

warning "Atualização de backend (outubro de 2026)" A inferência do GitHub Models foi descontinuada em 30 de julho de 2026. As referências abaixo são mantidas apenas como contexto histórico enquanto este lab é migrado. Use o Lab 013 ? Backends de LLM gratuitos para configurar o Foundry Local ou o Ollama antes de continuar.

Lab 015: Ollama ? Execute LLMs Localmente de Graça

Nível: L100 Caminho: Todos os caminhos Tempo: ~30 min ? Custo: Gratuito ? Roda na sua máquina, sem nuvem, sem chave de API

tip "Experimente também o Foundry Local" O Microsoft Foundry Local é uma alternativa ao Ollama com uma API compatível com OpenAI. Veja o Lab 078: Foundry Local para um guia prático.

O Que Você Vai Aprender

Instalar e executar o Ollama para servir LLMs localmente Executar o Phi-4 (poderoso modelo pequeno da Microsoft) e o Llama 3.2 na sua própria máquina Gerar embeddings de texto localmente com nomic-embed-text Chamar o Ollama a partir de Python e C# usando a API compatível com OpenAI Usar o Ollama como backend de LLM para o Semantic Kernel (sem necessidade de chave de API)

Introdução

O Ollama é uma ferramenta de código aberto que torna a execução de LLMs no seu laptop tão fácil quanto `ollama run phi4`. Sem chave de API, sem conta na nuvem, sem custos de uso ? apenas o seu próprio hardware.

Isso é valioso para: Privacidade: dados sensíveis nunca saem da sua máquina Desenvolvimento

offline: funciona sem internet Controle de custos: zero custos de API durante o

desenvolvimento Aprendizado: experimente livremente sem se preocupar com cobranças

info "Requisitos de hardware" O Ollama funciona em Mac (Apple Silicon ou Intel), Windows e Linux. Para melhor desempenho: 16GB+ de RAM. Funciona com 8GB, mas mais lento. GPU é opcional ? os modelos rodam na CPU também (apenas mais lento).

Configuração de Pré-requisitos

Instalar o Ollama

1. Acesse ollama.com e baixe o instalador para o seu sistema operacional 2. Instale e verifique:

[diagram or code sample]

O Ollama roda como um serviço em segundo plano em `http://localhost:11434`.

tip "Início Rápido com GitHub Codespaces" Open in GitHub Codespaces

Todas as dependências já estão pré-instaladas no devcontainer.

? Arquivos de Apoio

note "Baixe estes arquivos antes de iniciar o laboratório" Salve todos os arquivos em uma pasta `lab-015/` no seu diretório de trabalho.

Arquivo | Descrição | Download | Modelfile | Configuração de modelo do Ollama | ? Download | `chat_starter.py` | Script inicial com TODOs | ? Download |

Exercício do Laboratório

Passo 1: Execute seu primeiro modelo

[diagram or code sample]

Isso baixa o Phi-4 (~9GB) na primeira execução e depois inicia um chat interativo.

[diagram or code sample]

Outros modelos para experimentar:

[diagram or code sample]

Verifique o que você tem instalado: [diagram or code sample]

Passo 2: Baixe um modelo de embedding

[diagram or code sample]

Isso fornece um modelo de embedding local gratuito ? perfeito para RAG sem nenhum custo de API.

Passo 3: Chame o Ollama a partir do Python

A API do Ollama é 100% compatível com OpenAI, então o mesmo código que chama o GitHub Models ou Azure OpenAI funciona aqui:

[diagram or code sample]

Passo 4: Gere embeddings localmente

[diagram or code sample]

Passo 5: Use o Ollama com o Semantic Kernel

Como o Ollama é compatível com OpenAI, integrá-lo ao Semantic Kernel é trivial:

```
=== "Python"
```

[diagram or code sample]

```
=== "C#"
```

[diagram or code sample]

Passo 6: Use o Ollama como backend de um servidor MCP

Como o Ollama é compatível com OpenAI, qualquer servidor MCP que chame um LLM pode usá-lo localmente. Basta trocar a configuração do cliente:

[diagram or code sample]

Nenhuma outra alteração de código é necessária.

Passo 7: Ollama via API REST diretamente

Você também pode chamar a API nativa do Ollama (não compatível com OpenAI):

[diagram or code sample]

? Arquivos Iniciais

Dois arquivos são fornecidos para ajudá-lo a acompanhar:

[diagram or code sample]

O ? Modelfile cria uma persona personalizada de Consultor de Equipamentos Outdoor em cima do Phi-4. O ? chat_starter.py tem 5 exercícios cobrindo completção básica, modelos personalizados, comparação e streaming.

Comparação de Modelos (em um laptop típico)

Modelo	Tamanho	RAM necessária	Velocidade	Qualidade
phi4-mini	2.5GB	4GB	???	
llama3.2:1b	1.3GB	4GB	???	Muito rápido
Básico	llama3.2	2.0GB	6GB	??
Rápido	Bom	phi4	9.1GB	12GB
? Moderado	Excelente	mistral	4.1GB	8GB
?? Rápido	Muito bom	deepseek-r1	4.7GB	8GB
? Moderado	Melhor raciocínio			

Resumo

Agora você tem uma stack de LLM totalmente local:

? Ollama servindo modelos em localhost:11434 ? Phi-4 (ou Llama) para chat/raciocínio ?

gratuito, privado, offline ? nomic-embed-text para embeddings ? gratuito, local ? O mesmo

código funciona para Ollama, GitHub Models e Azure OpenAI ? basta mudar a URL base

Próximos Passos

Construa um app RAG com embeddings locais: ? Lab 022 ? RAG com GitHub Models + pgvector Use

com plugins do Semantic Kernel: ? Lab 023 ? Plugins, Memória e Planejadores do SK IA local

em produção: ? Lab 044 ? Phi-4 + Ollama em Produção

Kit de apoio

O build do site gera downloads estáveis em PDF e ZIP. O ZIP contém o guia do lab e todos os arquivos de apoio declarados para este lab.

Baixar PDF gerado Baixar ZIP gerado

```

""" Lab 015: Ollama ? Local LLMs ===== Starter file ? chat with
Ollama running locally. 100% free, no API keys, no internet required after model download.
Prerequisites: 1. Install Ollama: https://ollama.com 2. Pull a model: ollama pull phi4 3.
pip install openai (Ollama uses OpenAI-compatible API)
No environment variables needed ? Ollama runs at http://localhost:11434 """
from openai import OpenAI

```

[illegible][illegible]

Exercise 3:
Chat with the custom OutdoorGear model

??? Exercise 4:
Compare two local models side-by-side

[illegible]

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?" )
```