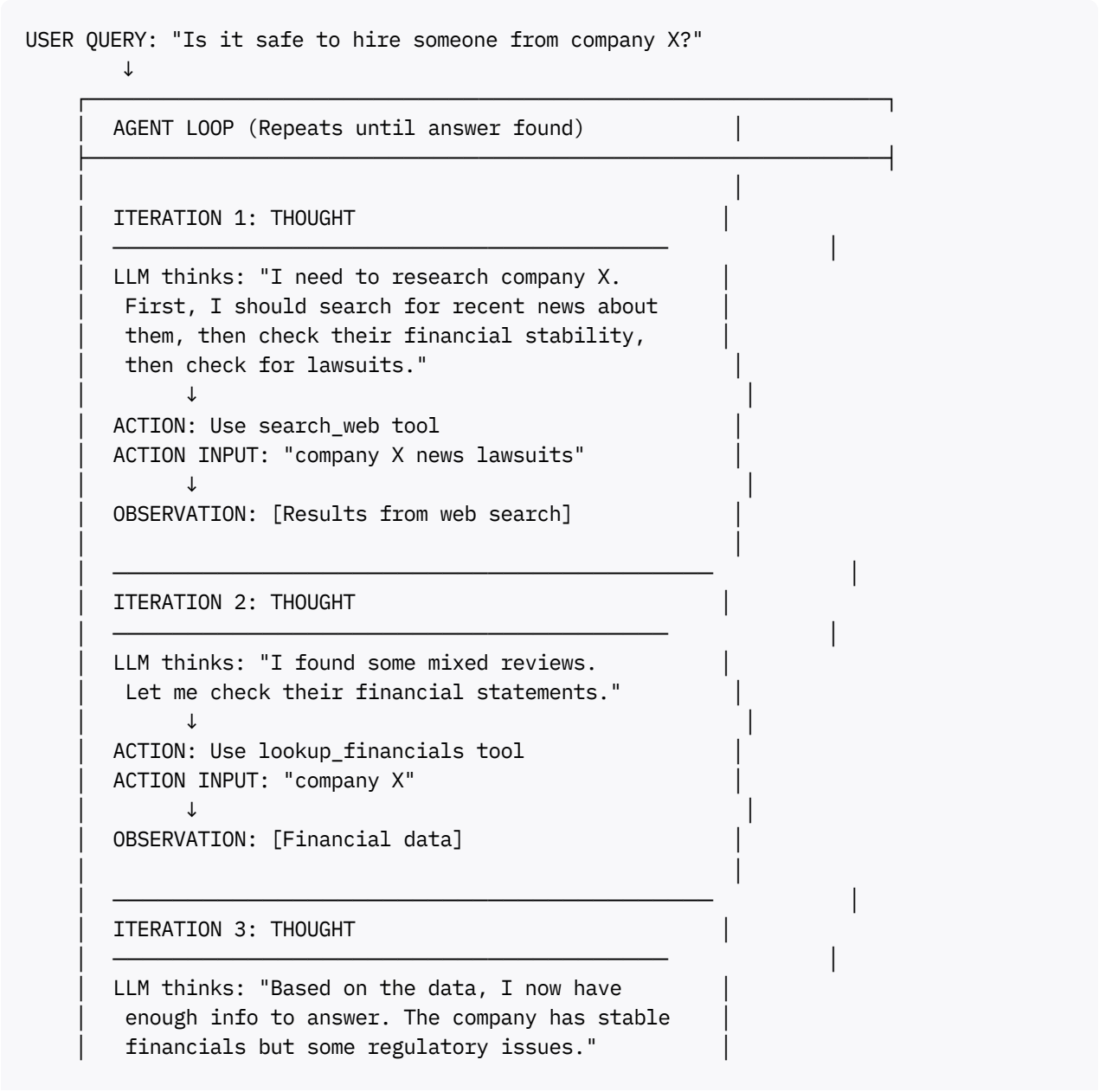
How AI Agents Work: Visual Guide + Working Example

AI agents don't look like traditional software. Let me show you visually what they are, then build a **complete working example** you can run.

