Lecture 17: Safety, Hallucination & Guardrails

🎯 Learning Objectives

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

- Understand common safety risks in agentic AI systems.
- Identify causes and examples of hallucination in LLM-based agents.
- Apply techniques to reduce misinformation and unsafe behavior.
- Use guardrail frameworks to constrain agent output and tool use.

🧩 Key Concepts

What Is Hallucination?

- When an LLM **confidently generates false or unsupported information**.
- Examples:
  - Fabricated citations or URLs
  - Incorrect math or logic
  - Misleading tool outputs

Safety Challenges in Agentic Systems

- **Autonomy + Tools** = Amplified risk
- Possibilities include:
  - Accessing unsafe tools
  - Creating toxic or biased output
  - Looping on bad reasoning
  - Leaking sensitive information

Guardrail Strategies

- **Prompt-level**: Add refusals, constraints, or system messages.
- **Output filtering**: Use regex, keyword blacklists, or content classifiers.
- **Function boundaries**: Restrict which tools an agent can call.
- **Validation hooks**: Confirm actions before execution.
- **Human-in-the-loop**: Manual review before execution or deployment.

🛠️ Required Tools/Libraries

- OpenAI API (or other LLM)
- Python
- Optional: Guardrails.ai, Rebuff, LangChain `OutputParser`

```
pip install guardrails-ai rebuff openai
```

🔬 Hands-on Exercise: Add Guardrails to an Agent

Goal: Modify an existing agent to avoid hallucinations and unsafe tool use.

Step 1: Add a system prompt

```
system_prompt = """
You are a safe and helpful assistant.
Never fabricate information. If unsure, say "I don't know".
Only use tools that have been explicitly approved.
"""
```

Step 2: Restrict tool usage

```
approved_tools = ["calculator", "search"]
if tool_name not in approved_tools:
    raise Exception("Unauthorized tool")
```

Step 3: Validate final output

```
def validate_output(response):
    if "Final Answer:" not in response:
        return "Response incomplete."
    if any(bad_word in response.lower() for bad_word in ["kill", "hack", "fake"]):
        return "Unsafe content detected."
    return "OK"

result = validate_output(agent_response)
```

Bonus:

- Use Rebuff to add adversarial test cases against the agent.
- Build a red-teaming harness to probe for vulnerabilities.
- Add a logging layer for all agent outputs with safety scores.
