Rainy Day Stocks

Building Your First AI Agent

A technical workshop on ReAct patterns, Tool Calling, and LLM orchestration.

Hypothesis: Do rainy days cause stock prices to drop?

Source: alfasin/stock-weather-agent

Chatbots Talk. Agents Do.

The Chatbot

Standard LLM

- **Stateless:** Forgets context immediately.
- **No Tools:** Cannot access real-time data.
- **Single Turn:** Input → Output.
- **Result:** Plausible Hallucination.

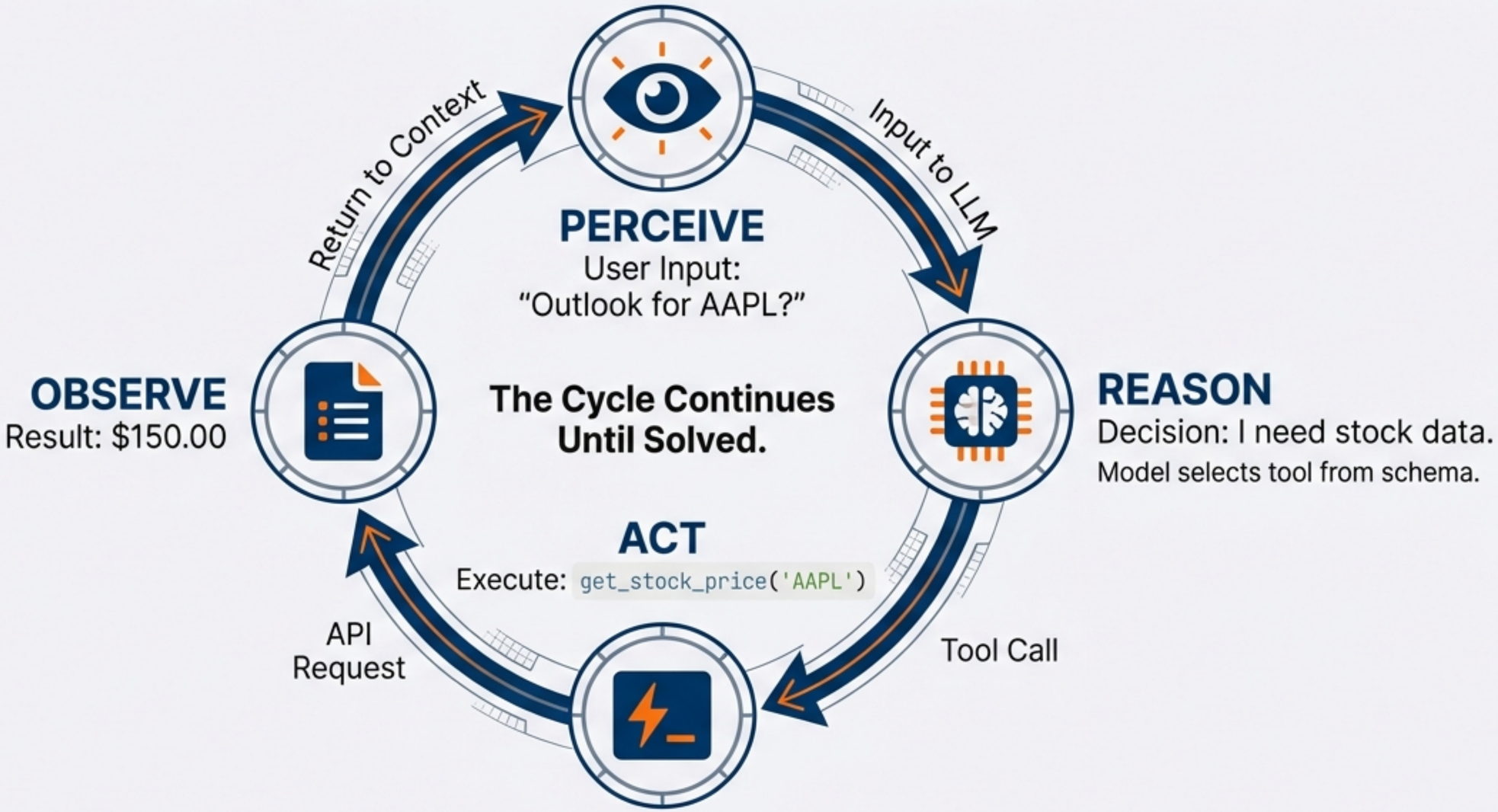
The Agent

Agentic Workflow

- **Memory:** Maintains history across turns.
- **Tool Use:** APIs, Functions, Database access.
- **The Loop:** Reason → Act → Observe → Repeat.
- **Result:** Actionable Fact.

To test the Rainy Day Hypothesis, we need to fetch live weather and market data. A **chatbot cannot** do this. **We need an Agent.**

