

LLM Agent Basics

BY JACK YANSONG LI

Liii Network

Email: yansong@liii.pro



Figure. QQ Group QR Code

Table of contents

1 What is a LLM Agent?	1
1.1 Workflow	3
Workflow products:	3
1.2 Coding Agent	3
Coding agent products:	3
Appendix A Shell command basics	3
Appendix B Version Control	3

1 What is a LLM Agent?

We give an informal definition for LLM agent:

Informal Definition 1. A LLM agent is defined as:

$$\text{Agent} = \text{Context} + \text{LLMs} + \text{Tool Use}.$$

In practice, we can also classify LLM agents into therapy agent, service agent, coding agent... We will focus mostly on coding agent in this short lecture. However, we take a Customer Service Agent as our first example by its simplicity.

Example 1. (Customer Service Agent)

- Context: Product Manual. New Employee Training Manual...
- LLMs: Any LLMs

- Tool Use: call transfer, web link...

Informal Definition 2.

Long-term Memory=Context Window=Input Token Limit

1.5 Token \approx 1 Word

- **Question 1.** What's gonna happen if we put an entire Product Manual as a context? Will the agent behave properly?

Answer 1. No, the agent will forget most of the contents! One solution to Question 1 is by building a workflow, a topic we will cover later in this course.

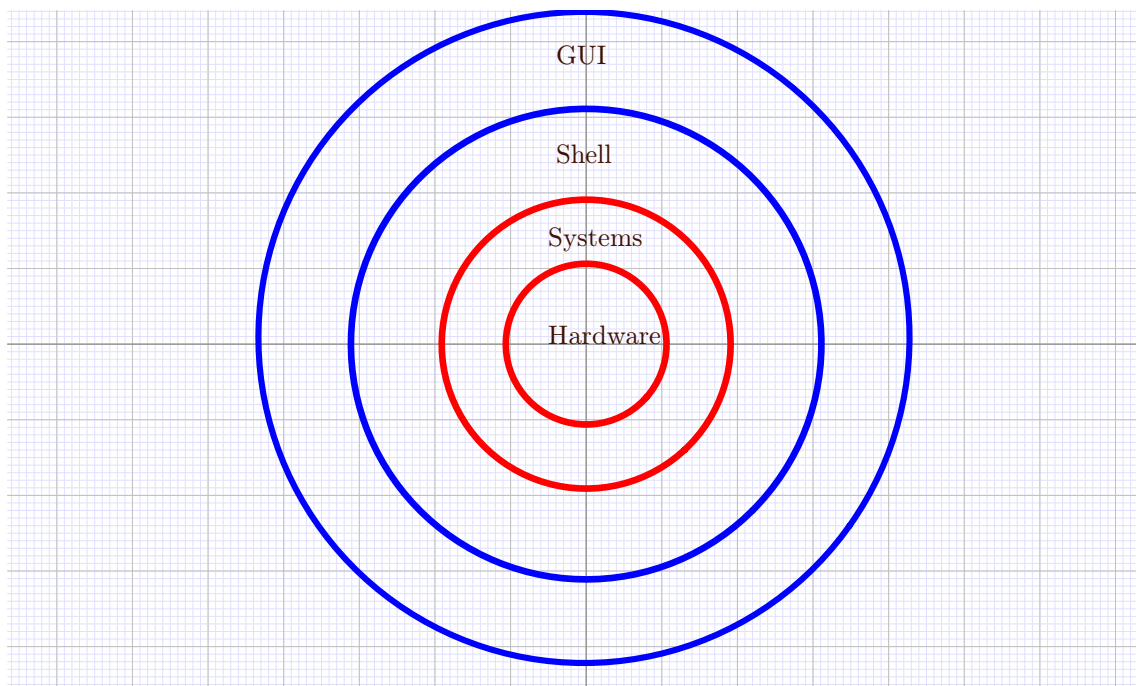


Figure 1. Different level of computer interactions

Remark 1. For a more tutorials on shell commands and command line tools, I highly recommend the *Missing Semester* Course provided by MIT: <https://missing.csail.mit.edu/>

Example 2. (Claude Code)

- Context: Code, Documents, *System Prompts*
- LLMs: By default Claude Sonnet & Opus. By using *Claude Code Router*, one can apply any LLM through keys.
- Tool: Shell command

Example 3. (Cline, Copilot, Trae, Cursor)

- Context: Code, Documents, *System Prompts*
- LLMs: Any LLMs
- Tool: Shell command (through VSCode, unstable, failed sometimes!), VSCode commands

- **Question 2.** Claude Code and Cursor can use the same model, why the price is different?

