

Designing Multi-Agent Systems

Principles, Patterns, and Implementation for AI Agents

Victor Dibia

2024-12-31

Table of contents

1	Preface	1
1.1	About This Book	4
1.2	Who This Book Is For	4
1.3	What Makes This Book Different	5
1.4	Prerequisites	5
1.5	Implementation Philosophy	5
1.6	How This Book Is Organized	6
1.6.1	Part I: Foundations of Multi-Agent Systems	6
1.6.2	Part II: Building Multi-Agent Systems from Scratch	7
1.6.3	Part III: Evaluating and Optimizing Multi-Agent Systems	7
1.6.4	Part IV: Real-World Applications	7
1.7	Code Examples and Resources	8
1.8	References and Academic Sources	8
1.9	Acknowledgments	9
I	Part I: Foundations of Multi-Agent Systems	11
2	Understanding Multi-Agent Systems	12
2.1	Introduction	12
2.2	A Primer on Generative AI	16
2.3	Why Multiple Agents? Complex Task Characteristics	18
2.3.1	Planning	20
2.3.2	Diverse Expertise	20
2.3.3	Extensive Context	21
2.3.4	Adaptive Solutions	21

2.3.5	Examples of Complex Tasks	22
2.4	What is an Agent?	23
2.4.1	Core Capabilities	25
2.4.2	How Agents Work	26
2.4.3	Agent vs. Model: A Clear Distinction	28
2.5	What is a Multi-Agent System?	29
2.5.1	Two Approaches to Multi-Agent Orchestration	31
2.6	Why Now?	33
2.6.1	Advances in Generative AI Reasoning Capabilities	33
2.6.2	Economic Value Through Time Arbitrage	33
2.6.3	The Promise of Self-Improving Systems	34
2.6.4	Opportunity to Address Tacit Knowledge Tasks	34
2.6.5	Increasing Demand for Reliable Automation	34
2.6.6	Platform Economics and Integration Value	35
2.6.7	Ethical AI Deployment	35
2.7	Choosing the Right AI Agent Architecture	35
2.7.1	Decision Framework	36
2.7.2	When Multi-Agent Approaches Excel	37
2.7.3	When to Avoid Multi-Agent Approaches	37
2.8	Building Your First Multi-Agent System	38
2.8.1	Environment Setup	38
2.8.2	Step 1: Import and Model Setup	38
2.8.3	Step 2: Create Your First Agent	39
2.8.4	Step 3: Add a Critic Agent	39
2.8.5	Step 4: Round-Robin Orchestration	40
2.9	Summary	42
3	Multi-Agent Patterns	43
3.1	A Taxonomy of Multi-Agent Orchestration Patterns	44
3.2	Workflow Patterns (Explicit Control)	45
3.2.1	Sequential Workflows	45
3.2.2	Conditional Workflows	46
3.2.3	Parallel Workflows	47
3.3	Autonomous Patterns (Emergent Control)	49
3.3.1	Plan-Based Orchestration Pattern	50
3.3.2	Handoff Pattern	51
3.3.3	Conversation-Driven Pattern (Group Chat)	53
3.4	Pattern Selection and Comparison	58
3.4.1	Comparative Analysis	58

3.4.2	Selection Criteria	59
3.4.3	Hybrid Approaches	59
3.5	Task Management Patterns	60
3.5.1	Termination Patterns	60
3.5.2	Human Delegation Patterns	60
3.6	Summary	60
	References	61
4	UX Principles for Multi-Agent Systems	64
4.1	Background: The Evolution of Interfaces and Software Development	66
4.1.1	User Interfaces - From Command-Line to Multimodal, Multi-Agent Interfaces	66
4.1.2	From Software 1.0 (Rules) to Software 3.0+ (Autonomous Multi-Agent Systems)	67
4.2	The Multi-Agent Reliability Challenge	69
4.3	Multi-Agent User Personas and Challenges	70
4.3.1	End User Persona: Jane - Functional Transparency Needs	70
4.3.2	Developer Challenges: Coordination Design	71
4.4	Multi-Agent UX Design Principles	72
4.4.1	Capability Discovery	73
4.4.2	Cost-Aware Action Delegation	74
4.4.3	Observability and Provenance	75
4.4.4	Interruptibility	77
4.5	Interactive vs Offline Multi-Agent Scenarios	78
4.6	Summary	80
II	Part II: Building Multi-Agent Systems from Scratch	82
5	Building Your First Agent	83
5.1	Design Principles	85
5.2	The Agent Execution Loop	86
5.2.1	BaseAgent: The Foundation	87
5.3	Adding a Model Client	89
5.3.1	Agent Model Integration	92
5.4	Enabling Structured Output - The Key to Reliable Agents	93
5.5	Adding Tools	94
5.5.1	Function Calling (Tool Calling) in LLMs	95
5.5.2	The BaseTool Class	95

