Chapter 1: What Are AI Agents?

Introduction

You've probably heard the buzz: AI agents are transforming how we build software. Companies are deploying agents that schedule meetings, write code, conduct research, and even manage customer support—all with minimal human intervention. But what exactly *is* an AI agent? And more importantly, when should you build one?

This chapter lays the groundwork for everything you'll learn in this book. Before we write a single line of code, we need a clear mental model of what agents are, how they differ from simpler approaches, and when they're the right tool for the job. Too many developers jump straight to building agents when a simple prompt would suffice—and too many others avoid agents entirely, missing opportunities where they truly shine.

By the end of this chapter, you'll understand the full spectrum from basic prompts to autonomous agents, know when to use each approach, and have a roadmap of exactly what we'll build together throughout this book.

Learning Objectives

By the end of this chapter, you will be able to:

- Explain the spectrum from simple prompts to autonomous agents
- Distinguish between workflows and agents, and know when to use each
- Identify use cases where agents excel and where they're overkill
- Describe the core building blocks that make up an agent system
- Recognize real-world agent applications and their architectures

The Spectrum of LLM Applications

When you work with Large Language Models (LLMs) like Claude, you're not limited to a single approach. Think of it as a spectrum, with simplicity on one end and autonomy on the other:

```
Simple ──────────────────────────────────────────────────────────── Complex

Single       Multi-turn      LLM + Tools      Workflows          Autonomous
Prompt    →  Conversations → (Augmented   →  (Chained      →    Agents
                              LLM)             Operations)
```

Let's walk through each level:

Level 1: Single Prompts

The simplest interaction with an LLM is a single prompt and response. You send a message, the model responds, and you're done.

Example: "Translate 'Hello, world' to Spanish."

Best for: One-off tasks like translation, summarization, or answering questions.

Limitation: No memory, no tools, no ability to break down complex tasks.

Level 2: Multi-turn Conversations

Add conversation history, and now the LLM can reference previous messages. This enables back-and-forth dialogue.

Example: A chatbot that remembers you asked about Italian restaurants and can follow up with "What about their hours?"

Best for: Interactive applications where context matters across messages.

Limitation: Still can't take actions or access external information.

Level 3: Augmented LLMs

Give the LLM access to **tools**—functions it can call to retrieve information or perform actions. This is where things get interesting.

Example: An LLM that can:

- Check the current weather via an API
- Search a database for product information
- Perform calculations

Best for: Tasks requiring real-world data or actions beyond the LLM's training.

Limitation: You (the developer) still control the flow. The LLM responds to your prompts and uses tools when told.

Level 4: Workflows

Combine multiple LLM calls in a predetermined sequence. Each step might use different prompts, tools, or even different models.

Example: A content pipeline that:

1. Generates a draft article
2. Checks facts against a database
3. Edits for style and tone
4. Translates to three languages

Best for: Complex, multi-step tasks where you know the steps in advance.

Limitation: The steps are fixed. If the task requires adaptation, you need to code for every possibility.

Level 5: Autonomous Agents

Here's the key distinction: **agents direct their own execution**. Instead of following a predetermined path, an agent decides what to do next based on the current situation.

Example: A research agent that:

1. Receives a question: "What are the environmental impacts of lithium mining?"
2. Decides to search the web for recent articles
3. Finds an article mentioning a new study
4. Decides to look up the original study
5. Realizes it needs to understand a technical term
6. Searches for a definition
7. Synthesizes everything into a report
8. Decides the report is complete and stops

Notice that the agent made decisions at every step. It chose which tools to use, when to dig deeper, and when to stop. You didn't hardcode these decisions—the agent figured them out.

> **Key Insight:** The defining characteristic of an agent is that the LLM dynamically directs its own processes and tool usage, rather than following a predetermined path.

Workflows vs. Agents: A Critical Distinction

This distinction is so important that it deserves special attention. Understanding when to use workflows versus agents will save you countless hours of over-engineering (or under-building).