The Anatomy of Agency: The ReAct Loop



The Engine Room

High-performance stack for local agent development.

The Brain (Inference)

groq

Model: llama-3.1-8b-instant

Speed: Up to 30K TPM (Free Tier)

Role: Fast inference for multi-step loops.

The Data (World Knowledge)

Financial Modeling Prep

Real-time Stock Prices

Open-Meteo

Weather WMO Codes (No Key Required)

The Core (Orchestration)

uv

Language: Python 3.10+

Manager: uv (Ultra-fast package sync)

Structure: No frameworks initially. Raw Python.

16:9 aspect ratio

Paper White
(#F5F5F7)

Initialize the Environment

```
> git clone stock-weather-agent
> cd stock-weather-agent
> uv sync
[uv] Resolved 42 packages in 0.12ms...
> cp .env.example .env
```

Configuration Guide

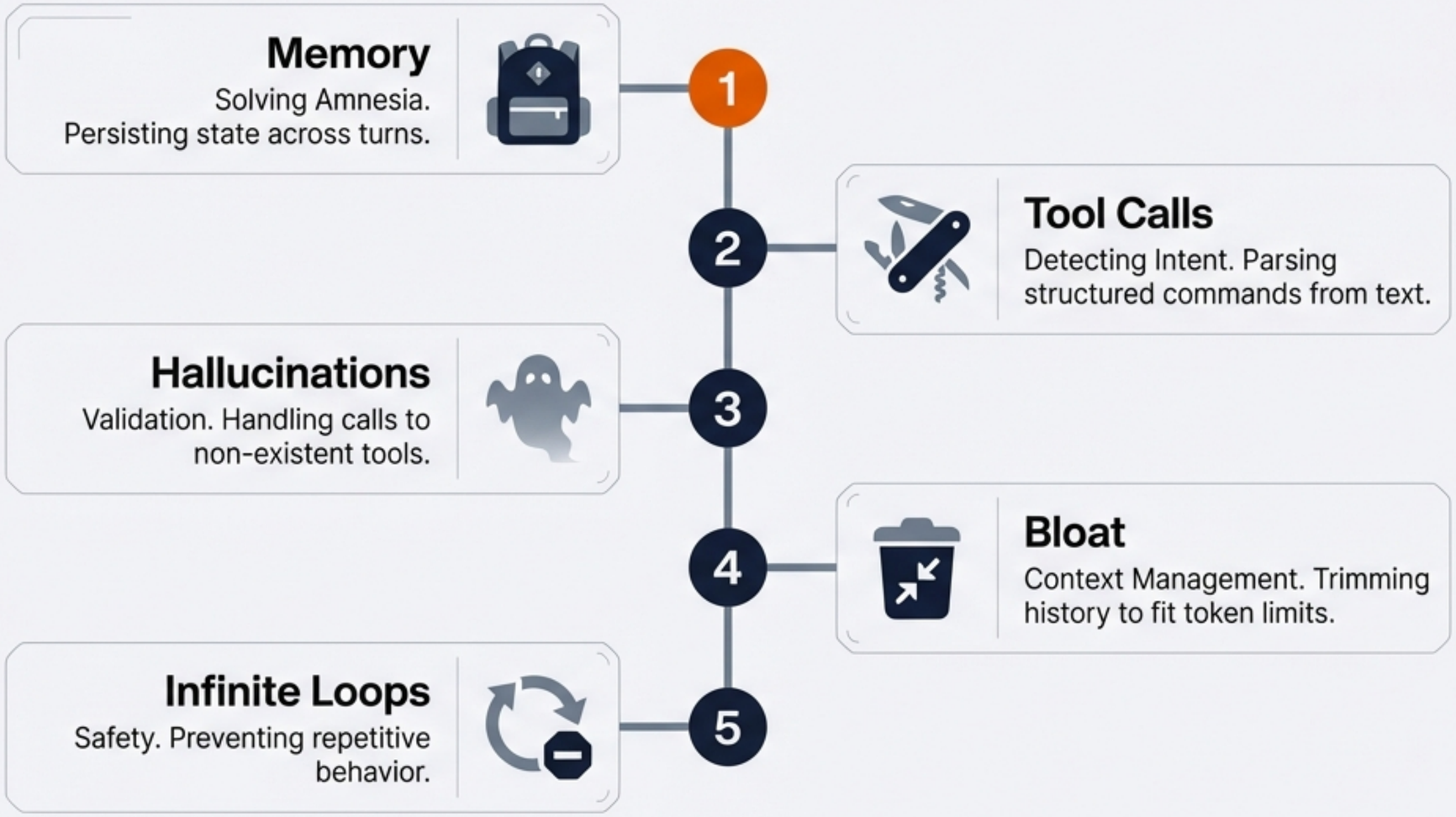
Environment Variables (.env)