5.5.3	FunctionTool: Developer Ergonomics	96
5.5.4	Agent Tool Integration	97
5.6	Adding Memory	99
5.6.1	BaseMemory Interface	99
5.6.2	Key Memory Concepts	100
5.6.3	Agent Memory Integration	101
5.7	Adding Guardrails via Agent Middleware	103
5.7.1	How Middleware Works in Practice	103
5.7.2	The BaseMiddleware Interface	104
5.7.3	Real-World Middleware Applications	104
5.7.4	How Agents Use Middleware	105
5.8	Putting It All Together	106
5.8.1	Response 1: Agent Without Memory	106
5.8.2	Response 2: Agent With Memory	107
5.9	Summary	109
6	Building Multi-Agent Workflows	111
6.1	Workflows as Computational Graphs	113
6.2	Steps: Units of Computation	113
6.2.1	The BaseStep Foundation	114
6.2.2	FunctionStep: Converting Functions to Steps	116
6.3	Edges: Transitions and Conditions	117
6.3.1	Edge Fundamentals	118
6.3.2	Conditional Edges	118
6.3.3	Parallel Execution Patterns	119
6.4	Workflow Class: The Container	120
6.4.1	Workflow Construction	120
6.4.2	Dependency Management and Validation	122
6.4.3	Shared State Management	122
6.5	Runner: Execution Engine	122
6.5.1	Execution Engine Capabilities	123
6.5.2	Event-Driven Observability	124
6.6	Putting It All Together	124
6.6.1	Workflow Serialization and Persistence	124
6.7	Summary	125
7	Building Autonomous Multi-Agent Orchestration	126
7.1	The Orchestrator Loop	128
7.1.1	Building the BaseOrchestrator	129

7.1.2	The Universal Run Loop	131
7.2	Building Termination Conditions	133
7.2.1	The BaseTermination Interface	134
7.2.2	Common Termination Patterns	135
7.2.3	Composable Termination Logic	136
7.3	Implementing Round-Robin Orchestration	137
7.3.1	The RoundRobinOrchestrator Implementation	137
7.3.2	Observability and Debugging	139
7.4	AI-Driven Orchestration	140
7.5	Plan-Based Orchestration	142
7.6	Excercises	142
7.7	Summary	143
8	But What About Multi-Agent Frameworks?	144
8.1	Why Consider a Framework at All?	145
8.2	Ten Core Framework Capabilities	146
8.2.1	1. Intuitive Developer Experience with Layered Abstractions	146
8.2.2	2. Async-First Architecture with Streaming Support	147
8.2.3	3. Comprehensive Observability and Development Tools	148
8.2.4	4. Comprehensive State Management and Persistence	149
8.2.5	5. Declarative Configuration and Reproducibility	150
8.2.6	6. Comprehensive Guardrail and Middleware Infrastructure	151
8.2.7	7. Comprehensive Pattern Support and Extensibility	152
8.2.8	8. Production Deployment, Security, and Operations	153
8.2.9	9. Evaluation and Testing Support	154
8.2.10	10. Active Ecosystem and Long-Term Viability	155
8.3	Framework Evaluation Process	155
8.3.1	Step 1: Define Your Requirements	155
8.3.2	Step 2: Create an Evaluation Matrix	156
8.3.3	Step 3: Prototype and Validate	156
8.3.4	Step 4: Make the Decision	157
8.4	Decision Guidelines	157
8.5	Is Picoagents Feature Complete?	158
8.6	Summary	159
III	Part III: Evaluating and Optimizing Multi-Agent Systems	161
9	Evaluating Multi-Agent Systems	162

