  
 tokenizer: Tokenizer  
 instruction: Instruction  
 input\_text: Entrée optionnelle  
 max\_tokens: Nombre max de tokens  
  
 Returns:  
 Réponse générée  
 """  
 alpaca\_prompt = """Instruction :  
{}  
  
Entrée :  
{}  
  
Réponse :  
{}"""  
  
 prompt = alpaca\_prompt.format(instruction, input\_text, "")  
 inputs = tokenizer(prompt, return\_tensors="pt").to("cuda")  
  
 outputs = model.generate(  
 \*\*inputs,  
 max\_new\_tokens=max\_tokens,  
 temperature=0.7,  
 top\_p=0.9,  
 do\_sample=True,  
 pad\_token\_id=tokenizer.eos\_token\_id,  
 )  
  
 reponse\_complete = tokenizer.decode(outputs[0], skip\_special\_tokens=True)  
  
 # Extraire seulement la partie réponse  
 if "Réponse :" in reponse\_complete:  
 reponse = reponse\_complete.split("Réponse :")[1].strip()  
 else:  
 reponse = reponse\_complete  
  
 return reponse  
  
  
def evaluer\_sur\_dataset(  
 model,  
 tokenizer,  
 test\_file: str,  
 output\_file: str = "evaluation\_results.json"  
):  
 """  
 Évalue le modèle sur un dataset de test.  
  
 Args:  
 model: Modèle  
 tokenizer: Tokenizer  
 test\_file: Fichier JSONL de test  
 output\_file: Fichier de sortie des résultats  
 """  
 print(f"📂 Chargement du dataset de test : {test\_file}")  
 dataset = load\_dataset("json", data\_files=test\_file, split="train")  
  
 resultats = []  
  
 print(f"\n🧪 Évaluation sur {len(dataset)} exemples...\n")  
  
 for i, exemple in enumerate(dataset):  
 instruction = exemple["instruction"]  
 input\_text = exemple.get("input", "")  
 expected = exemple["output"]  
  
 # Générer la réponse  
 predicted = generer\_reponse(model, tokenizer, instruction, input\_text)  
  
 # Stocker le résultat  
 resultat = {  
 "index": i,  
 "instruction": instruction,  
 "input": input\_text,  
 "expected": expected,  
 "predicted": predicted,  
 }  
 resultats.append(resultat)  
  
 # Afficher  
 print(f"Exemple {i+1}/{len(dataset)}")  
 print(f" Instruction: {instruction[:60]}...")  
 print(f" Attendu: {expected[:60]}...")  
 print(f" Prédit: {predicted[:60]}...")  
 print()  
  
 # Sauvegarder les résultats  
 with open(output\_file, 'w', encoding='utf-8') as f:  
 json.dump(resultats, f, ensure\_ascii=False, indent=2)  
  
 print(f"✅ Résultats sauvegardés : {output\_file}")  
  
 return resultats  
  
  
# ═══════════════════════════════════════════════════════════  
# EXEMPLE D'UTILISATION  
# ═══════════════════════════════════════════════════════════  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 1. Charger le modèle fine-tuné  
 MODEL\_PATH = "./models/llama\_finetuned\_20250125\_143000"  
 model, tokenizer = charger\_modele\_finetuned(MODEL\_PATH)  
  
 # 2. Test interactif  
 print("\n" + "="\*60)  
 print("🤖 Mode test interactif")  
 print("="\*60 + "\n")  
  
 tests = [  
 ("Explique ce qu'est le RAG", ""),  
 ("Traduis en anglais", "Bonjour le monde"),  
 ("Résume ce texte", "L'intelligence artificielle locale permet..."),  
 ]  
  
 for instruction, input\_text in tests:  
 reponse = generer\_reponse(model, tokenizer, instruction, input\_text)  
 print(f"❓ {instruction}")  
 if input\_text:  
 print(f"📝 Entrée: {input\_text}")  
 print(f"💬 Réponse: {reponse}\n")  
  
 # 3. Évaluation sur dataset de test  
 # evaluer\_sur\_dataset(model, tokenizer, "dataset\_finetuning/test.jsonl")

**Commandes d’exécution** :

# 1. Préparer le dataset  
python prepare\_dataset.py  
  
# 2. Lancer l'entraînement  
python train\_qlora.py  
  
# 3. Évaluer le modèle  
python evaluate\_model.py  
  
# 4. Déployer avec Ollama  
ollama create mon\_modele\_custom -f Modelfile  
ollama run mon\_modele\_custom

### 8.5 Déploiement du modèle fine-tuné

**Fichier Modelfile pour Ollama** :

# Modelfile pour déployer un modèle fine-tuné dans Ollama  
  
FROM ./models/llama\_finetuned\_20250125\_143000  
  
# Paramètres système  
PARAMETER temperature 0.7  
PARAMETER top\_p 0.9  
PARAMETER top\_k 40  
PARAMETER num\_ctx 4096  
  
# Prompt système  
SYSTEM """Tu es un assistant IA expert qui a été spécialement entraîné pour répondre aux questions sur [VOTRE DOMAINE].  
  
