LLM Engineering: Master AI, Large Language Models & Agents

Day 2: Frontier Models & Running LLMs Locally with Ollama

Agenda

1. Path to LLM Engineering
2. Introduction to Frontier Models
3. Comparing Closed-Source vs Open-Source LLMs
4. Exploring Ollama Internals
5. Setting Up Ollama (Recap + Advanced Notes)
6. Using Ollama in Python (API & SDK)
7. Building a Summarizer with Ollama
8. Advantages & Limitations
9. Technical Considerations
10. Final Notes

1 Path to LLM Engineering

Focusing on learning by doing, this course blends theory with practical application. LLM Engineers master:

- Model architecture and landscape (open vs closed source)
- Tools like Hugging Face, LangChain, Gradio, Weights & Biases, Modal
- Application techniques: Retrieval-Augmented Generation (RAG), fine-tuning, Agentic AI

2 Introduction to Frontier Models

Frontier models are the most powerful LLMs, pushing AI boundaries for enterprise use. Examples include:

- GPT-4 / GPT-4o (OpenAI)
- Claude 3 (Anthropic)
- Gemini (Google)
- Command-R (Cohere)
- Perplexity AI (chat/search hybrid)

These are typically closed-source, paid, and cloud-hosted.

3 Comparing Closed vs Open Source LLMs

Feature	**Closed Source (GPT, Claude)**	**Open Source (LLaMA, Mistral)**
Accuracy	High	Moderate
Cost	Pay-per-use	Free
Privacy	Cloud-hosted	Fully local
Setup Complexity	Minimal	Moderate/High
Customization	Limited	High

4 Exploring Ollama Internals

Ollama runs open-source LLMs locally using C++ inference via llama.cpp:

- Supports LLaMA, Mistral, Phi, and more
- Works on macOS, Windows, Linux
- Keeps data on your device
- Serves local REST API at localhost:11434

Ideal for prototyping, avoiding API charges, and handling confidential data.

5 Setting Up Ollama (Recap)

Install Ollama:

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

Pull a model:

```
ollama pull llama3
```

Run Ollama server:

```
ollama serve
```

Check if it's running: Visit http://localhost:11434 in your browser (should show "Ollama is running").

6 Using Ollama in Python (Two Methods)

Method 1: Manual REST API

```
import requests

messages = [{"role": "user", "content": "Hello Ollama!"}]
response = requests.post("http://localhost:11434/api/chat",
                         json={"model": "llama3", "messages
                            ": messages})
print(response.json()["message"]["content"])
```

Method 2: Using ollama Python SDK

```
import ollama

response = ollama.chat(model="llama3", messages=[
    {"role": "user", "content": "What are the applications
       of LLMs in business?"}
])
print(response['message']['content'])
```

7 Building a Summarizer with Ollama

Upgrades the Day 1 tool to use a local Ollama model.

7.1 Install dependencies

```
pip install requests beautifulsoup4 ollama
```

7.2 Summarization Code

```
import requests
from bs4 import BeautifulSoup
import ollama
from IPython.display import Markdown, display

MODEL = "llama3"

class Website:
    def __init__(self, url):
        self.url = url
        headers = {"User-Agent": "Mozilla/5.0"}
        response = requests.get(url, headers=headers)
        soup = BeautifulSoup(response.content, 'html.parser
            ')
        self.title = soup.title.string if soup.title else "
            No title found"
        for tag in soup(["script", "style", "img", "input"
            ]):
            tag.decompose()
        self.text = soup.body.get_text(separator="\n",
            strip=True)

def user_prompt_for(website):
    return (
```

```
        f"You are looking at a website titled {website.
            title}.\n"
        f"The contents of this website is as follows:\n"
        f"{website.text[:2000]}\n\n"
        "Please provide a concise summary in markdown. "
        "Ignore navigation menus, ads, and boilerplate."
    )

def summarize(url):
    site = Website(url)
    messages = [
        {"role": "system", "content": "You are an assistant
            that summarizes websites."},
        {"role": "user", "content": user_prompt_for(site)}
    ]
    response = ollama.chat(model=MODEL, messages=messages)
    return response['message']['content']

# Try it!
summary = summarize("https://edwarddonner.com")
display(Markdown(summary))
```

7.3 Sample Output

```
# Summary of edwarddonner.com

A personal portfolio for Ed Donner, showcasing AI education
    and LLM engineering. Highlights include an 8-week Udemy
    course focused on Generative AI and practical real-
   world projects.
```

8 Advantages & Limitations

Advantages

- No API charges
- Runs offline
- No data leaves your machine
- Great for confidential use cases

Limitations

- Lower model performance vs GPT-4
- Heavier system requirements
- Won't work for JavaScript-heavy websites
- Requires setup (Ollama, model pulls, local runtime)

9 Technical Considerations

- Content limits: Truncate or chunk long pages (e.g., first 2000 chars)
- Selenium for JS: Use selenium instead of requests for dynamic sites
- Error handling: Add retry logic for failed requests
- GPU/CPU: Models like LLaMA 3.2 may be slow on CPU

10 Final Notes

You now understand:

- Frontier models vs open-source
- Ollama internals and setup
- Calling models via REST and Python

- Building summarizers with local LLMs

Next up: benchmarking six frontier models (GPT-4, Claude, Gemini, and others) with real-world prompts. Stay tuned for Day 3!