9.1	Introduction	162
9.2	What We're Evaluating: Multiagent Trajectories	163
9.2.1	Representing Trajectories in Code	164
9.2.2	Observability as Foundation	164
9.3	How We Evaluate: Metrics and Judges	165
9.3.1	Reference-Free vs Reference-Based Metrics	165
9.3.2	LLM-as-a-Judge: A Practical Approach	165
9.3.3	Scoring Strategies	166
9.4	Building Our Evaluation System	166
9.4.1	Evaluation Targets	167
9.4.2	Creating Evaluation Tasks	167
9.4.3	Running Evaluations	168
9.4.4	Understanding Results	168
9.5	Related Research and Datasets	169
9.5.1	The Contamination Challenge	169
9.6	Start with Evaluation	170
9.6.1	1. Define Success Criteria	170
9.6.2	2. Create Your Task Suite	170
9.6.3	3. Choose Your Metrics	170
9.6.4	4. Establish Baselines	171
9.6.5	5. Iterate and Improve	171
9.7	Summary	171
	References	172

IV Part IV: Real-World Applications 175

10	Answering Business Questions from Unstructured Data	176
10.1	The Unstructured Data Challenge	177
10.2	Expressing the Task as a Workflow	178
10.3	Step 1: Data Loading with Intelligent Caching	179
10.4	Step 2: Cost-Effective Pre-Filtering	181
10.5	Step 3: Structured LLM Analysis	184
10.6	Step 4: Analysis and Insight Generation	190
10.7	Testing Each Component Independently	193
10.8	Results and Validation	196
10.9	Current Results	196
10.9.1	Key Insights Visualization	197
10.9.2	Year-over-Year Trends	198

TABLE OF CONTENTS

vii

10.10Applying These Patterns to Business Problems

199

10.11Limitations and Future Optimizations

199

10.12Summary

200

References

202

Chapter 1

Preface

! Early Access Edition

This is an **early access edition** of the book. The core content covering multi-agent fundamentals, implementation from scratch, and evaluation principles is complete and ready for you to learn from. The final sections on optimizing multi-agent systems and real-world applications will be available by **November 10, 2025**.

As an early access edition, expect **rapid updates** and improvements based on reader feedback. While the content has been carefully reviewed, you may encounter minor mistakes or areas for improvement. Your feedback and suggestions are most welcome—please [open an issue on GitHub](#) to report any issues or share ideas. Early access buyers receive all future updates at no additional cost.

Writing this book has been an exploration of one of the fastest-evolving fields in technology, and I'd like to start by sharing my personal experience with AI agents.

In early 2022, the core capabilities of generative AI models began showing incredible promise in solving various text generation tasks out of the box (question answering, summarization, and code generation, for example). It became clear that the potential for creating new types of applications addressing new tasks was immense. At the time, I was working with the Human/AI Experiences (HAX) group at Microsoft Research, partnering with GitHub to improve offline evaluation for an early version of GitHub Copilot — arguably the first example of a modern LLM working in real time at scale to provide large code completions to developers

in an IDE. Our work identified the need for granular metrics (Dibia et al. 2022) to better capture code correctness and proposed improvements to the UX for Copilot. Early results showed that Copilot effectively doubled productivity for developers (Peng et al. (2023))!

Soon afterwards, I created [LIDA](#) (Dibia 2023)—one of the first systems for automatic data visualization using language models—in July 2022, months before ChatGPT’s public release. The system demonstrated how users with no visualization skills could produce high-quality visualizations, and how authoring time could be drastically reduced even for experts. LIDA was implemented as a 4-stage pipeline: a data summarizer, visualization goal generator, visualization code generator, and an optional infographics generator. Ideas from LIDA are now integrated across Microsoft products (Excel, Fabric, Purview, and internal tools). Since then, exploring agents—AI models that not only generate artifacts but can also *act* (call APIs, execute code, etc.)—has become standard practice.

A Note on AI Hype and Reality

In my experience, there are two camps of well-intentioned AI practitioners. The first fixate on *what AI cannot do today*, dismissing it as hype. The second explores what AI can do today, acknowledging limitations while focusing on solving real problems.

When I first showed early LIDA prototypes to researchers, one pointed to a visualization error and declared AI unsuitable for data visualization based on a 20% error rate. A month later, with improved prompts and a new OpenAI model release, errors dropped to 3%—suddenly the reception was entirely different. Meanwhile, product teams who saw the same demos immediately began experimenting and released features that improved alongside model updates.

