Lecture 15: Building Your Own Agentic Framework

🎯 Learning Objectives

By the end of this lecture, you should be able to:

- Understand the core architectural components of an agentic framework.
- Design a modular and extensible agent framework from scratch.
- Implement a reusable loop for reasoning, memory, and tool use.
- Build a lightweight, pluggable system tailored to your use case.

🧩 Key Concepts

Why Build Your Own Framework?

- Greater control over execution, memory, and integration.
- Tailored to domain-specific constraints (e.g., latency, safety).
- Avoids over-engineering or dependency overload.

Core Building Blocks

- **Controller**: Orchestrates the reasoning-action loop.
- **Reasoner**: Generates thoughts, actions, and decisions using the LLM.
- **Toolset**: Modular tools callable by the agent.
- **Memory**: Stores relevant history or retrieved knowledge.
- **Goal Tracker**: Optional state machine to monitor progress.

🛠️ Required Tools/Libraries

- Python
- OpenAI / Hugging Face API
- Optional: FAISS or Chroma for memory, JSON schema for tool definition

---

🔬 Hands-on Exercise: Build a Mini Agent Framework

Goal: Create a simple Python-based agent framework with reasoning, tool use, and memory support.

Step 1: Define tool functions

```
tools = {
    "search": lambda q: f"[Search result for '{q}']",
    "math": lambda expr: str(eval(expr))
}
```

Step 2: Define LLM reasoner function

```
def reasoner(prompt):
    # Placeholder for actual LLM call
    print("LLM receives prompt:\n", prompt)
    return input("LLM response: ")
```

Step 3: Implement the agent loop

```
memory = []
goal = "Find the square root of 144 and explain it."

while True:
    context = "\n".join(memory[-3:])
    prompt = f"Goal: {goal}\nContext:\n{context}\nWhat should I do next?"
    response = reasoner(prompt)

    memory.append(response)

    if "Final Answer:" in response:
        break
```

Step 4: Review and extend

- Add regex to extract tool calls.
- Log each step for transparency.
- Modularize components into classes: Agent, Tool, Memory, GoalTracker.

Bonus:

- Add a configuration file (e.g., YAML) to define available tools and agent personality.
- Support multiple agents working collaboratively.
- Integrate vector memory for semantic recall.
