

Lab 013: Free LLM Backends ? Foundry Local and Ollama

Level: L100 Journey: LLM Foundations Time: ~45 min ? Cost: Free ? local execution, no account or credit card

info "Why this lab changed" GitHub Models inference was retired on July 30, 2026. This lab now provides the shared free backend used by the hub: Foundry Local on supported Windows devices or Ollama on Windows, macOS, and Linux.

What You'll Learn

Choose between Foundry Local and Ollama Start an OpenAI-compatible local endpoint Call a local chat model from Python Configure the environment variables used by other labs Verify that no prompts or data leave your machine

Choose a Runtime

=== "Foundry Local" Best for supported Windows devices and DirectML acceleration.

[diagram or code sample]

=== "Ollama" Cross-platform and easy to use on Windows, macOS, and Linux.

1. Install Ollama. 2. Start a small model:

[diagram or code sample]

Configure the shared endpoint:

=== "PowerShell" [diagram or code sample]

=== "macOS / Linux" [diagram or code sample]

Supporting Files

File | Description | Download | requirements.txt | Python dependency | Download | starter.py | Local-backend exercises | Download |

Exercise 1 ? Verify the Endpoint

Install the client and run the starter:

[diagram or code sample]

The OpenAI SDK accepts both runtimes because they expose an OpenAI-compatible API:

[diagram or code sample]

Exercise 2 ? Confirm Local Processing

1. Disconnect from the internet after the model download completes. 2. Run python starter.py again. 3. Confirm the response still succeeds. 4. Stop the runtime and confirm the script reports a connection error rather than silently switching to a cloud service.

Shared Configuration for Other Labs

The migrated labs use these variables:

Variable | Example | Purpose | LOCAL_LLM_BASE_URL | http://localhost:11434/v1 | OpenAI-compatible endpoint | LOCAL_LLM_MODEL | phi4-mini | Installed chat model | LOCAL_EMBEDDING_MODEL | Runtime-specific | Optional embedding model | warning "Model capabilities vary" Tool calling, structured output, embeddings, and context limits depend on the selected model and runtime. Each lab lists the capabilities it requires.

Knowledge Check

question "Why do the labs use environment variables instead of hardcoding the endpoint?"

They let the same code work with Foundry Local, Ollama, or a documented paid backend without rewriting the exercise.

question "Does the OpenAI SDK mean the request is sent to OpenAI?" No. The SDK is only the client library. base_url determines where the request is sent.

Next Steps

Lab 015 ? Ollama: Local LLMs Lab 078 ? Foundry Local Lab 017 ? Structured Output

Supporting kit

Download generated PDF Download generated ZIP

Supporting file: requirements.txt
openai>=1.0.0

Supporting file: starter.py

""Lab 013 starter for Foundry Local or Ollama.""

```
from openai import OpenAI import os
```

```
BASE_URL = os.environ.get("LOCAL_LLM_BASE_URL", "http://localhost:11434/v1") MODEL =
```

```
os.environ.get("LOCAL_LLM_MODEL", "phi4-mini") client = OpenAI(base_url=BASE_URL,
```

```
api_key="local")
```

[illegible][illegible]

```
def simple_chat(user_message: str) -> str: """Send a single message and return the
```

```
response.text.""" response = client.chat.completions.create( model=MODEL,
```

```
messages=[{"role": "user", "content": user_message}], ) return
```

```
response.choices[0].message.content or ""
```

[illegible]

Chat with a system prompt (OutdoorGear assistant)

[illegible]

= ""You are an expert outdoor gear advisor for OutdoorGear Inc. You help customers choose

the right gear for their adventures. Keep answers concise (max 3 sentences). Always ask one

follow-up question. Never recommend products you don't know exist."'''

```
def outdoor_advisor(user_question: str) -> str: """Answer outdoor gear questions using the
```

```
system prompt above.""" response = client.chat.completions.create( model=MODEL, messages=[
```

```

{"role": "system", "content": SYSTEM_PROMPT}, {"role": "user", "content": user_question},

```

```
], ) return response.choices[0].message.content or ""
```

[illegible][illegible]

```
????????? def multi_turn_chat(): """A simple CLI loop that maintains conversation
```

```
history[""] messages = [{"role": "system", "content": SYSTEM_PROMPT}] print("OutdoorGear
```

Advisor (type 'quit' to exit)\n"

```
while True: user_input = input("You: ").strip() if user_input.lower() in ("quit", "exit",
```

```
"q"): break
```

```
# TODO 4: Append the user message to messages # TODO 5: Call the API with the full messages
```

```
list # TODO 6: Extract the assistant reply and print it # TODO 7: Append the assistant
```

```
reply to messages for next turn print("TODO: implement multi turn chat()") break # Remove
```

this line once implemented

[illegible][illegible]

?????? MODELS TO TRY = [MODEL]

QUESTION = "What sleeping bag temperature rating do I need for winter camping at -10°C?"

```
def compare_models(): """Send the same question to multiple models and compare
```

```
responses.""" for model_name in MODELS_TO_TRY: print(f'\n{'='*60}') print(f'Model:
```

```
{model_name}") print('='*60) # TODO 8: Call the API with each model_name and print the
```

```
response # Handle errors gracefully (some models may not be available) print(f"TODO: call
```

```
{model_name}")
???????????????????????????????????????????????????????????????????????????????? Main
???????????????????????????????????????????????????????????????????????????????? if __name__
== "__main__": print("=== Exercise 1: Simple Chat ===") # Uncomment after completing TODO
2: # print(simple_chat("What's a good tent for solo hiking?"))
print("\n=== Exercise 2: Outdoor Advisor ===") # Uncomment after completing TODO 3: #
print(outdoor_advisor("I need a sleeping bag for -5°C winter camping."))
print("\n=== Exercise 3: Multi-turn Chat ===") # Uncomment after completing TODOs 4-7: #
multi_turn_chat()
print("\n=== Exercise 4: Compare Models ===") # Uncomment after completing TODO 8: #
compare_models()
print("\n? Complete the TODOs above and uncomment the exercises!")
```