LLM Agents & Tool Use

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

From Chatbots to Agents

Chatbot: Passive. Responds to text with text.

- User: "What is the weather?"
- Bot: "I cannot browse the internet."

Agent: Active. Can "do" things using tools.

- User: "What is the weather?"
- Agent: *Calls Weather API* → *Gets Data* → "It is 25°C in Gandhinagar."

Core Loop:

1. **Reason**: What do I need to do?
2. **Act**: Call a function/tool.
3. **Observe**: Read the output.
4. **Repeat** until answer found.

ReAct Pattern

Reason + Act.

Prompting strategy to force the LLM to think before acting.

```
Question: What is the elevation range of the area that the eastern sector of the Colorado orogeny extends into?

Thought 1: I need to search for Colorado orogeny, find the area that the eastern sector extends into, then find the elevation range of the area.
Action 1: Search[Colorado orogeny]
Observation 1: The Colorado orogeny was an episode of mountain building ...

Thought 2: It does not mention the eastern sector. So I need to look up eastern sector.
...
```

Function Calling (Tool Use)

Modern LLMs (GPT-4, Gemini, Claude) are trained to output structured JSON calls.

Schema:

```
{
  "name": "get_weather",
  "description": "Get current weather",
  "parameters": {
    "type": "object",
    "properties": {
      "location": {"type": "string"},
      "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
    }
  }
}
```

LLM Output:

`{"function": "get_weather", "args": {"location": "Ahmedabad", "unit": "celsius"}}`

LangGraph: Graph-based Agents

LangChain's Agent library.
Define agents as nodes in a state graph.

Concepts:

- **State**: Shared memory (messages, variables).
- **Nodes**: Functions that modify state (e.g., "Agent", "Tool").
- **Edges**: Control flow (Conditional logic).

```
from langgraph.graph import StateGraph, END

workflow = StateGraph(AgentState)

workflow.add_node("agent", call_model)
workflow.add_node("tools", tool_executor)

workflow.set_entry_point("agent")
workflow.add_conditional_edges("agent", should_continue, {"continue": "tools", "end": END})
workflow.add_edge("tools", "agent")
```

Multi-Agent Systems

Single Agents get confused with complex tasks.
Multi-Agent: Specialized roles.

Example: **Software Dev Team**

1. **Product Manager**: Breaks down requirements.
2. **Coder**: Writes python code.
3. **Reviewer**: Checks code for bugs.
4. **Tester**: Writes tests.

Frameworks:

- **AutoGen** (Microsoft): Conversational agents.
- **CrewAI**: Role-playing agents.

AutoGen Example

```
from autogen import AssistantAgent, UserProxyAgent

assistant = AssistantAgent("assistant", llm_config= ... )
user_proxy = UserProxyAgent("user_proxy", code_execution_config={"work_dir": "coding"})

# Start conversation
user_proxy.initiate_chat(
    assistant,
    message="Plot a chart of NVDA and TESLA stock price change YTD."
)
```

What happens?

1. Assistant writes Python code to fetch data.
2. User Proxy executes code locally.
3. User Proxy shares errors/output.
4. Assistant fixes code.

Lab: Build a Research Agent

Objective: Build an agent that can browse the web and summarize topics.

Tools:

- **Tavily API**: Search engine optimized for LLMs.
- **LangChain / LangGraph**: Orchestration.
- **Gemini / OpenAI**: Brain.

Task:

- User: "Research the latest breakthroughs in Solid State Batteries from 2024."
- Agent:
  - Search Query 1...
  - Read Page...
  - Search Query 2...
  - Summarize Findings.

Risks & Challenges

1. **Infinite Loops**: Agent keeps trying failing action.
2. **Cost**: Agents can burn tokens fast.
3. **Security**:
    - "Prompt Injection" can hijack agents.
    - Don't give agents `rm -rf /` capabilities!
    - Always have "Human in the loop" for critical actions.

Resources

- **LangGraph Docs**: python.langchain.com/docs/langgraph
- **AutoGen**: microsoft.github.io/autogen
- **DeepLearning.AI**: "AI Agents in LangGraph" short course.

Questions?