Workflows: Predetermined Control Flow

In a **workflow**, *you* (the developer) define the sequence of steps. The LLM executes each step, but the order and logic are fixed in your code.

```
[Start] → [Step 1: Generate] → [Step 2: Review] → [Step 3: Refine] → [End]
              ↓                      ↓                    ↓
           LLM Call              LLM Call             LLM Call
```

Characteristics of workflows:

- Steps are known in advance
- Control flow is in your code
- Predictable execution path
- Easier to debug and test
- More reliable and consistent

Agents: Dynamic Control Flow

In an **agent**, the LLM decides what to do next. Your code provides the tools and guidelines, but the agent chooses its own path.

```
[Start] → [LLM: What should I do?] → [Action] → [LLM: What next?] → [Action] →
...
                    ↓                                    ↓
           Could be Tool A                      Could be Tool B
           Could be Tool B                      Could be Tool A
           Could be "Done"                      Could be "Need more info"
```

Characteristics of agents:

- Steps are determined at runtime
- Control flow is in the LLM's responses
- Variable execution path
- Harder to predict and test

- More flexible but less consistent

A Practical Analogy

Think of it like giving directions:

Workflow approach: "Turn left on Main Street. Go straight for two blocks. Turn right on Oak Avenue. Your destination is on the left."

Agent approach: "Get to the library. You have a map, GPS, and can ask for directions. Figure out the best route based on traffic and construction."

The workflow is explicit and reliable. The agent is flexible but might take unexpected routes.

When to Use Each

Use Workflows When...	Use Agents When...
Steps are known in advance	Steps depend on results
Predictability matters	Flexibility matters
Errors must be contained	Recovery is part of the task
Tasks are well-defined	Tasks are open-ended
Testing is a priority	Exploration is needed
Compliance is required	Creativity is valued

> **Rule of Thumb:** Start with workflows. Only graduate to agents when the dynamic decision-making provides clear value that outweighs the added complexity.

When Agents Shine (and When They Don't)

Let's get practical about when agents are the right choice.

Agents Excel At:

1. Open-ended Research

When you don't know what you'll find or how deep you'll need to go, agents can adapt. A research agent might discover that a topic has multiple angles and decide to explore each one.

2. Complex Problem-Solving

Tasks that require trying multiple approaches, backtracking from dead ends, or combining information in unexpected ways benefit from agent autonomy.

3. Interactive Assistance

Agents that help users accomplish goals—like a coding assistant that can run code, check results, and iterate—naturally fit the agent pattern.

4. Multi-Step Tasks with Dependencies

When later steps genuinely depend on earlier results in unpredictable ways, agents can make runtime decisions that workflows can't anticipate.

Agents Are Overkill For:

1. Predictable Transformations

If you're converting data from format A to format B, that's a workflow (or even a single prompt). No agency needed.

2. Fixed Pipelines

Document processing that always follows the same steps? Use a workflow. You'll get more reliability and easier debugging.

3. Simple Q&A

Answering questions from a knowledge base? An augmented LLM with retrieval tools is sufficient. Full agency adds complexity without benefit.

4. Time-Critical Operations

Agents can take unpredictable amounts of time. If you need responses in under two seconds, workflows give you more control.

The Cost of Agency

Every time you choose agents over workflows, you're trading:

- **Predictability** for flexibility
- **Speed** for thoroughness
- **Simplicity** for capability
- **Control** for autonomy

Make this trade consciously. Many production systems that started as agents were eventually simplified to workflows once the requirements became clear.

The Building Blocks We'll Construct

Throughout this book, we'll build every component you need for production-ready agents. Here's what's coming:

Part 1: Foundations (Chapters 1-6)

The basics: setting up your environment, making API calls, understanding conversations, and crafting system prompts. Everything starts here.

Part 2: The Augmented LLM (Chapters 7-14)

The core building block for everything else. You'll learn to:

- Define tools the LLM can use
- Handle tool calls and return results
- Build multi-tool agents
- Create sequential tool chains
- Parse structured outputs

By Chapter 14, you'll have a complete `AugmentedLLM` class that serves as the foundation for all future work.

Part 3: Workflow Patterns (Chapters 15-25)

Five battle-tested patterns for combining LLM calls:

1. **Prompt Chaining:** Break complex tasks into simple steps
2. **Routing:** Direct inputs to specialized handlers
3. **Parallelization:** Run multiple calls simultaneously
4. **Orchestrator-Workers:** Dynamic task delegation
5. **Evaluator-Optimizer:** Iterative refinement through feedback

These patterns handle 90% of real-world use cases.