1. **GROQ_API_KEY**: [Required] The reasoning engine.
2. **FMP_API_KEY**: [Optional] Stocks. Falls back to mock data if missing.
3. **Open-Meteo**: [Free] No key needed.

Run the agent: `uv run python main.py "What's the outlook for AAPL?"`

Workshop Phase 1: Building the ReAct Agent

Five engineering challenges found in "assignments/react_agent.py"



Engineering Context and Action

Exercise A Inter, Deep Navy (#003366)

The Memory List

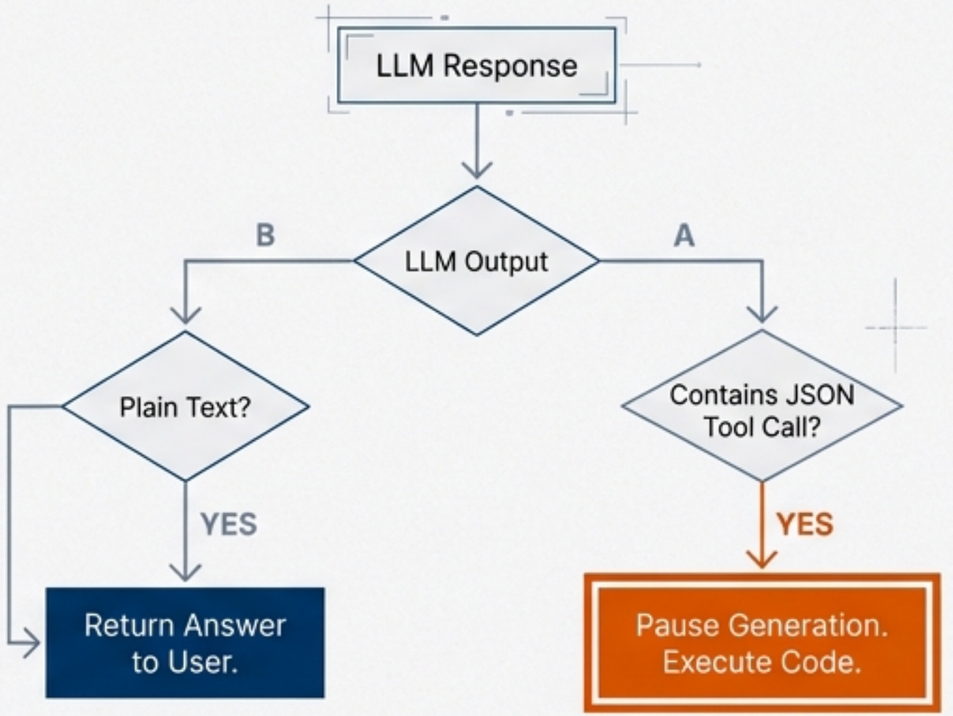
The Context Window.
Must be passed back to the LLM every single time.

```
messages = [
  {"role": "user", "content": "Hi"},
  {"role": "assistant", "content": "Hello!"},
  {"role": "user", "content": "Stock price?"}
]
```

