Mise en place de l'environnement Ollama

Tarek ZLITNI

Qu'est-ce qu'Ollama ?

Outil open-source qui permet d'exécuter des LLMs directement sur une machine Locale.
Gère le téléchargement, la quantisation, le chargement en mémoire et l'exposition via des API REST.

Avantages

- Confidentialité totale : données jamais envoyées au cloud
- Zéro coût d'API : pas de facturation par requête
- Mode hors-ligne après le téléchargement initial
- Compatible LangChain, LlamaIndex, Continue.dev, Open WebUI
- Support natif quantisation GGUF via llama.cpp
- API REST compatible format OpenAI (drop-in replacement)

Architecture interne

Moteur	llama.cpp (inférence CPU/GPU)
Format	Modèles GGUF depuis le registre Ollama
Détection	CPU, GPU NVIDIA/AMD
KV-Cache	Accélère les conversations multi-tours
Serveur	HTTP local sur localhost:11434
Cycle vie	Chargement / déchargement automatique

Prérequis matériels

Configuration minimale, recommandée et optimale

Composant	Minimum	Recommandé	Optimal
RAM	8 GB	16 GB	32 GB+
GPU VRAM	Optionnel (CPU seul)	6 GB (modèles 7B Q4)	16 GB+ (13B–34B)
Stockage	10 GB libres	50 GB libres	100 GB+ (multi-modèles)
CPU	x86-64 moderne	8 coeurs+	16 coeurs+ (inférence CPU)
OS	Win 10 / macOS 12 / Ubuntu 20.04	Win 11 / macOS 13+ / Ubuntu 22.04	Linux récent recommandé

Indicateurs de performance à l'inference

GPU dédié (RTX 3090)	**~40–80 tok/s**
CPU seul (16 coeurs)	**~5–15 tok/s**

Installation d'Ollama

macOS

```
# Via Homebrew (recommandé)
brew install ollama

# Script officiel
curl -fsSL https://ollama.com/install.sh | sh

# Ou télécharger le .dmg
# ollama.com/download
```

Vérifier :

```
ollama --version
```

Résultat : ollama version 0.x.x

Service lancé automatiquement en arrière-plan

Linux (Ubuntu/Debian)

```
# Script officiel
curl -fsSL https://ollama.com/install.sh | sh

# Service systemd auto-configuré
sudo systemctl status ollama
sudo systemctl enable ollama
sudo systemctl start ollama
```

Systemd configuré automatiquement : Ollama démarre à chaque boot.

```
sudo journalctl -u ollama -f
# logs
```

Windows

- Télécharger OllamaSetup.exe

depuis ollama.com/download/windows

Exécuter en admin

Service Windows auto-démarré

Vérifier dans le Gestionnaire des tâches : ollama.exe

```
# PowerShell
ollama --version
curl http://localhost:11434
```

CUDA : installer drivers NVIDIA 12.x si GPU

Commandes essentielles

Commandes essentielles

```
# Télécharger (version par défaut)
ollama pull llama3.2
ollama pull llama3.2:3b
ollama pull llama3.2:3b-instruct-q4_K_M

# Lister les modèles locaux
ollama list

# Inspecter un modèle
ollama show llama3.2
ollama show llama3.2 --modelfile
ollama show llama3.2 --parameters
# Supprimer / Copier
ollama rm llama3.2
ollama cp llama3.2 mon-llama
```

Modèles disponibles

Usage général & chat

Modèle	Taille	VRAM	Usage
`llama3.2:1b`	0.8 GB	2 GB	Edge, tests rapides
`llama3.2:3b`	2.0 GB	4 GB	Chat léger, prototypage
`llama3.2:7b`	4.1 GB	6 GB	Usage général : recommandé
`mistral:7b`	4.1 GB	6 GB	Français, multilingue
`gemma2:9b`	5.5 GB	8 GB	Raisonnement, instruction
`qwen2.5:7b`	4.4 GB	6 GB	Multilingue, code

Code & développement

Modèle	Taille	VRAM	Spécialité
`codellama:7b`	3.8 GB	6 GB	Python, C++, Java
`codellama:13b`	7.4 GB	10 GB	Code complexe, debug
`deepseek-coder:6.7b`	3.8 GB	6 GB	Code + maths, état de l'art
`codegemma:7b`	4.5 GB	8 GB	Code Google-grade, docs
`starcoder2:7b`	4.0 GB	6 GB	80+ langages, complétion

Modèles vision (multimodal)

`llava:7b`	4.5 GB / VRAM 8 GB
`llava:13b`	7.9 GB / VRAM 12 GB
`llama3.2-vision:11b`	7.8GB/ VRAM 2 GB

ollama.com/library

Créer un Modelfile personalisé

```
# Modelfile.code-assistant
FROM codellama:7b

PARAMETER temperature 0.1
PARAMETER top_p 0.9
PARAMETER top_k 10
PARAMETER num_ctx 8192

SYSTEM """
You are an expert software engineer.
When asked to write code, always:
1. Write clean, well-commented code
2. Include error handling
3. Explain your implementation briefly
4. Suggest improvements if relevant
Only provide the code and concise explanations. No fluff.
"""
```

```
# Construire le modèle personnalisé
ollama create code-assistant -f Modelfile. code-assistant

 # Tester le modèle
ollama run code-assistant « Ecire une fonction de somme de liste des entiers."

# Lister pour confirmer
ollama list
```

API REST Ollama

Ollama expose une API REST JSON sur http://localhost:11434. Endpoints : /api/generate (completion), /api/chat (multi-tours), /api/embeddings, /api/tags, /api/pull, /api/ps.