Tes caractéristiques :  
- Tu es précis et factuel  
- Tu cites tes sources quand possible  
- Tu admets quand tu ne sais pas  
- Tu réponds en français  
"""  
  
# Template de conversation  
TEMPLATE """{{ if .System }}### Système:  
{{ .System }}  
  
{{ end }}### Instruction:  
{{ .Prompt }}  
  
### Réponse:  
"""

**Commandes de déploiement** :

# Créer le modèle Ollama  
ollama create mon\_ia\_custom -f Modelfile  
  
# Lister les modèles  
ollama list  
  
# Tester le modèle  
ollama run mon\_ia\_custom  
  
# Démarrer en serveur API  
ollama serve

### 8.6 Métriques et benchmarks

| Métrique | Description | Commande/Outil |
| --- | --- | --- |
| **Perplexity** | Mesure de confusion du modèle | trainer.evaluate() |
| **BLEU Score** | Similarité avec référence (traduction) | nltk.translate.bleu\_score |
| **ROUGE Score** | Qualité de résumés | rouge\_score |
| **Exact Match** | Correspondance exacte | Custom |
| **Loss** | Perte d’entraînement/validation | Logs Trainer |

## 9. Déploiement API et Production

Cette section couvre le déploiement d’une [API](#def-api) de production pour votre IA locale.

### 9.1 API FastAPI complète

**Fichier api.py** :

"""  
API FastAPI pour servir un système RAG ou un modèle fine-tuné.  
"""  
  
from fastapi import FastAPI, HTTPException, BackgroundTasks  
from fastapi.middleware.cors import CORSMiddleware  
from pydantic import BaseModel  
from typing import List, Optional, Dict, Any  
import uvicorn  
from datetime import datetime  
import logging  
  
# Importer vos modules RAG  
from src.rag\_pipeline import RAGPipeline  
from src.config import Config  
  
# Configuration du logging  
logging.basicConfig(level=logging.INFO)  
logger = logging.getLogger(\_\_name\_\_)  
  
# Initialiser l'application FastAPI  
app = FastAPI(  
 title="API IA Locale",  
 description="API pour système RAG et modèles fine-tunés",  
 version="1.0.0"  
)  
  
# Configuration CORS  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=Config.CORS\_ORIGINS,  
 allow\_credentials=True,  
 allow\_methods=["\*"],  
 allow\_headers=["\*"],  
)  
  
# Initialiser le pipeline RAG au démarrage  
pipeline = None  
  
@app.on\_event("startup")  
async def startup\_event():  
 """Initialisation au démarrage de l'API."""  
 global pipeline  
 logger.info("🚀 Démarrage de l'API...")  
  
 try:  
 pipeline = RAGPipeline()  
 logger.info("✅ Pipeline RAG initialisé")  
 except Exception as e:  
 logger.error(f"❌ Erreur initialisation pipeline : {e}")  
 raise  
  
  
# ═══════════════════════════════════════════════════════════  
# MODÈLES DE DONNÉES  
# ═══════════════════════════════════════════════════════════  
  
class QueryRequest(BaseModel):  
 """Modèle de requête."""  
 question: str  
 max\_tokens: Optional[int] = 512  
 temperature: Optional[float] = 0.7  
 file\_type\_filter: Optional[str] = None  
  
 class Config:  
 schema\_extra = {  
 "example": {  
 "question": "Qu'est-ce que le RAG ?",  
 "max\_tokens": 512,  
 "temperature": 0.7  
 }  
 }  
  
  
class Source(BaseModel):  
 """Modèle de source."""  
 filename: str  
 file\_type: str  
 excerpt: str  
  
  
class QueryResponse(BaseModel):  
 """Modèle de réponse."""  
 question: str  
 answer: str  
 sources: List[Source]  
 num\_sources: int  
 timestamp: str  
  
  
class HealthResponse(BaseModel):  
 """Modèle de santé de l'API."""  
 status: str  
 timestamp: str  
 pipeline\_ready: bool  
 version: str  
  
  
# ═══════════════════════════════════════════════════════════  
# ENDPOINTS  
# ═══════════════════════════════════════════════════════════  
  