Without appending history, the agent is stateless.

Exercise B Inter, Deep Navy (#003366)

The Decision Tree



We must programmatically detect if the LLM is 'talking' or 'acting'.

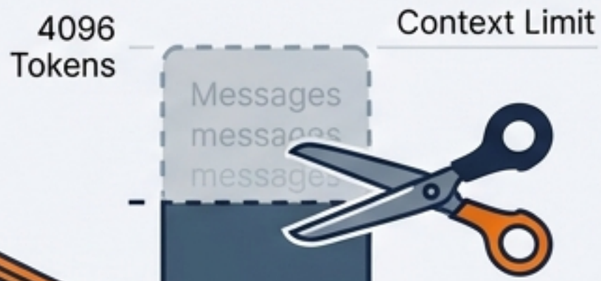
Handling the Chaos (Edge Cases)

Exercise C
Hallucinations


```
try:
   tool = registry[name]
except KeyError:
   return "Error: Tool not found"
```

JetBrains Mono

Exercise D
Memory Bloat



Strategy: FIFO (First In, First Out). Remove oldest messages when limit > 4096 tokens.

Exercise E
Infinite Loops

Max Retries. Force exit if the agent spins in circles.

The ‘Hello World’ of Tooling

Stock Tool

tools/stock_tool.py

JetBrains Mono

```
def get_stock_price(ticker: str):
    """Get current price and daily change."""
    # Check cache (24h validity)
    if ticker in cache: return cache[ticker]

    # Call FMP API
    return requests.get(f".../quote/{ticker}")
```

Includes caching to save API limits.

Weather Tool

tools/weather_tool.py

JetBrains Mono

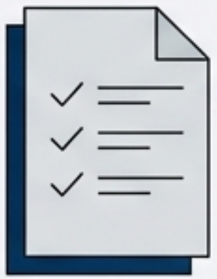
```
def get_weather(city: str):
    """Get weather & interpret WMO codes."""
    data = open_meteo.get(city)
    code = data['current']['weather_code']

    # 51-67, 80-82 indicate rain
    is_raining = code in [51, 53, 55, 61, ...]
    return {"rain": is_raining, "temp": ...}
```

Logic layer converts raw codes to semantic meaning (Rain/No Rain).

Workshop Phase 2: The Planning Agent

Alternative Pattern: Plan first, then execute.

Step 1: The Plan

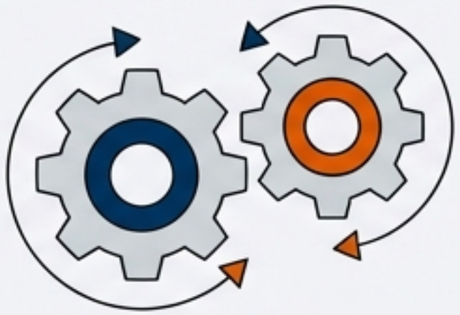


Generate Plan

```
["get_weather", "get_stock"]
```

LLM reviews request and lists all necessary tools upfront.

Step 2: Execution



Parallel Execution

System fires all tool calls in the list. No waiting for intermediate reasoning.

Step 3: Synthesis

Final Response

LLM receives all data at once and writes the answer.

