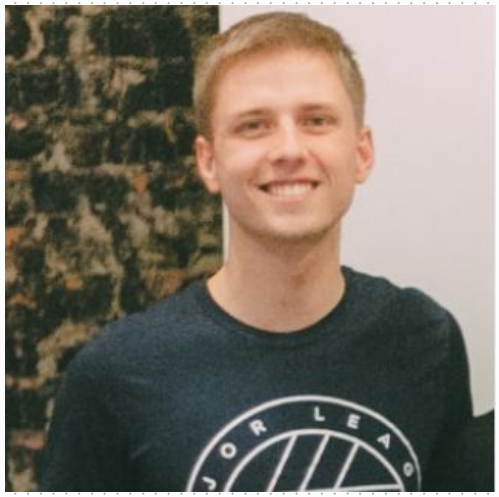
AI Agents with Low-Cost LLMs

GHW May 2025



@MLHacks

@MLHacks

Hey! I'm **Stephen**.

- Coach at Major League Hacking (~5 years).
- Super excited for GHW Open Source!

Overview

- We will learn to build agentic AI applications using CrewAI and Ollama
- Focus is on cost-effective implementation with local LLMs.
- The tutorial is split into two 2-hour sessions
- Some theoretical knowledge…
- Some hands-on experience…

Let’s get started!

Prerequisites

- Python 3.9+
- Git
- Basic understanding of Python programming
- Basic understanding of AI/ML concepts

Quick Start

Clone this repository:

```bash
git clone https://github.com/croppers/crewai
cd crewai
```

Create and activate a virtual environment

```bash
python -m venv venv
source venv/bin/activate  # On Windows:
venv\Scripts\activate
```

Install dependencies

```bash
pip install -r requirements.txt
```

Install Ollama

- Visit ollama.com
- Download and install for your operating system
- Pull a base model:

```bash
ollama pull mistral
```

Part 1: Foundations & Basic Implementation

Part 1: Foundations & Basic Implementation

- Environment Setup
- CrewAI Fundamentals
- Ollama Deep Dive
- Building Your First Agent
- Multi-Agent Basics

Welcome & Overview

Introduction to CrewAI and Ollama

CrewAI: A framework for building agentic AI applications

- Agent-based architecture
- Collaborative AI systems
- Tool integration capabilities
- Process management

Introduction to CrewAI and Ollama

Ollama: Local LLM deployment

- Open-source model hosting
- Cost-effective inference
- Model management
- Performance optimization

Why Local LLMs Matter

- Cost reduction
- Data privacy
- Latency improvement
- Customization potential
- Offline capabilities

Cost Comparison Overview

- Cloud LLM costs (GPT-4, Claude, etc.)
- Local LLM costs (Ollama)
- Infrastructure requirements
- Total cost of ownership

Environment Setup

Installing Ollama

Download and Installation:

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

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

# Windows
# Download from https://ollama.com/download
```

https://ollama.com/download

Basic Model Testing

```bash
# Pull a base model
ollama pull gemma:2b

# Test the model
ollama run gemma:2b "Hello, how are you?"
```

What is gemma:2b?

Model Management

```bash
# List available models
ollama list

# Remove a model
ollama rm gemma:2b

# Pull specific model version
ollama pull gemma:2b
```

Setting up CrewAI

Python Environment Setup

```bash
# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

Basic Configuration

Create **.env** file:

```env
OLLAMA_BASE_URL=http://localhost:11434
DEFAULT_MODEL=gemma:2b
```

CrewAI Fundamentals

Agent Architecture

hello_world.py

Crew Concepts

first_crew.py

Ollama Deep Dive

Available Models

- Gemma (2B, 7B)
- Llama2 (7B, 13B, 70B)
- Mistral
- Qwen
- and more…

Model Selection Criteria

- Task requirements
- Hardware constraints
- Performance needs
- Cost considerations

Performance Considerations

- Memory usage
- Inference speed
- Quality of outputs
- Resource utilization

Cost Implications

- Hardware requirements
- Electricity costs
- Maintenance overhead
- Scaling considerations

Building Your First Agent

Single Agent Implementation

```
from crewai import Agent, Task
from langchain_community.llms import Ollama
from langchain.tools import Tool
# Custom tool example
def search_web(query: str) -> str:
    # Implement web search functionality
    return f"Search results for: {query}"
# Create custom tools
tools = [
    Tool(
        name="WebSearch",
        func=search_web,
        description="Search the web for information"
    )
]
```

Single Agent Implementation

```
# Create agent with custom tools
agent = Agent(
    role='Research Analyst',
    goal='Analyze and research topics thoroughly',
    backstory='Expert analyst with strong research skills',
    llm=Ollama(model="ollama/gemma:2b"),
    tools=tools,
    verbose=True
)
# Execute task
task = Task(
    description="Research the latest developments in quantum computing",
    agent=agent
)
result = agent.execute(task)
```