@app.get("/", response\_model=Dict[str, str])  
async def root():  
 """Endpoint racine."""  
 return {  
 "message": "API IA Locale",  
 "version": "1.0.0",  
 "docs": "/docs"  
 }  
  
  
@app.get("/health", response\_model=HealthResponse)  
async def health():  
 """Vérifie la santé de l'API."""  
 return HealthResponse(  
 status="healthy" if pipeline else "unhealthy",  
 timestamp=datetime.now().isoformat(),  
 pipeline\_ready=pipeline is not None,  
 version="1.0.0"  
 )  
  
  
@app.post("/query", response\_model=QueryResponse)  
async def query(request: QueryRequest):  
 """  
 Traite une question et retourne une réponse.  
  
 Args:  
 request: Requête contenant la question  
  
 Returns:  
 Réponse avec sources  
 """  
 if not pipeline:  
 raise HTTPException(status\_code=503, detail="Pipeline non initialisé")  
  
 logger.info(f"📥 Question reçue : {request.question[:50]}...")  
  
 try:  
 # Exécuter la requête  
 if request.file\_type\_filter:  
 result = pipeline.query\_with\_filter(  
 request.question,  
 file\_type=request.file\_type\_filter  
 )  
 else:  
 result = pipeline.query(request.question)  
  
 # Convertir en modèle Pydantic  
 response = QueryResponse(  
 question=result["question"],  
 answer=result["answer"],  
 sources=[Source(\*\*s) for s in result["sources"]],  
 num\_sources=result["num\_sources"],  
 timestamp=datetime.now().isoformat()  
 )  
  
 logger.info(f"✅ Réponse générée ({len(response.answer)} caractères)")  
 return response  
  
 except Exception as e:  
 logger.error(f"❌ Erreur lors du traitement : {e}")  
 raise HTTPException(status\_code=500, detail=str(e))  
  
  
@app.get("/stats")  
async def get\_stats():  
 """Récupère les statistiques de la base vectorielle."""  
 if not pipeline:  
 raise HTTPException(status\_code=503, detail="Pipeline non initialisé")  
  
 try:  
 stats = pipeline.vector\_store\_manager.get\_stats()  
 return stats  
 except Exception as e:  
 logger.error(f"❌ Erreur récupération stats : {e}")  
 raise HTTPException(status\_code=500, detail=str(e))  
  
  
# ═══════════════════════════════════════════════════════════  
# LANCEMENT  
# ═══════════════════════════════════════════════════════════  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 uvicorn.run(  
 "api:app",  
 host=Config.API\_HOST,  
 port=Config.API\_PORT,  
 reload=True,  
 log\_level="info"  
 )

**Fichier docker-compose.yml** :

version: '3.8'  
  
services:  
 # Service Ollama  
 ollama:  
 image: ollama/ollama:latest  
 container\_name: ollama  
 ports:  
 - "11434:11434"  
 volumes:  
 - ollama\_data:/root/.ollama  
 restart: unless-stopped  
 deploy:  
 resources:  
 reservations:  
 devices:  
 - driver: nvidia  
 count: all  
 capabilities: [gpu]  
  
 # Service API RAG  
 api:  
 build: .  
 container\_name: ia\_locale\_api  
 ports:  
 - "8000:8000"  
 volumes:  
 - ./data:/app/data  
 - ./src:/app/src  
 environment:  
 - OLLAMA\_BASE\_URL=http://ollama:11434  
 depends\_on:  
 - ollama  
 restart: unless-stopped  
  
 # Interface Gradio (optionnel)  
 gradio:  
 build:  
 context: .  
 dockerfile: Dockerfile.gradio  
 container\_name: ia\_locale\_ui  
 ports:  
 - "7860:7860"  
 depends\_on:  
 - api  
 restart: unless-stopped  
  
volumes:  
 ollama\_data:

**Fichier Dockerfile** :

FROM python:3.11-slim  
  
WORKDIR /app  
  
# Installation des dépendances système  
RUN apt-get update && apt-get install -y \  
 build-essential \  
 curl \  
 && rm -rf /var/lib/apt/lists/\*  
  