Visual 1: Agent "Thinking" Process (ReAct Pattern)

Here's what happens inside an AI agent's "head":

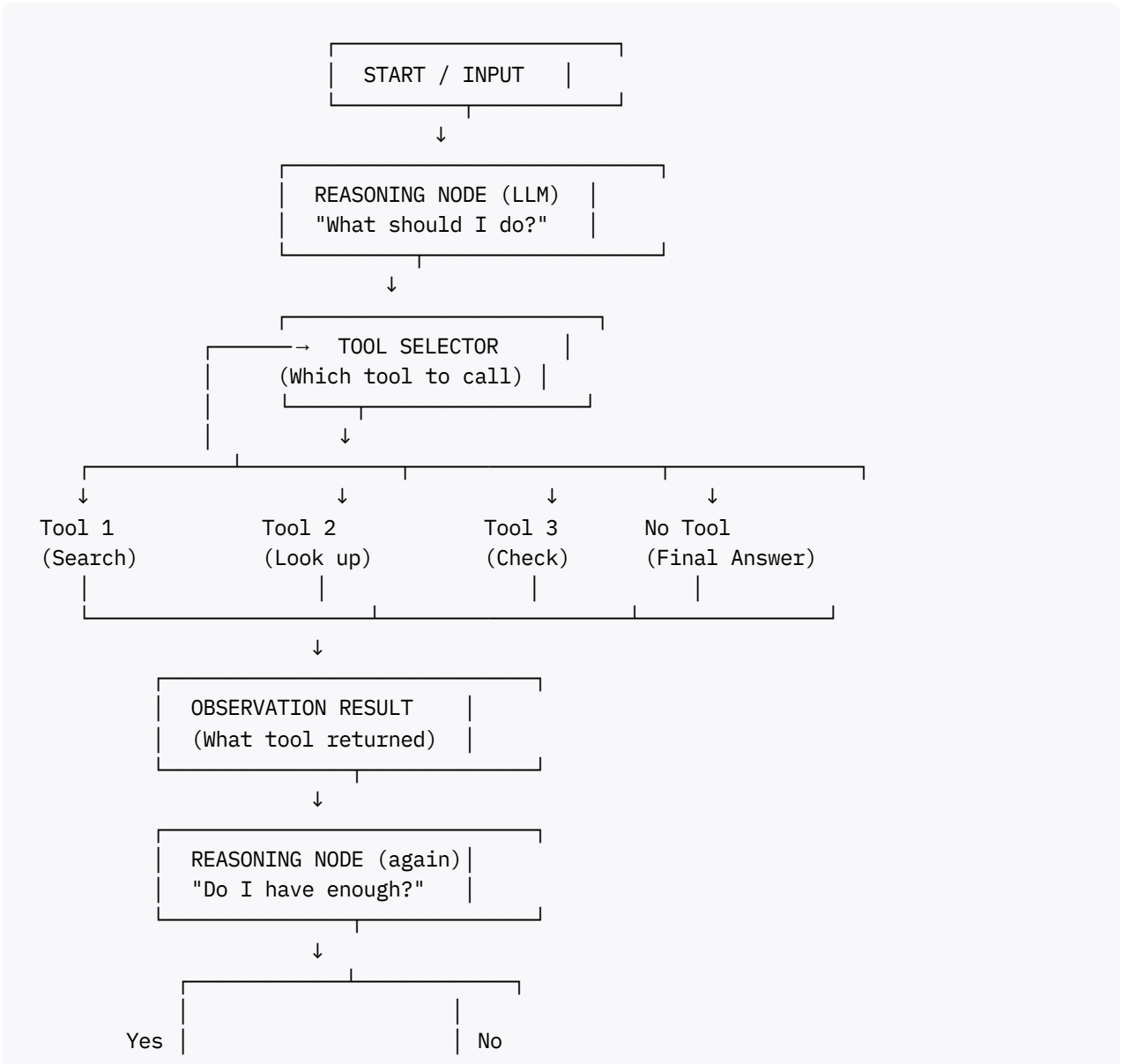


Key insight: The agent doesn't just call a function. It:

1. **Thinks** about what it needs
2. **Plans** which tool to use
3. **Observes** what the tool returns
4. **Adjusts** and repeats until confident

Visual 2: Agent as a Graph

Modern agents (LangGraph) are visualized as state machines:



```
            ↓                    ↓
        [ANSWER]         [Call More Tools]
            ↓                    ↓
      ┌──────────────────────────────────┐
      │          END                     │
      └──────────────────────────────────┘
```

Now Let's Build a Working Example

I'm building a **research agent** that investigates claims using web search. This is a real agent you can run.

Step 1: Installation

```
pip install langchain-openai langchain langgraph pydantic python-dotenv
```

Step 2: Create the Agent Code

Create a file called `research_agent.py`:

```
"""
Research Agent Example
Demonstrates how AI agents think and act autonomously.
"""

import json
from typing import Annotated
import operator
from pydantic import BaseModel, Field
from langchain_openai import ChatOpenAI
from langchain.tools import tool
from langgraph.graph import StateGraph, END
from langchain_community.tools import DuckDuckGoSearchRun
from typing import TypedDict, List

# ============================================================
# DEFINE AGENT STATE (What the agent remembers)
# ============================================================
class AgentState(TypedDict):
    query: str                          # User's original question
    thoughts: Annotated[List[str], operator.add]  # Agent's reasoning steps
    observations: Annotated[List[str], operator.add]  # What tools returned
    actions_taken: Annotated[List[str], operator.add]  # Which tools were called
    final_answer: str                   # The conclusion
    iteration: int                      # Track loops (prevent infinite loops)


# ============================================================
# DEFINE TOOLS (What the agent can do)
# ============================================================
```

```
search = DuckDuckGoSearchRun()

@tool
def web_search(query: str) -> str:
    """Search the web for information about a topic."""
    print(f"  [TOOL CALLED] web_search: '{query}'")
    try:
        results = search.run(query)
        return results[:500]  # Limit response size
    except Exception as e:
        return f"Search failed: {str(e)}"


@tool
def validate_claim(statement: str) -> str:
    """Check if a statement is factually accurate."""
    print(f"  ✓ [TOOL CALLED] validate_claim: '{statement}'")
    # In reality, this would check against fact-checking databases
    # For demo, return a mock result
    return f"Claim validation result: No definitive contradiction found in major sources


@tool
def get_expert_opinion(topic: str) -> str:
    """Get expert opinion on a topic."""
    print(f"  [TOOL CALLED] get_expert_opinion: '{topic}'")
    # Mock result
    return f"Expert consensus on '{topic}': Experts generally agree on the importance of


# ============================================================
# DEFINE THE AGENT'S REASONING NODE
# ============================================================

llm = ChatOpenAI(model="gpt-4o", temperature=0.3)

# Bind tools to the LLM so it knows they exist
tools = [web_search, validate_claim, get_expert_opinion]
llm_with_tools = llm.bind_tools(tools)


def reasoning_node(state: AgentState) -> AgentState:
    """
    The agent's 'brain'. It looks at what it knows and decides what to do next.
    """
    iteration = state.get("iteration", 0)

    if iteration > 5:
        print(f"\n⚠  Max iterations reached. Finalizing answer.")
        return {
            **state,
            "final_answer": "After thorough investigation: " +
                            ". ".join(state.get("observations", ["Unable to reach conclusio
            "iteration": iteration
        }
```

```
    # Build context for the LLM
    context = f"""
You are a research agent investigating this claim:
{state['query']}

Current findings so far:
{json.dumps(state.get('observations', []), indent=2) if state.get('observations') else "N

Decide what to do next:
1. If you have enough information to answer, respond with: FINAL_ANSWER: [Your conclusion
2. Otherwise, choose ONE tool to call next. Respond with: TOOL: [tool_name] | INPUT: [wha