Multi-Agent Basics

Introduction to Crews

- Crew architecture
- Agent communication
- Task delegation
- Collaboration patterns

Building a Simple Crew

```
from crewai import Agent, Crew, Task
from langchain_community.llms import Ollama
# Initialize agents
researcher = Agent(
    role='Researcher',
    goal='Research topics thoroughly',
    backstory='Expert researcher',
    llm=Ollama(model="ollama/gemma:2b")
)
analyst = Agent(
    role='Analyst',
    goal='Analyze research findings',
    backstory='Data analyst with strong analytical skills',
    llm=Ollama(model="ollama/gemma:2b")
)
writer = Agent(
    role='Writer',
    goal='Create engaging content',
    backstory='Experienced content writer',
    llm=Ollama(model="ollama/gemma:2b")
)
```

Building a Simple Crew

```
# Create tasks
research_task = Task(
    description="Research AI in healthcare",
    agent=researcher
)
analysis_task = Task(
    description="Analyze the research findings",
    agent=analyst
)
writing_task = Task(
    description="Write a comprehensive report",
    agent=writer
)
# Create and run crew
crew = Crew(
    agents=[researcher, analyst, writer],
    tasks=[research_task, analysis_task, writing_task],
    verbose=True
)
result = crew.kickoff()
```

Part 2: Advanced Implementation & Real-World Applications

Part 2: Advanced Implementation & Real-World Applications

- Advanced Agent Development
- Building Complex Crews
- Real-World Application Development
- Production & Beyond

Part 1: Custom Tools Development

```python
from typing import List, Dict, Any
from langchain.tools import BaseTool
from pydantic import BaseModel, Field
import requests
from bs4 import BeautifulSoup

class WebScraperTool(BaseTool):
    name = "web_scraper"
    description = "Scrape content from a website"

    class InputSchema(BaseModel):
        url: str = Field(..., description="URL to scrape")
        selector: str = Field(..., description="CSS selector for content")

    def _run(self, url: str, selector: str) -> str:
        try:
            response = requests.get(url)
            soup = BeautifulSoup(response.text, 'html.parser')
            content = soup.select(selector)
            return "\n".join([elem.text for elem in content])
        except Exception as e:
            return f"Error scraping website: {str(e)}"
```

Part 1: Custom Tools Development

```python
class DataAnalysisTool(BaseTool):
    name = "data_analyzer"
    description = "Analyze data and generate insights"

    class InputSchema(BaseModel):
        data: List[Dict[str, Any]] = Field(..., description="Data to analyze")
        metrics: List[str] = Field(..., description="Metrics to calculate")

    def _run(self, data: List[Dict[str, Any]], metrics: List[str]) -> str:
        # Implement data analysis logic
        return "Analysis results..."

# Create advanced agent with custom tools
from crewai import Agent
from langchain_community.llms import Ollama

advanced_agent = Agent(
    role='Data Analyst',
    goal='Analyze data and generate insights',
    backstory='Expert data analyst with strong analytical skills',
    llm=Ollama(model="mistral"),
    tools=[WebScraperTool(), DataAnalysisTool()],
    verbose=True
)
```

Part 2: Advanced Agent Patterns

```python
from crewai import Agent, Task, Crew
from langchain_community.llms import Ollama
from typing import List, Dict
import asyncio

class HierarchicalAgent(Agent):
    def __init__(self, sub_agents: List[Agent], **kwargs):
        super().__init__(**kwargs)
        self.sub_agents = sub_agents

    async def delegate_task(self, task: Task) -> str:
        # Implement task delegation logic
        results = []
        for agent in self.sub_agents:
            result = await agent.execute(task)
            results.append(result)
        return self.aggregate_results(results)

    def aggregate_results(self, results: List[str]) -> str:
        # Implement result aggregation logic
        return "\n".join(results)
```

Part 2: Advanced Agent Patterns

```
# Create hierarchical agent system
researcher = Agent(
    role='Researcher',
    goal='Research topics thoroughly',
    llm=Ollama(model="mistral")
)

analyst = Agent(
    role='Analyst',
    goal='Analyze research findings',
    llm=Ollama(model="mistral")
)

writer = Agent(
    role='Writer',
    goal='Create engaging content',
    llm=Ollama(model="mistral")
)

team_lead = HierarchicalAgent(
    role='Team Lead',
    goal='Coordinate team efforts',
    backstory='Experienced team leader',
    llm=Ollama(model="mistral"),
    sub_agents=[researcher, analyst, writer]
)
```

Part 3: Building Complex Crews

complex_crew.py

Challenge: Build our own AI Agent with this model!

We will use codeshare.io to share

Next time…

- Production environment!
- More use cases
- More models
- Fun!