Part 4: Building True Agents (Chapters 26-33)

The leap to autonomy:

- The agentic loop (perceive → think → act → repeat)
- State management and memory
- Planning and reasoning
- Error handling and recovery
- Human-in-the-loop patterns
- Guardrails and safety

Part 5: Production Readiness (Chapters 34-41)

Taking agents from prototype to production:

- Testing strategies for non-deterministic systems
- Observability and debugging
- Cost and latency optimization
- Deployment patterns
- Security considerations

Part 6: Capstone Projects (Chapters 42-45)

Apply everything to build real agents:

- Research assistant
- Code analyzer
- Personal productivity agent
- Framework for designing your own

Real-World Agent Examples

Let's ground these concepts with real examples of agents in production:

Example 1: Customer Support Agent

Task: Help customers resolve issues across multiple channels.

Tools available:

- Knowledge base search
- Order lookup
- Refund processing

- Ticket escalation
- Calendar scheduling (for callbacks)

Agent behavior: The agent receives a customer query, searches the knowledge base for relevant information, looks up the customer's order history if needed, determines whether it can resolve the issue or needs to escalate, and either provides a solution or creates a ticket for human review.

Why agent over workflow? Customer issues are unpredictable. One customer might need order tracking, another needs a refund, another has a technical question. The agent adapts to each situation.

Example 2: Code Review Agent

Task: Review pull requests for potential issues.

Tools available:

- File reader (access code files)
- Test runner (execute test suite)
- Linter (check style violations)
- Security scanner (check for vulnerabilities)
- Comment poster (leave feedback on PR)

Agent behavior: The agent reads the changed files, decides which tools to run based on the type of changes (security scanner for auth-related code, performance profiler for database queries), iterates through findings, and posts consolidated feedback.

Why agent over workflow? Different PRs need different reviews. A simple documentation change doesn't need security scanning. A complex refactor might need multiple rounds of analysis.

Example 3: Research Agent

Task: Investigate a topic and produce a comprehensive report.

Tools available:

- Web search
- Document reader
- Note-taking
- Citation manager

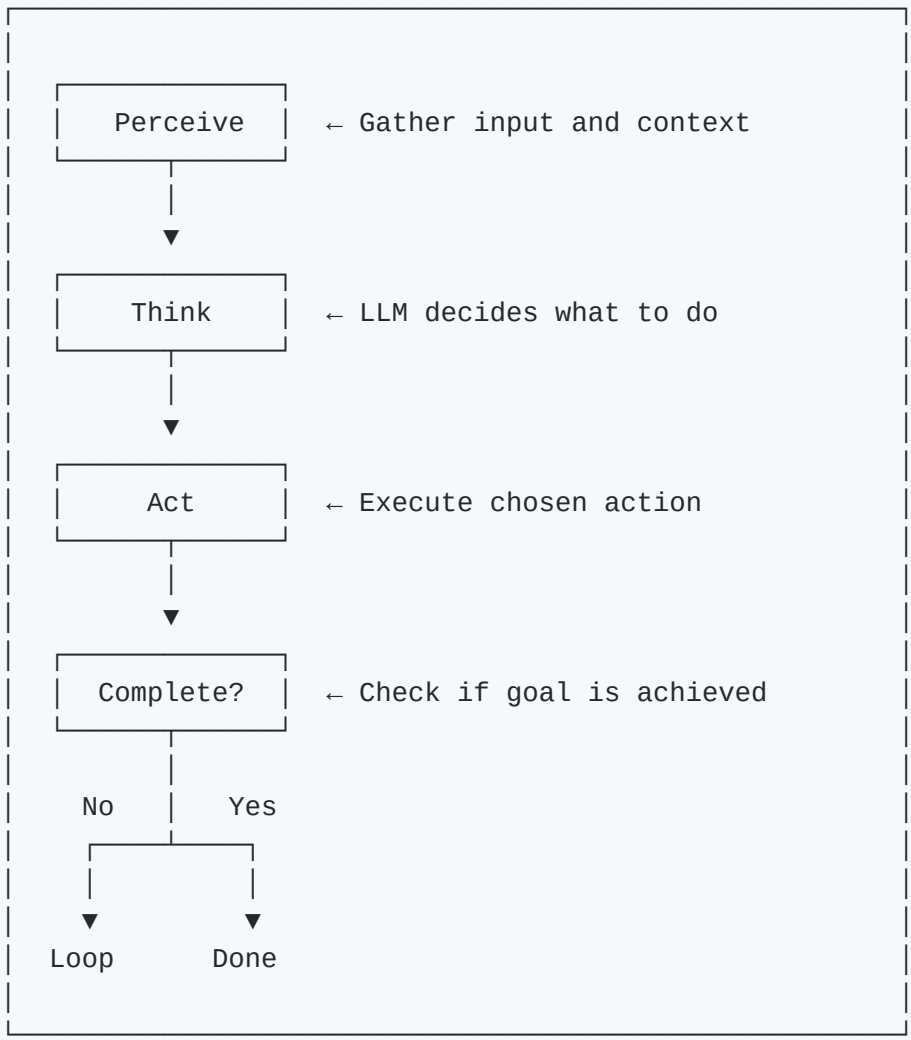
- Report generator

Agent behavior: The agent receives a research question, searches for initial sources, reads promising articles, identifies gaps in understanding, searches for additional sources, synthesizes findings, and generates a structured report with citations.

Why agent over workflow? Research is inherently exploratory. The agent discovers what it doesn't know as it goes and decides how deep to explore each subtopic.

Common Pattern: The Agentic Loop

All these agents share a common structure:



This loop—perceive, think, act, check—is the heart of every agent system. We'll implement it in detail in Part 4.

Common Pitfalls

Before you start building, here are mistakes to avoid:

Pitfall 1: Building an Agent When a Prompt Would Suffice

The mistake: Adding tools, loops, and decision-making when a single, well-crafted prompt would produce the same result.

The fix: Always ask: "Can I get acceptable results with a single LLM call?" If yes, start there. You can always add complexity later.

Pitfall 2: Underestimating the Complexity of Agent Systems

The mistake: Assuming agents are just "LLMs with extra steps." Real agents need error handling, state management, cost controls, timeout limits, and safety guardrails.

The fix: Read Part 4 (Building True Agents) and Part 5 (Production Readiness) before deploying anything. Better yet, build the whole book progressively.

Pitfall 3: Ignoring the Human in the Loop

The mistake: Building fully autonomous agents for tasks where human judgment matters—like sending emails, making purchases, or deleting data.

The fix: Design approval gates for high-stakes actions. Chapter 31 covers human-in-the-loop patterns in detail.

Practical Exercise

Task: Categorize potential applications by approach

For each of the following scenarios, decide whether you'd recommend: (a) a single prompt, (b) an augmented LLM (LLM + tools), (c) a workflow, or (d) an agent. Justify your choice.

Scenarios:

1. Converting a CSV file to JSON format
2. A chatbot that answers questions using a company FAQ database
3. Generating social media posts in five languages from a product announcement
4. A coding assistant that helps users debug Python scripts
5. Automatically categorizing incoming support tickets by department

Hints:

- Consider how predictable the steps are
- Think about whether real-world actions are needed
- Ask if the task requires adaptation based on intermediate results

Solutions:

1. **CSV to JSON:** Single prompt or simple code. No LLM needed—this is pure data transformation.
2. **FAQ chatbot:** Augmented LLM. The LLM needs a tool to search the FAQ database, but the flow is straightforward: receive question → search → respond.
3. **Multi-language social posts:** Workflow (prompt chain). Generate the base post, then run five translation steps in parallel. The steps are known in advance.
4. **Debugging assistant:** Agent. Debugging is exploratory—the assistant might need to run code, read error messages, search documentation, suggest fixes, and iterate based on results.
5. **Ticket categorization:** Single prompt or augmented LLM. If categories are fixed, a single prompt works. If the LLM needs to look up department information, add a tool.

Key Takeaways

- **AI agents are LLMs that dynamically direct their own processes and tool usage.** The LLM decides what to do next, not your code.
- **There's a spectrum from simple prompts to autonomous agents.** Start simple and add complexity only when it provides clear value.
- **Workflows have predetermined steps; agents choose their own path.** Workflows are more predictable; agents are more flexible.
- **Agents excel at open-ended, exploratory tasks** where the steps can't be known in advance. They're overkill for predictable transformations.
- **The agentic loop—perceive, think, act, check—is the core pattern** that underlies all agent systems.
- **Building production agents requires more than just the loop.** You need error handling, state management, cost controls, and safety guardrails.

What's Next

Now that you understand what agents are and when to use them, it's time to set up your development environment. In Chapter 2, you'll install Python 3.10+, set up `uv` (a modern package manager), and create your first project. This foundation ensures all future code works smoothly on your machine.

The journey from concept to running code starts in the next chapter. Let's go!