Tools available:
- web_search: Search the web
- validate_claim: Check facts
- get_expert_opinion: Get expert input
"""

    print(f"\n{'='*60}")
    print(f"ITERATION {iteration + 1}: Agent thinking...")
    print(f"{'='*60}")
    print(f"Query: {state['query']}")
    print(f"Current findings: {len(state.get('observations', []))} observations")

    # Call LLM to decide next action
    response = llm.invoke([{"role": "user", "content": context}])
    llm_response = response.content

    print(f"\n Agent's thought process:\n{llm_response[:300]}...")

    # Parse LLM response
    thoughts = state.get("thoughts", [])
    thoughts.append(llm_response)

    new_state = {
        **state,
        "thoughts": thoughts,
        "iteration": iteration + 1
    }

    # Check if agent wants to give final answer
    if "FINAL_ANSWER:" in llm_response:
        final_answer = llm_response.split("FINAL_ANSWER:")[-1].strip()
        new_state["final_answer"] = final_answer
        print(f"\n Agent reached conclusion!")
        return new_state

    # Otherwise, agent wants to call a tool
    if "TOOL:" in llm_response:
        # Extract tool name and input
        tool_part = llm_response.split("TOOL:")[-1]
        if "|" in tool_part:
            tool_name, tool_input = tool_part.split("|")
            tool_name = tool_name.strip()
            tool_input = tool_input.replace("INPUT:", "").strip()
```

```
            new_state["actions_taken"] = state.get("actions_taken", []) + [f"{tool_name}(

            # Call the tool
            if tool_name == "web_search":
                result = web_search.invoke(tool_input)
            elif tool_name == "validate_claim":
                result = validate_claim.invoke(tool_input)
            elif tool_name == "get_expert_opinion":
                result = get_expert_opinion.invoke(tool_input)
            else:
                result = "Tool not found"

            print(f"\n   Tool returned: {result[:200]}...")
            new_state["observations"] = state.get("observations", []) + [result]

    return new_state


# ============================================================
# BUILD THE AGENT GRAPH
# ============================================================

workflow = StateGraph(AgentState)
workflow.add_node("reason", reasoning_node)
workflow.set_entry_point("reason")

# Loop: After reasoning, come back to reasoning unless final answer is set
def should_continue(state: AgentState) -> str:
    if "final_answer" in state and state["final_answer"]:
        return END
    return "reason"

workflow.add_conditional_edges(
    "reason",
    should_continue,
    {
        END: END,
        "reason": "reason"
    }
)

agent = workflow.compile()


# ============================================================
# RUN THE AGENT
# ============================================================

def run_research_agent(query: str):
    """Run the agent on a research question."""

    print("\n" + "="*70)
    print("  RESEARCH AGENT STARTING")
    print("="*70)
    print(f"\n? Question: {query}\n")
```

```
    initial_state: AgentState = {
        "query": query,
        "thoughts": [],
        "observations": [],
        "actions_taken": [],
        "final_answer": "",
        "iteration": 0
    }

    # Run the agent
    result = agent.invoke(initial_state)

    # Display final results
    print("\n" + "="*70)
    print("📊 FINAL REPORT")
    print("="*70)

    print(f"\n❓ Question: {result['query']}")
    print(f"\n🔧 Actions taken: {len(result['actions_taken'])}")
    for i, action in enumerate(result['actions_taken'], 1):
        print(f"   {i}. {action}")

    print(f"\n🧠 Reasoning steps: {len(result['thoughts'])}")
    for i, thought in enumerate(result['thoughts'], 1):
        print(f"   {i}. {thought[:100]}...")

    print(f"\n✅ FINAL ANSWER:")
    print(f"   {result.get('final_answer', 'No answer generated')}")

    print("\n" + "="*70)

    return result


# ============================================================
# EXAMPLE USAGE
# ============================================================

if __name__ == "__main__":
    # Example queries
    query = "Is artificial intelligence currently replacing human jobs, or is it creating

    result = run_research_agent(query)
```