# Copier les requirements  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
  
# Copier le code  
COPY src/ ./src/  
COPY api.py .  
  
# Exposer le port  
EXPOSE 8000  
  
# Commande de démarrage  
CMD ["uvicorn", "api:app", "--host", "0.0.0.0", "--port", "8000"]

**Commandes de déploiement** :

# 1. Construire et lancer tous les services  
docker-compose up -d  
  
# 2. Vérifier les logs  
docker-compose logs -f  
  
# 3. Tester l'API  
curl http://localhost:8000/health  
  
# 4. Arrêter les services  
docker-compose down

## 10. Évaluation et Optimisation

### 10.1 Métriques d’évaluation

| Métrique | Utilisation | Bon score | Commande |
| --- | --- | --- | --- |
| **Latence** | Temps de réponse | < 2s | time curl ... |
| **Throughput** | Requêtes/sec | > 10 req/s | apache bench |
| **Précision** | Qualité RAG | > 85% | Évaluation manuelle |
| **VRAM** | Utilisation GPU | < 80% | nvidia-smi |
| **CPU** | Utilisation CPU | < 70% | htop |

### 10.2 Optimisations clés

**1. Cache de réponses** :

from functools import lru\_cache  
import hashlib  
  
class CachedRAGPipeline(RAGPipeline):  
 """Pipeline RAG avec cache."""  
  
 @lru\_cache(maxsize=100)  
 def query\_cached(self, question: str):  
 """Requête avec cache."""  
 return self.query(question)

**2. Batch processing** :

def query\_batch(questions: List[str]) -> List[Dict]:  
 """Traite plusieurs questions en lot."""  
 # Recherche vectorielle en batch  
 all\_results = vectorstore.similarity\_search\_batch(questions, k=3)  
  
 # Génération en batch  
 contexts = [format\_context(res) for res in all\_results]  
 answers = llm.generate(contexts)  
  
 return answers

**3. Quantification du modèle** :

# Quantifier en 4-bit avec llama.cpp  
python convert.py models/llama --outtype q4\_0 --outfile model\_q4.gguf  
  
# Utiliser avec Ollama  
ollama create mon\_modele\_q4 -f Modelfile.q4

### 10.3 Monitoring avec Prometheus

**Fichier prometheus.yml** :

global:  
 scrape\_interval: 15s  
  
scrape\_configs:  
 - job\_name: 'fastapi'  
 static\_configs:  
 - targets: ['localhost:8000']

## 11. Sécurité, Anonymisation et RGPD

### 11.1 Sécurité de l’API

from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials  
from fastapi import Depends, HTTPException  
import os  
  
security = HTTPBearer()  
  
def verify\_token(credentials: HTTPAuthorizationCredentials = Depends(security)):  
 """Vérifie le token d'authentification."""  
 token = credentials.credentials  
 expected\_token = os.getenv("API\_TOKEN")  
  
 if token != expected\_token:  
 raise HTTPException(status\_code=401, detail="Token invalide")  
  
 return token  
  
# Protéger les endpoints  
@app.post("/query", dependencies=[Depends(verify\_token)])  
async def query(request: QueryRequest):  
 ...

### 11.2 Anonymisation des données

import re  
import hashlib  
  
def anonymiser\_texte(texte: str) -> str:  
 """  
 Anonymise les données personnelles dans un texte.  
  
 Args:  
 texte: Texte à anonymiser  
  
 Returns:  
 Texte anonymisé  
 """  
 # Emails  
 texte = re.sub(  
 r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',  
 '[EMAIL]',  
 texte  
 )  
  
 # Téléphones français  
 texte = re.sub(  
 r'\b0[1-9](?:[\s.-]?\d{2}){4}\b',  
 '[TELEPHONE]',  
 texte  
 )  
  
 # Numéros de sécurité sociale  
 texte = re.sub(  
 r'\b[12]\s?\d{2}\s?\d{2}\s?\d{2}\s?\d{3}\s?\d{3}\s?\d{2}\b',  
 '[NUM\_SECU]',  
 texte  
 )  
  
 # Noms propres (heuristique simple)  
 # Note : utiliser un modèle NER pour plus de précision  
 texte = re.sub(  
 r'\b[A-Z][a-z]+ [A-Z][a-z]+\b',  
 '[NOM]',  
 texte  
 )  
  
 return texte  
  
  
def hasher\_identifiant(identifiant: str) -> str:  
 """  
 Hash un identifiant de manière irréversible.  
  
 Args:  
 identifiant: Identifiant à hasher  
  
 Returns:  
 Hash SHA-256  
 """  
 return hashlib.sha256(identifiant.encode()).hexdigest()[:16]

### 11.3 Conformité RGPD

**Checklist RGPD pour IA locale** :

* ✅ **Minimisation des données** : Ne collecter que les données nécessaires
* ✅ **Traitement local** : Toutes les données restent sur vos serveurs
* ✅ **Droit à l’oubli** : Fonction de suppression des documents indexés
* ✅ **Transparence** : Informer les utilisateurs du traitement
* ✅ **Sécurité** : Chiffrement, authentification, logs

**Fonction de suppression (droit à l’oubli)** :

def supprimer\_donnees\_utilisateur(user\_id: str):  
 """  
 Supprime toutes les données d'un utilisateur.  
  
