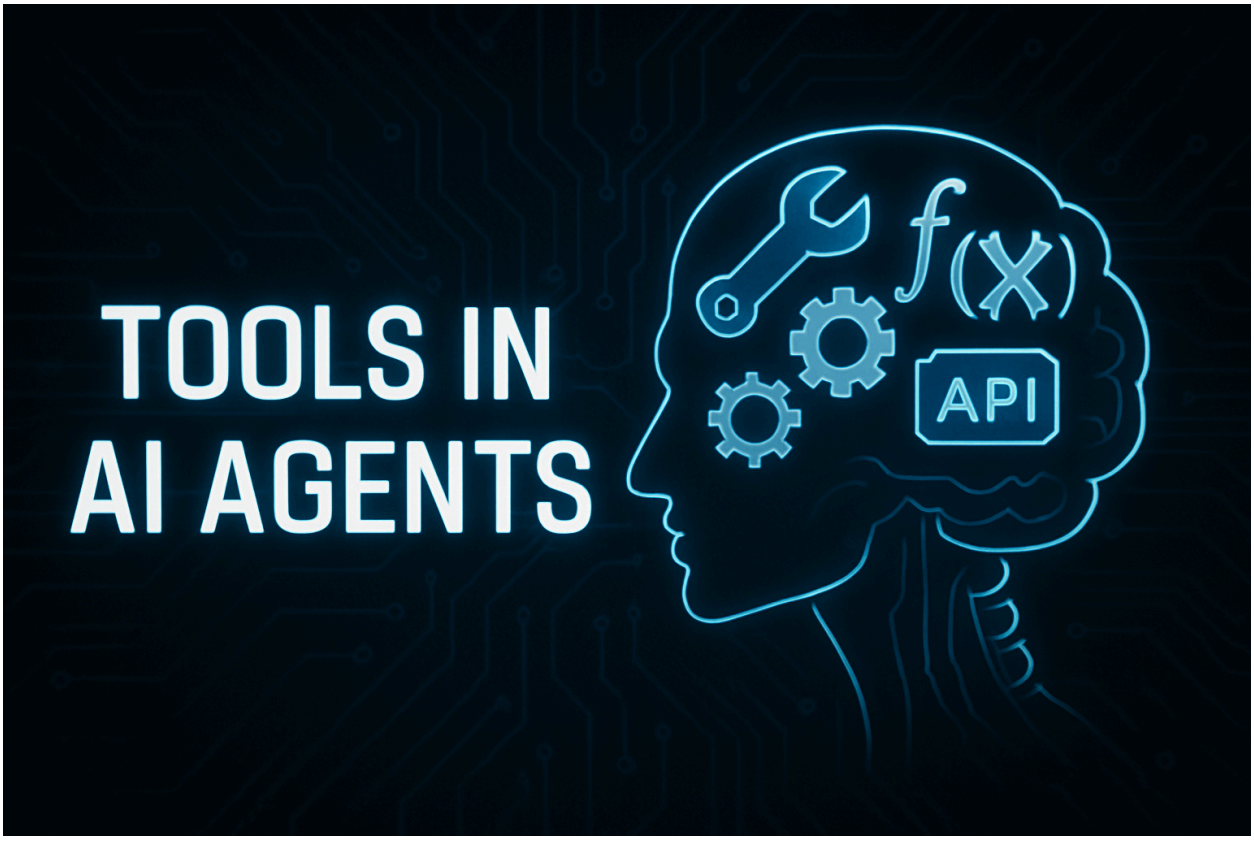
Learning AI Agents - Lesson 5: Tools in AI Agents

Type	@datasciencebrain



When building AI agents, one of the most powerful design principles is enabling them to use tools. Think of it like this: instead of relying solely on what the model "knows" from training, we let it interact with external tools to fetch data, call APIs, use code, or take actions—extending its capabilities in real time.

In this guide, I'll walk you through everything you need to know about tools in AI agents: what they are, why they matter, how to define and implement them, and how to use them in actual projects using the Hugging Face `transformers` and `transformers-agent` framework.

1. What Are Tools in AI Agents?

Tools are external functions that an agent can call during its reasoning process to accomplish tasks it cannot do on its own. This is inspired by the idea that language models are great at understanding and reasoning in natural language—but they may not always be up-to-date, precise, or capable of acting in the real world.

Examples of tools:

- A calculator for precise math
- A Python shell for running code
- A Wikipedia retriever for knowledge lookup
- An API wrapper to get stock prices or weather updates

---

2. Why Use Tools in AI Agents?

Here's why tool integration is such a game-changer in agent design:

- **Extends model capabilities:** Tools enable agents to go beyond static knowledge.
- **Keeps outputs accurate:** Tools allow access to real-time or verified data.
- **Supports automation:** Agents can trigger actions, call APIs, or make decisions.
- **Makes reasoning explainable:** Tools can be logged and traced to understand the agent's decision process.