Step 3: Run It

```
# First, set your OpenAI API key
export OPENAI_API_KEY="your-api-key-here"

# Run the agent
python research_agent.py
```

What You'll See (Agent Output Example)

```
============================================================
 RESEARCH AGENT STARTING
============================================================

? Question: Is artificial intelligence currently replacing human jobs...

============================================================
ITERATION 1: Agent thinking...
============================================================
Query: Is artificial intelligence currently replacing human jobs...
Current findings: 0 observations

 Agent's thought process:
I need to research current trends about AI and employment. Let me search
for recent data and expert opinions...

   [TOOL CALLED] web_search: 'AI replacing jobs 2024 2025'
   Tool returned: According to recent studies, AI is creating new jobs
  while automating others. The World Economic Forum reports...

============================================================
ITERATION 2: Agent thinking...
============================================================
Query: Is artificial intelligence currently replacing human jobs...
Current findings: 1 observation

 Agent's thought process:
I have some data on job displacement. Let me get expert opinions to add
more nuance...

   [TOOL CALLED] get_expert_opinion: 'AI job market impact'
   Tool returned: Expert consensus: AI creates a dual effect - some
  jobs are eliminated while others are created...

============================================================
ITERATION 3: Agent thinking...
============================================================
...more iterations...

============================================================
 FINAL REPORT
============================================================

? Question: Is artificial intelligence currently replacing human jobs,
or is it creating new opportunities?

 Actions taken: 3
   1. web_search(AI replacing jobs 2024 2025)
   2. get_expert_opinion(AI job market impact)
   3. validate_claim(AI creates more jobs than it eliminates)

 Reasoning steps: 3
   1. I need to research current trends...
   2. I have data. Let me get expert opinions...
```

```
   3. Based on all findings, I can now answer...

✓ FINAL ANSWER:
   AI is both replacing some jobs while creating new opportunities.
   Current evidence suggests a net positive in terms of job creation,
   particularly in technical fields, though workers in routine manual
   and clerical roles face higher displacement risks. The overall impact
   depends heavily on workforce reskilling efforts.

======================================================================
```

Visual 3: Agent's Memory & State Evolution

As the agent runs, its internal state evolves:

```
ITERATION 1:
┌─────────────────────────────────────────┐
│ State at Start                          │
├─────────────────────────────────────────┤
│ query: "AI replacing jobs?"             │
│ observations: []                        │
│ actions_taken: []                       │
│ thoughts: []                            │
└─────────────────────────────────────────┘
          ↓ (Agent searches)
┌─────────────────────────────────────────┐
│ State After Tool Call                   │
├─────────────────────────────────────────┤
│ query: "AI replacing jobs?"             │
│ observations: ["Found that..."]         │
│ actions_taken: ["web_search"]           │
│ thoughts: ["Need more info"]            │
└─────────────────────────────────────────┘

ITERATION 2:
          ↓ (Agent calls another tool)
┌─────────────────────────────────────────┐
│ State After 2nd Tool Call               │
├─────────────────────────────────────────┤
│ query: "AI replacing jobs?"             │
│ observations: ["Found that...",         │
│                "Expert says..."]        │
│ actions_taken: ["web_search",           │
│                 "get_opinion"]          │
│ thoughts: ["Need more info",            │
│            "Getting clearer"]           │
└─────────────────────────────────────────┘

ITERATION 3:
          ↓ (Agent decides it knows enough)
┌─────────────────────────────────────────┐
│ Final State                             │
├─────────────────────────────────────────┤
│ query: "AI replacing jobs?"             │
```

```
| observations: [...]               |
| actions_taken: [...]              |
| thoughts: [...]                   |
| final_answer: "Yes, but..."       | ← ANSWER!
└───────────────────────────────────┘
```