Pattern Comparison

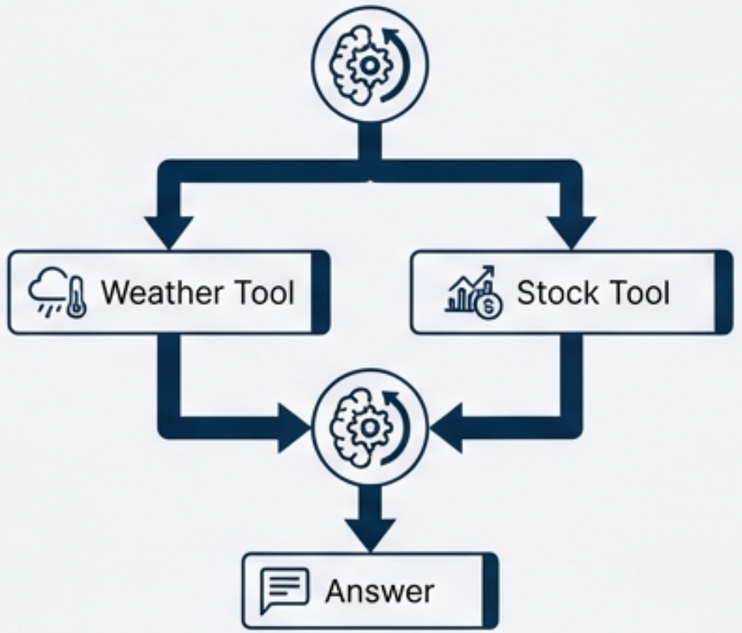
Query: “What’s the outlook for NVDA in NYC?”

ReAct (The Zig-Zag)

Interleaved Reasoning

Best For: Dynamic tasks where step 2 depends on step 1.

Planning (The Fork)



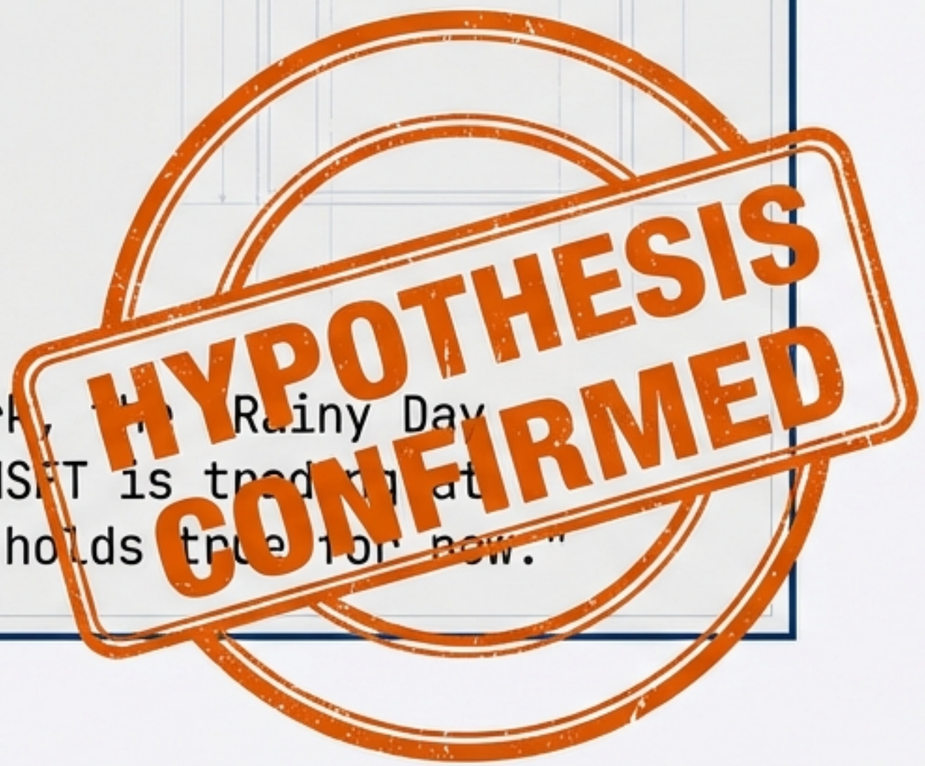
Sequential/Parallel Execution

Best For: Tasks with known, independent steps.

```
Run the planning agent: `uv run python main.py --planning ...`
```

