

warning "Mise à jour du backend (octobre 2026)" L'inférence GitHub Models a été retirée le 30 juillet 2026. Les références ci-dessous sont conservées uniquement comme contexte historique pendant la migration de ce lab. Utilisez le Lab 013 ? Backends LLM gratuits pour configurer Foundry Local ou Ollama avant de continuer.

Lab 015 : Ollama ? Exécutez des LLMs localement et gratuitement

Niveau : L100 Parcours : Tous les parcours Durée : ~30 min ? Coût : Gratuit ? S'exécute sur votre machine, pas de cloud, pas de clé API

tip "Essayez aussi Foundry Local" Microsoft Foundry Local est une alternative à Ollama avec une API compatible OpenAI. Consultez le Lab 078 : Foundry Local pour un guide pratique.

Ce que vous allez apprendre

Installer et exécuter Ollama pour servir des LLMs localement Exécuter Phi-4 (le puissant petit modèle de Microsoft) et Llama 3.2 sur votre propre machine Générer des embeddings de texte localement avec nomic-embed-text Appeler Ollama depuis Python et C# en utilisant l'API compatible OpenAI Utiliser Ollama comme backend LLM pour Semantic Kernel (pas de clé API nécessaire)

Introduction

Ollama est un outil open source qui permet d'exécuter des LLMs sur votre ordinateur portable aussi facilement que ollama run phi4. Pas de clé API, pas de compte cloud, pas de coûts d'utilisation ? juste votre propre matériel.

Cela est utile pour : Confidentialité : les données sensibles ne quittent jamais votre machine Développement hors ligne : fonctionne sans internet Contrôle des coûts : zéro frais d'API pendant le développement Apprentissage : expérimentez librement sans vous soucier des factures

info "Configuration matérielle requise" Ollama fonctionne sur Mac (Apple Silicon ou Intel), Windows et Linux. Pour de meilleures performances : 16 Go+ de RAM. Fonctionne avec 8 Go mais plus lentement. Le GPU est optionnel ? les modèles fonctionnent aussi sur CPU (juste plus lentement).

Configuration des prérequis

Installer Ollama

1. Allez sur ollama.com et téléchargez l'installateur pour votre système d'exploitation 2.

Installez et vérifiez :

[diagram or code sample]

Ollama s'exécute comme un service en arrière-plan sur <http://localhost:11434>.

tip "Démarrage rapide avec GitHub Codespaces" Open in GitHub Codespaces

Toutes les dépendances sont pré-installées dans le devcontainer.

? Fichiers de support

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-015/ dans votre répertoire de travail.

Fichier | Description | Téléchargement | Modelfile | Configuration de modèle Ollama | ?

Télécharger | chat_starter.py | Script de démarrage avec des TODOs | ? Télécharger |

Exercice du lab

Étape 1 : Exécuter votre premier modèle

[diagram or code sample]

Cela télécharge Phi-4 (~9 Go) lors de la première exécution, puis démarre un chat interactif.

[diagram or code sample]

Autres modèles à essayer :

[diagram or code sample]

Vérifiez ce que vous avez installé : [diagram or code sample]

Étape 2 : Télécharger un modèle d'embedding

[diagram or code sample]

Cela vous donne un modèle d'embedding local gratuit ? parfait pour le RAG sans aucun coût d'API.

Étape 3 : Appeler Ollama depuis Python

L'API d'Ollama est 100 % compatible OpenAI, donc le même code qui appelle GitHub Models ou Azure OpenAI fonctionne ici :

[diagram or code sample]

Étape 4 : Générer des embeddings localement

[diagram or code sample]

Étape 5 : Utiliser Ollama avec Semantic Kernel

Comme Ollama est compatible OpenAI, l'intégrer dans Semantic Kernel est trivial :

=== "Python"

[diagram or code sample]

=== "C#"

[diagram or code sample]

Étape 6 : Utiliser Ollama comme backend de serveur MCP

Comme Ollama est compatible OpenAI, tout serveur MCP qui appelle un LLM peut l'utiliser localement. Il suffit de changer la configuration du client :

[diagram or code sample]

Aucune autre modification de code nécessaire.