POST /api/generate

```
curl http://localhost:11434/api/generate \
  -H 'Content-Type: application/json' \
  -d '{
    "model": "llama3.2",
    "prompt": "Explique le RAG en 3 phrases.",
    "stream": false,
    "options": {
      "temperature": 0.7,
      "num_predict": 200
    }
  }'
```

POST /api/chat

```
curl http://localhost:11434/api/chat \
  -H 'Content-Type: application/json' \
  -d '{
    "model": "llama3.2",
    "stream": false,
    "messages": [
      {"role": "system",
       "content": "Tu es un expert Python."},
      {"role": "user",
       "content": "Qu est-ce qu un LLM ?"}
    ]
  }'
```

API REST Ollama : Endpoints

Endpoints de gestion	
`GET /api/tags`	Lister les modèles disponibles
`POST /api/pull`	Télécharger un modèle
`DELETE /api/delete`	Supprimer un modèle
`POST /api/copy`	Copier / renommer un modèle
`GET /api/ps`	Modèles chargés en mémoire
`POST /api/show`	Inspecter un modèle
`POST /api/embeddings`	Calculer un vecteur d'embedding
`GET /`	Statut du serveur

Paramètres	
`temperature`	0.0–2.0 : créativité (défaut 0.8)
`top_k`	Top K tokens (défaut 40)
`top_p`	Nucleus sampling (défaut 0.9)
`num_predict`	Tokens max à générer (-1 = infini)
`num_ctx`	Fenêtre de contexte (défaut 2048)
`repeat_penalty`	Pénalité répétition (défaut 1.1)
`seed`	Reproductibilité (0 = aléatoire)
`stop`	Séquences d'arrêt ex: ["###"]
`num_gpu`	Couches GPU (-1 = auto)
`num_thread`	Threads CPU (défaut : auto)

Intégration Python

pip install ollama

```
import ollama
# Génération simple
r = ollama.generate(
    model='llama3.2',
    prompt='Explique la quantisation.',
    options={'temperature': 0.5}
)
print(r['response'])
# Chat avec historique
msgs = [{'role': 'system',
         'content': 'Tu es un tuteur.'}]
msgs.append({'role':'user','content':'...' })
r = ollama.chat(model='llama3.2',
                messages=msgs)
print(r['message']['content'])
```

Streaming : Génération token par token

Génération avec streaming

```
import ollama

stream = ollama.generate(
    model='llama3.2',
    prompt='Explique l attention du Transformer',
    stream=True
)

print('Reponse : ', end='', flush=True)
for chunk in stream:
    print(chunk['response'],
          end='', flush=True)
print()  # saut de ligne final
```

Chaque chunk = { response: "token", done: false } : done: true en fin de génération.

Calcul d'Embeddings et recherche sémantique

Embeddings & recherche sémantique

```
import ollama, numpy as np
def embed(text, model='nomic-embed-text'):
    r = ollama.embeddings(model=model,
                          prompt=text)
    return r['embedding']
def cosine(v1, v2):
    a,b = np.array(v1), np.array(v2)
    return float(np.dot(a,b) /
           (np.linalg.norm(a)*np.linalg.norm(b)))
docs = ['Ollama tourne des LLMs en local.',
        'Le RAG combine retriever et LLM.']
query = 'Comment executer un LLM ?'
q_emb = embed(query)
scores = [(d, cosine(q_emb, embed(d))) for d in docs]
best = max(scores, key=lambda x: x[1])
print(best[0], round(best[1], 4))
```

Intégration LangChain + Pipeline RAG complet

```
pip install langchain langchain-ollama

from langchain_ollama import ChatOllama,
OllamaEmbeddings
from langchain_core.messages import HumanMessage,
SystemMessage

llm = ChatOllama(model='llama3.2', temperature=0.5)
msgs = [
    SystemMessage(content='Tu es un expert Python.'),
    HumanMessage(content='Explique les decorateurs.')
]
r = llm.invoke(msgs)
print(r.content)

# Embeddings
emb = OllamaEmbeddings(model='nomic-embed-text')
vec = emb.embed_query('Texte a vectoriser')
print(f'Dimension : {len(vec)}')
```

Intégration LangChain + Pipeline RAG complet

```
from langchain_community.vectorstores import Chroma
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough


# Chunking + vector store
splitter = RecursiveCharacterTextSplitter(
    chunk_size=200, chunk_overlap=30)
chunks = splitter.create_documents(documents)
store = Chroma.from_documents(chunks,
    OllamaEmbeddings(model='nomic-embed-text'))
# Pipeline LCEL
prompt = ChatPromptTemplate.from_template(
  'Contexte: {context}\nQuestion: {question}')
chain = ({'context': retriever,
          'question': RunnablePassthrough()}
         | prompt | ChatOllama(model='llama3.2')
         | StrOutputParser())
print(chain.invoke('Qu est-ce qu Ollama ?'))
```