---

3. Key Concepts to Understand

Before diving into code, here are some foundational ideas you need to be comfortable with:

a. Agent vs Model

An *agent* is a system that wraps a language model and equips it with tools, memory, and planning strategies. A *model* alone cannot call tools—it just

generates text.

b. Tool Abstraction

A tool is just a Python function with a name, description, and input/output schema. The agent must understand when and how to use it based on user prompts.

---

4. Defining Tools in Hugging Face Agents

Hugging Face provides a clean way to define tools using a `@tool` decorator. Here's how to define one from scratch.

Step-by-step example:

```
from transformers.tools import tool

@tool
def get_temperature(city: str) → str:
    """Returns the current temperature for a given city."""
    return f"The temperature in {city} is 30°C."  # Replace with real API logic
```

What's happening here:

- The `@tool` decorator registers the function so the agent can see and use it.
- The function has a clear name, description, and input/output type.
- The docstring is used by the agent to understand what the tool does.

You can also explicitly set metadata like this:

```
@tool(name="WeatherTool", description="Gets weather for a given city.")
def get_weather(city: str) → str:
    ...
```

---

5. Using Tools with an Agent

Once you've defined tools, you can create an agent that uses them.

```
from transformers import HfAgent

agent = HfAgent("https://api-inference.huggingface.co/models/bigcode/starcoder")

result = agent.run("What is the square root of 144?", remote=True)
print(result)
```

To use your own tools:

```
agent.run("What's the temperature in Kochi?", tools=[get_temperature])
```

The agent automatically:

- Interprets the prompt
- Selects the appropriate tool
- Parses arguments
- Executes the tool
- Integrates the result into the final answer

---

6. Tool Composition and Multi-Step Tasks

Agents can call **multiple tools in sequence**, solving complex tasks step-by-step.

For example:

- Use a search tool to fetch a Wikipedia article
- Use a summarizer to extract key points
- Then generate a conclusion or action

The agent manages this autonomously, as long as tools are well-defined.

---

7. Best Practices for Defining Tools

To ensure tools are reliable and easy for the agent to use:

a. Clear and Concise Descriptions

Use simple language in the docstring or description. Avoid ambiguity.

b. Type Annotations

Use proper Python type hints (`str`, `int`, `float`, etc.) for inputs and outputs. This helps the agent parse inputs accurately.

c. Error Handling

Wrap APIs or code logic in try-except blocks to avoid failures.

```
@tool
def divide(a: float, b: float) → float:
    """Divides two numbers."""
    try:
        return a / b
    except ZeroDivisionError:
        return "Cannot divide by zero."
```

d. Test Independently

Always run and test the function standalone before wiring it into the agent.

---

8. Example: Multi-Tool AI Assistant

Let’s say we want to build a research assistant that can:

1. Search Wikipedia
2. Summarize the results
3. Run calculations if needed

```
@tool
def search_wikipedia(query: str) → str:
    ...

@tool
```

```
def summarize(text: str) → str:
    ...

@tool
def calculator(expression: str) → float:
    ...
```

```
agent.run("Find me the average lifespan of a cat and convert it to months.", tools=[search_wikipedia, summarize, calculator])
```

The agent will orchestrate these calls automatically, and return the final output.

---

9. Advanced Topics

a. Using Local Models Instead of API

If you don't want to use Hugging Face's hosted API, you can load models locally:

```
from transformers import pipeline

agent = HfAgent("gpt2", local=True)
```

Note: Tool support is best with code-capable models like `bigcode/starcoder` or `deepseek-coder`.

b. Toolkits

You can package your tools into reusable toolkits and pass them to multiple agents.

```
from transformers.tools import Toolkit

my_toolkit = Toolkit([tool1, tool2, tool3])
agent.run("Do something complex", toolkits=[my_toolkit])
```

---

10. Common Challenges and How to Solve Them

Problem	Solution
Tool not used	Check description and input types. Make it clearer.
Wrong arguments passed	Use proper type hints and error messages.
Agent stuck or slow	Use simpler prompts or switch to faster model.
Unexpected output	Log tool inputs/outputs during dev and testing.

Final Thoughts

Tools are a cornerstone in making AI agents truly useful. They unlock capabilities beyond language generation—enabling your agent to interact with APIs, run logic, and complete real-world workflows.

If you're serious about building intelligent systems, mastering the tool system is essential. Start with a few simple tools, test them in isolation, then gradually combine them into more powerful agents that reason, plan, and act.

This approach makes your AI systems scalable, debuggable, and far more useful than a vanilla chatbot.