The Result: Testing the Hypothesis

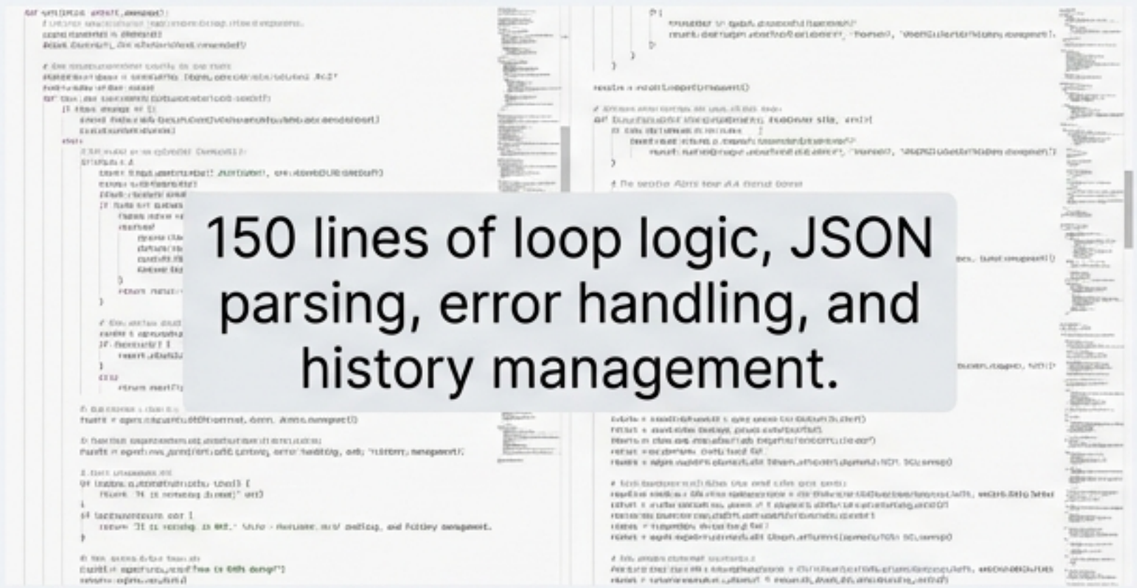
```
User: I'm looking at MSFT stock. Check the weather in New York and
tell me what you think.

[Agent] Thinking...
[Tool] Calling get_weather('New York')
[Obs] Rain detected (Code 61).
[Agent] Thinking...
[Tool] Calling get_stock_price('MSFT')
[Obs] Price: $410.00 | Change: -0.5%
[Agent] Response:

"Given it is currently raining in New York, the 'Rainy Day
Hypothesis' suggests a bearish outlook. MSFT is trading at
$410.00, down 0.5% today. The hypothesis holds true for now."
```



The Abstraction Layer

Manual Implementation vs. Framework (`pydantic-ai`)

The Hard Way (Manual)



150 lines of loop logic, JSON parsing, error handling, and history management.

The Smart Way (Framework)

```
@agent.tool
def get_weather(ctx, city: str):
    return "It is raining in NYC."
```

20 lines. The framework

```
# The entire ReAct loop in one line!
result = agent.run_sync("How is NVDA doing?")
```

Building from scratch teaches you *how* it works. Frameworks let you scale it.

Modern Engineering Editorial

Start Building

GitHub Repository

`github.com/alfasin/stock-weather-agent`

Next Steps

1. **Fork** the repository.
2. **Complete** the TODO items in `assignments/react_agent.py`.
3. **Compare** your manual loop against `bonus/pydantic_ai_version.py`.

> **You have the tools. Now go prove—or disprove—the hypothesis.”**