Étape 7 : Ollama via l'API REST directement

Vous pouvez aussi appeler l'API native d'Ollama (non compatible OpenAI) :

[diagram or code sample]

? Fichiers de démarrage

Deux fichiers sont fournis pour vous accompagner :

[diagram or code sample]

Le ? Modelfile crée un persona personnalisé OutdoorGear Advisor basé sur Phi-4. Le ? chat_starter.py contient 5 exercices couvrant la complétion de base, les modèles personnalisés, la comparaison et le streaming.

Comparaison des modèles (sur un ordinateur portable typique)

Modèle | Taille | RAM nécessaire | Vitesse | Qualité | phi4-mini | 2,5 Go | 4 Go | ???

Rapide | Bonne | llama3.2:1b | 1,3 Go | 4 Go | ??? Très rapide | Basique | llama3.2 | 2,0

Go | 6 Go | ?? Rapide | Bonne | phi4 | 9,1 Go | 12 Go | ? Modérée | Excellente | mistral |

4,1 Go | 8 Go | ?? Rapide | Très bonne | deepseek-r1 | 4,7 Go | 8 Go | ? Modérée | Meilleur raisonnement |

Résumé

Vous disposez maintenant d'une pile LLM entièrement locale :

? Ollama servant les modèles sur localhost:11434 ? Phi-4 (ou Llama) pour le

chat/raisonnement ? gratuit, privé, hors ligne ? nomic-embed-text pour les embeddings ?

gratuit, local ? Le même code fonctionne pour Ollama, GitHub Models et Azure OpenAI ? il suffit de changer l'URL de base

Prochaines étapes

Construire une application RAG avec des embeddings locaux : ? Lab 022 ? RAG avec GitHub

Models + pgvector Utiliser avec les plugins Semantic Kernel : ? Lab 023 ? Plugins, mémoire

et planificateurs SK IA locale en production : ? Lab 044 ? Phi-4 + Ollama en production

Kit d'accompagnement

Télécharger le PDF généré Télécharger le ZIP généré

```
def
compare_local_models(models: list[str], question: str): """Compare responses from multiple
local models.""" print(f"\n? Question: {question}\n")
```

```
for model in models: print(f"?*50)") print(f"Model: {model}") print('*50') try: reply = ask_ollama(question, model=model) print(reply) except Exception as e: print(f"? Model '{model}' not available: {e}") print(f" Pull it with: ollama pull {model}") print()
???????????????????????????????????????????????????????????????????????????????? Exercise 5: Streaming responses (watch tokens appear in real-time)
???????????????????????????????????????????????????????????????????????????????? def streaming_chat(prompt: str, model: str = "phi4"): """Stream the response token by token.""" print(f"\n? Streaming from {model}:\n") stream = ollama_client.chat.completions.create( model=model, messages=[{"role": "user", "content": prompt}], stream=True, ) for chunk in stream: delta = chunk.choices[0].delta.content if delta: print(delta, end="", flush=True) print("\n") # newline after streaming
???????????????????????????????????????????????????????????????????????????????? Main
???????????????????????????????????????????????????????????????????????????????? if __name__ == "__main__": # Step 1: Check what's available available = list_local_models() if not available: print("\n? Quick start:") print(" 1. Install Ollama: https://ollama.com") print(" 2. Run: ollama pull phi4") print(" 3. Run this script again") else: # Step 2: Simple question print("\n=== Exercise 2: Simple Completion ===") reply = ask_ollama("What's the most important piece of gear for a first-time hiker?", model=available[0] ) print(reply) # Step 3: Streaming print("\n=== Exercise 5: Streaming ===") streaming_chat( "Explain the layering system for outdoor clothing in 3 steps.", model=available[0] ) # Step 4: Custom OutdoorGear model (requires Modelfile setup) print("\n=== Exercise 3: Custom OutdoorGear Model ===") print("Create the model first: ollama create outdoorgear -f Modelfile") # Uncomment to run: chat_with_outdoorgear() # Step 5: Compare models (if you have multiple) if len(available) >= 2: print("\n=== Exercise 4: Compare Models ===") compare_local_models( available[:2], "What sleeping bag temperature rating do I need for 0°C camping?" )
```