Answer 2. Claude code uses more token \Leftarrow Claude code has longer workflow & Claude code requires more contents!

Previously: “Stop talking and show me the code”

Now: “Stop coding and show me the talk”

1.1 Workflow

V3.1> in one sentence, introduce me the difference between LLM coding agent and LLM workflow

An LLM coding agent is a specialized application that uses an LLM to generate and execute code for specific tasks, while an LLM workflow is a broader, multi-step process that chains together various tools (which can include coding agents) to accomplish a complex goal.

V3.1> a workflow itself is one implementation of coding agent?

V3.1>

Workflow products:

- Dify (1 year free for university student, web-based)
- LangChain (require coding)

1.2 Coding Agent

Coding agent products:

- Cursor/Trae/Cline/Copilot: in VSCode
- Claude Code: in Terminal

Cursor: \$20 per year

Claude Code: \$200 per month

Remark 2. You can use Kimi K2 model directly in Claude Code (without the need for Claude Code Router). See tutorial in [Claude code installation with Kimi K2 model](#).

Claude Code lives in Shell (recall Figure 1), so the tool is your shell commands! Learn some terminal commands!

2 ReAct

2.1 ReAct v.s. workflow

What is ReAct?

ReAct (Reason + Act) is a method for AI agents to solve problems by mimicking human thought:

1. **Reason:** Think step-by-step about what to do next.

2. **Act:** Use a tool (like Search or an API) to do it.
3. **Observe:** See the result.

This loop repeats until the problem is solved. It's dynamic and flexible.

Example:

- **Thought:** "I need the capital of France."
- **Act:** Search("capital of France")
- **Observe:** "Paris."
- **Answer:** "The capital is Paris."

ReAct vs. Workflow: The Key Difference

	ReAct	Workflow (e.g., Zapier)
Nature	Dynamic. Figures out the path as it goes.	Static. Follows a pre-defined path.
Decision	Makes decisions in real-time based on reasoning.	All decisions are pre-programmed (if/else).
Best For	Open-ended, complex tasks (research, analysis).	Repetitive, predictable tasks (send email on form submit).
Analogy	A Human Assistant you give a goal to.	A Factory Robot on an assembly line.
Products	Claude Code	Dify

In short: ReAct *figures out* the steps, a Workflow *follows* the steps.

V3.1> [introduceme ReAct, and what is the difference between react and workflow](#)

V3.1>

3 Demo: local agent server

Demo Github: https://github.com/JackYansongLi/local_llm_server.

3.1 Project Overview

The Demo is an educational project that demonstrates advanced LLM tool calling functionality across multiple platforms. This project serves as a comprehensive example of modern AI agent development, showcasing:

- **Tool Calling:** Standard function calling format with extensible tool registry
- **Real-time Streaming:** Multi-phase streaming with visual feedback for different content types
- **Cross-Platform Compatibility:** Universal deployment across Windows, macOS, and Linux
- **Modern Development Practices:** Comprehensive testing, configuration management, and deployment strategies

3.2 Technology Stack

See Table 1.

Component	Technology	Purpose
LLM platform	Ollama ($\geq 0.1.0$)	CPU-based local inference
Package Management	UV	Fast, reliable dependency management
Model	Qwen3-0.6B	Tool-optimized language model
Testing	pytest	Comprehensive test suite

Table 1. Technology Stack Overview

4 Architecture and Design Patterns

4.1 High-Level Architecture

The system employs a **Universal Agent Pattern** with intelligent backend selection:

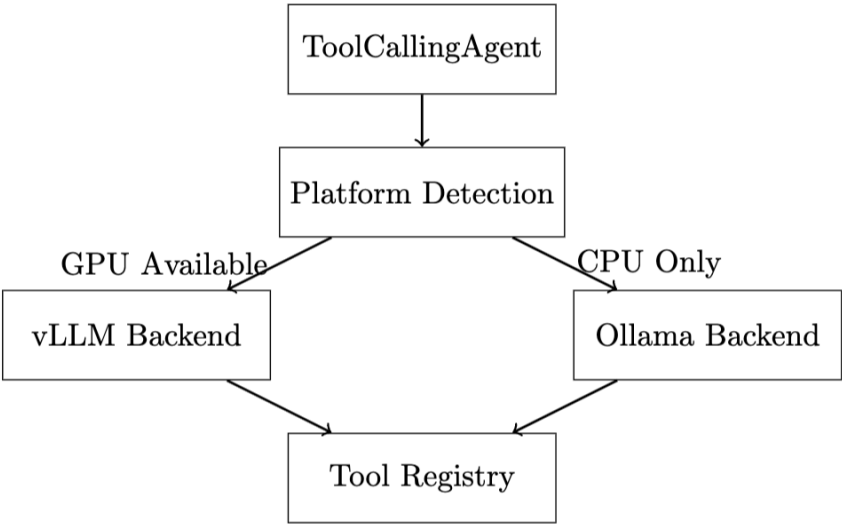


Figure 2. Universal Agent Pattern

4.2 Streaming Architecture

4.2.1 Multi-Type Streaming System

The streaming architecture provides real-time feedback with different content types:

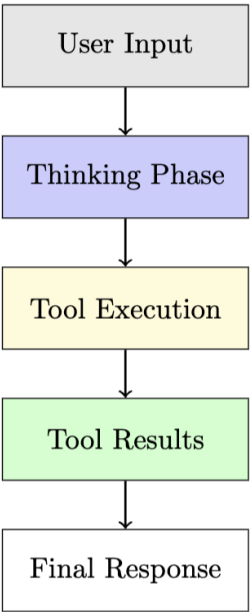


Figure 3. 1 React flow

4.2.2 Streaming Implementation

```
1 CHUNK_TYPES = {
2     "thinking": "Internal_reasoning_process",
3     "tool_call": "Tool_execution_initiation",
4     "tool_result": "Tool_execution_completion",
5     "content": "Final_response_content",
6     "error": "Error_notifications"
7 }
8
9 def chat_stream(self, message: str, use_tools: bool = True):
10     """Streaming chat with visual feedback."""
11     for chunk in self._process_stream(message):
12         chunk_type = chunk.get("type")
13         content = chunk.get("content", "")
14
15         if chunk_type == "thinking":
16             # Gray text for internal reasoning
17             print(f"\033[90m{content}\033[0m", end="", flush=True)
18             yield {"type": "thinking", "content": content}
19
20         elif chunk_type == "tool_call":
21             # Blue notification for tool execution
22             print(f"Tool: {content['name']}")
23             yield {"type": "tool_call", "content": content}
24
25         elif chunk_type == "tool_result":
26             # Green result display
27             print(f"Result: {content}")
28             yield {"type": "tool_result", "content": content}
29
30         elif chunk_type == "content":
31             # Standard text for final response
32             print(content, end="", flush=True)
33             yield {"type": "content", "content": content}
```

4.2.3 Registry Pattern: Tool Management

Extensible tool system with dynamic registration:

```
1 class ToolRegistry:
2     def __init__(self):
3         self.tools = {}
4         self._register_builtin_tools()
5
6     def register_tool(self, name: str, function: callable,
7                     description: str, parameters: Dict):
8         """Register a new tool with OpenAI-compatible schema."""
9         self.tools[name] = {
10             "function": function,
11             "description": description,
12             "parameters": parameters
13         }
14
15     def execute_tool(self, name: str, arguments: Dict) -> str:
16         """Execute a registered tool with error handling."""
17         if name not in self.tools:
```

```

18         return json.dumps({"error": f"Tool_{name}_not_found"})
19
20     try:
21         tool_func = self.tools[name]["function"]
22         result = tool_func(**arguments)
23         return json.dumps({"success": True, "result": result})
24     except Exception as e:
25         return json.dumps({
26             "success": False,
27             "error": str(e),
28             "traceback": traceback.format_exc()
29         })

```

A Homework

1. Install Debian 13
2. Build an Chatbot in Dify that can answer questions about BNBU
3. Install Claude Code with KIMI-K2 API KEY.
4. Finish Appendix B and Appendix C.
5. Add `ls`, `cd`, `mkdir`, `touch`, `echo`, `>>`, tool for the demo agent.
 - `ls` in Windows Powershell is `Get-ChildItem`.

B Shell command basics

See [Missing Semester Lecture 1](#). Most common shell “languages” are:

- Windows: Command Prompt, Powershell, Elvish.
- MacOS: Zsh, Elvish.
- Linux: Zsh, Bash, Fish, Elvish.

(Bash) Shell command includes make folder: `mkdir`, print working directory: `pwd`, list contents in the current working directory: `ls`

C Version Control

See [Missing Semester Lecture 6](#).

- **Question 3.** What is the different between `Git`, `GitHub`, `Gitee`, and `Gitlab`.

Answer 3. `Git` is a version control software/tool. The other three are “websites” that can be used as remote repositories through `Git`.