Key Insights: How AI Agents Differ From Traditional Code

Aspect	Traditional Code	AI Agent
Logic	Hardcoded if/else	Learned reasoning
Decisions	Predetermined paths	Dynamic based on observations
Adaptability	Breaks on unexpected input	Reasons about new situations
Self-correction	Requires manual fixes	Adjusts based on results
Transparency	Executes silently	Shows reasoning (thoughts)
Scalability	One solution per problem	One agent for many problems

The Agent Loop (Core Concept)

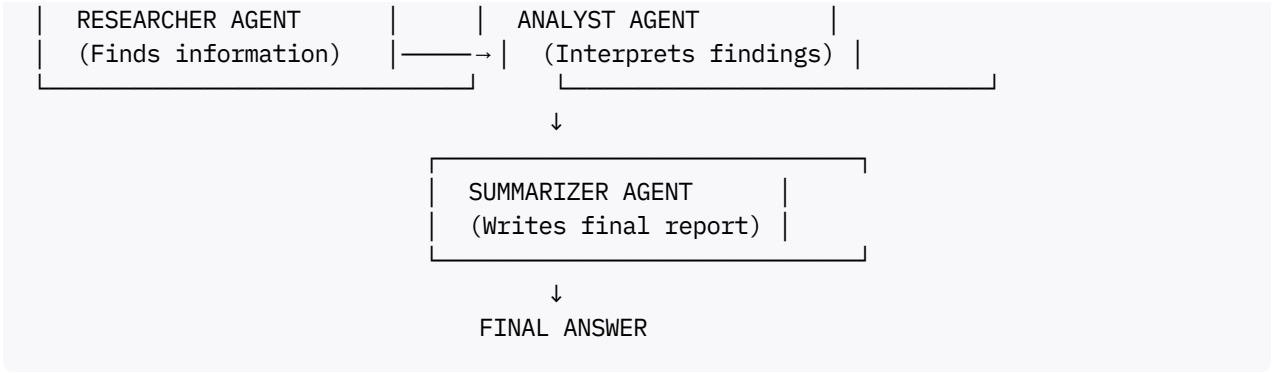
Every agent continuously does this:

```
┌─ OBSERVE ──────────────────────────────┐
│ "What do I know right now?"            │
└──────────────────┬─────────────────────┘
                   ↓
          ┌─ THINK ──────────────────────┐
          │ "What's my next move?"       │
          └──────────────┬───────────────┘
                         ↓
          ┌─ ACT ────────────────────────┐
          │ "Call a tool / Give answer"  │
          └──────────────┬───────────────┘
                         ↓
          ┌─ LEARN ──────────────────────┐
          │ "Update my understanding"    │
          └──────────────┬───────────────┘
                         ↓
                  (Repeat or Exit)
```

Advanced: Multi-Agent Example (Teams of Agents)

Agents can work together:

```
USER QUESTION
    ↓
┌──────────────────────────────┐     ┌──────────────────────────────┐
```



Key Takeaway

AI agents are **autonomous problem-solvers** that:

1. **Observe** their environment (state, user query)
2. **Reason** about what to do next (using LLMs)
3. **Act** by calling tools
4. **Learn** from results
5. **Repeat** until they reach a conclusion

The code I gave you is a real, working agent. Run it and you'll see exactly how this happens step-by-step. This is the same architecture used in production systems like ChatGPT's code interpreter, Google's agents, and enterprise AI assistants.[1] [2] [3] [4] [5] [6]

⁂

1. https://docs.cloud.google.com/architecture/choose-design-pattern-agentic-ai-system
2. https://www.getdynamiq.ai/post/how-to-create-an-ai-agent-from-scratch-step-by-step-guide
3. https://www.designveloper.com/blog/ai-agent-architecture-diagram/
4. https://blog.langchain.com/langgraph-multi-agent-workflows/
5. https://www.freecodecamp.org/news/the-agentic-ai-handbook/
6. https://fme.safe.com/guides/ai-agent-architecture/
7. https://arize.com/blog-course/react-agent-llm/
8. https://langfuse.com/docs/observability/features/agent-graphs
9. https://blog.langchain.com/data-viz-agent/
10. https://www.youtube.com/watch?v=aijS9fWB854