 Args:  
 user\_id: Identifiant utilisateur  
 """  
 # 1. Supprimer de la base vectorielle  
 vectorstore\_manager.delete\_by\_metadata({"user\_id": user\_id})  
  
 # 2. Supprimer les fichiers sources  
 user\_docs\_path = f"data/documents/user\_{user\_id}"  
 if os.path.exists(user\_docs\_path):  
 shutil.rmtree(user\_docs\_path)  
  
 # 3. Supprimer les logs  
 db.execute("DELETE FROM logs WHERE user\_id = ?", (user\_id,))  
  
 logger.info(f"✅ Données de l'utilisateur {user\_id} supprimées")

### 11.4 Chiffrement des données

from cryptography.fernet import Fernet  
  
class ChiffrementManager:  
 """Gestionnaire de chiffrement pour les données sensibles."""  
  
 def \_\_init\_\_(self):  
 # Charger ou générer une clé  
 key\_file = "encryption.key"  
 if os.path.exists(key\_file):  
 with open(key\_file, 'rb') as f:  
 self.key = f.read()  
 else:  
 self.key = Fernet.generate\_key()  
 with open(key\_file, 'wb') as f:  
 f.write(self.key)  
  
 self.cipher = Fernet(self.key)  
  
 def chiffrer(self, data: str) -> bytes:  
 """Chiffre une chaîne."""  
 return self.cipher.encrypt(data.encode())  
  
 def dechiffrer(self, encrypted\_data: bytes) -> str:  
 """Déchiffre une chaîne."""  
 return self.cipher.decrypt(encrypted\_data).decode()

## 12. Conclusion et Ressources

### 12.1 Récapitulatif

Ce guide a couvert l’intégralité du processus de création d’une IA locale :

1. ✅ **Installation** (Section 3) : Environnement Python, Ollama, dépendances
2. ✅ **Choix de l’approche** (Section 4) : RAG vs Fine-tuning
3. ✅ **Préparation des données** (Section 5) : Collecte, extraction, nettoyage
4. ✅ **Algorithmes** (Section 6) : RAG, LoRA/QLoRA, Embeddings
5. ✅ **Implémentation RAG** (Section 7) : Application complète multi-formats
6. ✅ **Fine-tuning** (Section 8) : Préparation dataset, entraînement, déploiement
7. ✅ **Déploiement** (Section 9) : API FastAPI, Docker, production
8. ✅ **Optimisation** (Section 10) : Métriques, cache, monitoring
9. ✅ **Sécurité** (Section 11) : Anonymisation, RGPD, chiffrement

### 12.2 Prochaines étapes recommandées

**Pour débuter** : 1. Commencer par un projet RAG simple avec Ollama + Chroma 2. Tester sur 10-20 documents de votre domaine 3. Évaluer la qualité des réponses 4. Itérer sur le prompt et les paramètres

**Pour aller plus loin** : 1. Implémenter un fine-tuning QLoRA sur un dataset spécifique 2. Déployer une API en production avec Docker 3. Ajouter une interface Gradio pour les utilisateurs 4. Mettre en place un monitoring avec Prometheus

### 12.3 Ressources et liens utiles

**Documentation officielle** : - Ollama : https://ollama.com/docs - LangChain : https://python.langchain.com/docs - Hugging Face : https://huggingface.co/docs - Unsloth : https://github.com/unslothai/unsloth

**Modèles recommandés** : - Llama 3.1 8B : Meilleur compromis qualité/performance - Mistral 7B : Excellent pour le français - Phi-3 : Très efficace pour les petits GPUs (< 8 GB)

**Communautés** : - Reddit : r/LocalLLaMA - Discord : LangChain, Ollama - GitHub Discussions : huggingface/transformers

**Commandes de conversion finale** :

# Générer le guide DOCX avec table des matières  
pandoc guide\_technique\_detaille.md \  
 -o guide\_technique\_detaille.docx \  
 --toc \  
 --toc-depth=3 \  
 -V geometry:margin=2.5cm \  
 -V fontsize=11pt \  
 -V documentclass=report  
  
# Vérifier le fichier généré  
ls -lh guide\_technique\_detaille.docx  
  
# Générer un PDF (optionnel)  
pandoc guide\_technique\_detaille.md \  
 -o guide\_technique\_detaille.pdf \  
 --toc \  
 --pdf-engine=xelatex

**📝 FIN DU GUIDE**

**Version** : 1.0 **Dernière mise à jour** : Octobre 2025 **Auteur** : Généré avec Claude Code