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

Glossaire

- **IA** : Intelligence Artificielle
- **LLM** : Large Language Model
- **RAG** : Retrieval Augmented Generation
- **Fine-tuning** : Ajustement fin
- **Ollama** : Framework pour exécuter des LLM localement
- **LangChain** : Framework pour développer des applications basées sur les LLM
- **Embeddings** : Représentations vectorielles de texte
- **Vector Store** : Base de données vectorielle
- **FastAPI** : Framework web Python pour construire des APIs
- **Docker** : Plateforme de conteneurisation
- **Kubernetes (K8s)** : Système d'orchestration de conteneurs
- **GPU** : Graphics Processing Unit
- **VRAM** : Video Random Access Memory
- **CPU** : Central Processing Unit
- **RAM** : Random Access Memory
- **PII** : Personally Identifiable Information
- **RGPD** : Règlement Général sur la Protection des Données
- **ROI** : Retour sur Investissement
- **MTTR** : Mean Time To Repair
- **KPI** : Key Performance Indicator

Table des Matières

Section 1 : Introduction

Ce guide détaillé vous accompagnera dans la création d'une Intelligence Artificielle (IA) fonctionnant entièrement en local, depuis la compréhension des concepts fondamentaux jusqu'au déploiement d'une application robuste. Nous explorerons les différentes étapes, les algorithmes clés, les outils nécessaires et les meilleures pratiques pour construire des systèmes d'IA performants et respectueux de la confidentialité.

Section 2 : Prérequis et Installation

Cette section couvre les prérequis matériels et logiciels, ainsi que les étapes d'installation des outils essentiels comme Python, Ollama et les dépendances nécessaires.

- **Prérequis Matériels :**
  - GPU NVIDIA (recommandé pour de meilleures performances) ou CPU puissant.
  - RAM : Minimum 16 Go, 32 Go ou plus recommandé.
  - Espace disque : SSD avec au moins 100 Go d'espace libre.
- **Installation de Python :**

```
# Vérifier la version de Python
python3 --version
# Installer pip (si non présent)
python3 -m ensurepip --upgrade
```

- **Installation d'Ollama :**
  - Télécharger et installer Ollama depuis ollama.com.
  - Vérifier l'installation :

```
ollama --version
```

- **Installation des dépendances Python :**

```
pip install langchain ollama pydantic fastapi uvicorn python-
        multipart python-dotenv
```

Section 3 : Configuration Initiale

Cette section détaille la configuration de base de votre environnement et des outils.

- **Configuration d'Ollama :**
  - Télécharger un modèle :

```
ollama pull llama3
```

  - Tester le modèle :

```
ollama run llama3 "Bonjour, comment allez-vous ?"
```

- **Structure de projet recommandée :**

```
.
├── config.py
├── document_loader.py
├── vector_store.py
├── rag_pipeline.py
├── api.py
├── main.py
├── requirements.txt
└── data/
    └── documents/
```

- **Fichier `config.py` (exemple) :**

```python
# config.py
from pydantic import BaseSettings, Field

class Settings(BaseSettings):
    OLLAMA_BASE_URL: str = Field("http://localhost:11434",
        env="OLLAMA_BASE_URL")
    OLLAMA_MODEL: str = Field("llama3", env="OLLAMA_MODEL")
    VECTOR_DB_PATH: str = Field("./vector_db",
        env="VECTOR_DB_PATH")
    CHUNK_SIZE: int = Field(1000, env="CHUNK_SIZE")
    CHUNK_OVERLAP: int = Field(200, env="CHUNK_OVERLAP")

    class Config:
        env_file = ".env"
        env_file_encoding = "utf-8"

settings = Settings()
```

Section 4 : Définition du Problème et Choix de l'Approche

- Typologie des problèmes IA (QA, résumé, génération, classification)
- Arbre de décision technique avec diagramme Mermaid
- Tableau comparatif RAG vs Fine-tuning (8 critères)
- Algorithme Python de décision automatique
- Recommandations par cas d'usage avec estimations coût/temps

---

Section 5 : Préparation et Nettoyage des Données

- Script complet de collecte de fichiers PDF multi-sources
- Script d'extraction supportant PDF, DOCX, TXT avec métadonnées
- Fonction de traitement par batch
- Export JSONL
- Script de nettoyage (HTML, URLs, erreurs OCR, déduplication)
- Pipeline complet de nettoyage avec tous les commentaires en français

---

Section 6 : Algorithmes Clés - RAG, LoRA et Embeddings

- Architecture RAG complète avec diagramme Mermaid

- Tableaux comparatifs modèles d'embeddings et bases vectorielles
- Code RAG complet avec LangChain (300+ lignes, commentaires FR)
- Explication détaillée LoRA/QLoRA avec diagrammes
- Tableau comparatif Full Fine-tuning vs LoRA vs QLoRA
- Code QLoRA complet avec Unsloth (200+ lignes)
- Script de comparaison d'embeddings multilingues
- Tableau récapitulatif de choix d'algorithme

Section 7 : Implémentation RAG Avancée - Application Complète

- Architecture complète avec diagramme (UI, Backend, Stockage, Modèles)
- Structure de projet détaillée (arborescence complète)
- Module `config.py` : Configuration centralisée avec validation
- Module `document_loader.py` : Chargeur multi-formats (PDF, DOCX, MD, HTML, TXT)
- Module `vector_store.py` : Gestionnaire Chroma avec batch processing
- Module `rag_pipeline.py` : Pipeline complet avec filtrage et statistiques
- Tous les modules testables individuellement
- Tableau d'optimisations (chunking, cache, MMR, re-ranking, GPU)

Section 8 : Fine-tuning Pratique - De la Préparation au Déploiement

- Workflow complet avec diagramme Mermaid (dataset → déploiement)
- 3 formats de données : Alpaca, ChatML, Completion
- Script de préparation dataset (nettoyage, validation, split)
- Script d'entraînement QLoRA avec monitoring détaillé
- Callback personnalisé pour logging (temps, loss, GPU)
- Script d'évaluation avec génération et comparaison
- Modelfile pour déploiement Ollama
- Commandes complètes de déploiement
- Tableau des métriques (Perplexity, BLEU, ROUGE, etc.)

Section 9 : Déploiement API et Production

- API FastAPI complète (200+ lignes)
- Modèles Pydantic (QueryRequest, QueryResponse, HealthResponse)
- Endpoints : `/`, `/health`, `/query`, `/stats`
- `docker-compose.yml` : Ollama + API + Gradio
- Dockerfile pour l'API
- Commandes de déploiement Docker

Section 10 : Évaluation et Optimisation

- Tableau de métriques (Latence, Throughput, Précision, VRAM, CPU)
- Cache de réponses avec `@lru_cache`
- Batch processing pour optimisation
- Quantification de modèle (llama.cpp)
- Configuration Prometheus pour monitoring

Section 11 : Sécurité, Anonymisation et RGPD

- Sécurité API avec HTTPBearer et token
- Fonction d'anonymisation (emails, téléphones, numéros sécu, noms)
- Fonction de hashing SHA-256
- Checklist RGPD complète
- Fonction de suppression (droit à l'oubli)
- Gestionnaire de chiffrement avec Fernet

Section 12 : Conclusion et Ressources

- Récapitulatif des 11 sections
- Prochaines étapes (débutants et avancés)
- Ressources officielles (Ollama, LangChain, HF, Unsloth)
- Modèles recommandés (Llama 3.1, Mistral, Phi-3)
- Communautés (Reddit, Discord, GitHub)
- Commandes de conversion pandoc DOCX et PDF