The lesson: take a pragmatic engineering approach. The best AI systems today combine traditional software engineering practices—proper task decomposition, systematic evaluation, and optimization—with AI capabilities. This book explores both the practical applications available today and the longer-term vision of autonomous AI, giving you the foundation to build effectively in this rapidly evolving landscape.

While creating pipelines encourages reliability (as each step can be independently tested), it also requires effort and assumes that the correct task decomposition is known, the expertise to implement each step is available, and the task is static and predictable. These assumptions often don’t hold true for many complex, real-world tasks that require planning, diverse expertise, or adaptation in dynamic environments. These observations raised an important question: How can we create systems that *independently adapt* to dynamic environments and generalize to solve multiple disparate task types?

At the time, a few colleagues at Microsoft Research were exploring conversation-based multi-

agent systems, which felt like a natural way to extend or generalize the rigid pipeline nature of LIDA. The overall promise was compelling: define discrete agents, give them access to models and tools, and let them *self-organize* through conversation to solve *general* problems. Even better, these systems could improve across multiple task categories simply through *improvements to the underlying models*.

As part of this work, we built a framework around this concept — AutoGen (Wu et al. 2023)— which was released in May 2023. We identified use cases where conversational and iterative reasoning capabilities improved system performance, but we also learned hard reliability lessons. These included agents failing to follow instructions (early models were so polite they would thank each other endlessly across multiple turns!), the importance of planning and careful tool selection for performance (see the Magentic One project (Fourney et al. 2024)), how to build effective developer tooling for these systems (Dibia et al. 2024; Mozannar et al. 2025; Epperson et al. 2025), and the security challenges associated with autonomous agents.

Through this work, I’ve had the privilege of advising dozens of internal Microsoft teams, customers, including Fortune 500 companies, on implementing multi-agent systems in production environments. Additionally, I’ve answered hundreds of questions from startups in the open-source AutoGen community. This experience has given me a unique perspective on what the building blocks of multi-agent systems are, where teams typically struggle, and what practical guidance they need most.

[Fifty thousand GitHub stars later](#), a major [API redesign](#), thousands of open source issues resolved, and multiple research papers published (Dibia et al. (2024)), AutoGen has helped shape the AI agent landscape and refine our collective understanding of multi-agent system design patterns.

The excitement around autonomous multi-agent systems as *one* way (certainly not the only way) to build applications has exploded across the entire industry. Today, there are over a dozen multi-agent frameworks (CrewAI, OpenAI Agents SDK, Google Agent Development Kit, Pydantic AI, and many others), accompanied by a dizzying array of buzzwords — Agents, Multi-agent Systems, Memory, Tools, Context Engineering, Computer Use, Agentic Protocols, and more. Everyone wants to use this technology, but there’s limited clarity on how to do it well.

As the dust settles, clear patterns are emerging that can guide the development of effective multi-agent systems — whether you’re building simple agent interactions or complex orchestrated workflows. This book focuses on identifying these patterns and providing practical guidance for applying them effectively.

1.1 About This Book

This book teaches you to build effective multi-agent systems from first principles, regardless of your chosen tools or frameworks. You'll learn not just *how* these systems work, but *why* they work—including how to translate business problems into multi-agent architectures.

A note on scope: While this book focuses on multi-agent systems, they're not the right solution for every problem. Throughout these pages, you'll develop not just the skills to build sophisticated agent systems, but the judgment to recognize when simpler approaches serve you better.

This book covers:

- **Multi-Agent Fundamentals:** Core concepts, design patterns, and user experience principles
- **Implementation from Scratch:** Building agents, workflows, and orchestration by creating a complete Python library called `picoagents`
- **Evaluation and Optimization:** Testing, measuring performance, and optimizing for reliability and scale
- **Production Deployment:** Security, ethics, and responsible AI practices for real-world applications
- **Domain Applications:** Complete implementations for information processing, data analysis, and software engineering

By the end, you'll know how to choose the right multi-agent architecture for any task and build your own systems from scratch (should you choose to do so).

1.2 Who This Book Is For

This book is designed for **technical practitioners** building AI-powered systems—whether you're just getting started with agents or implementing them in production.

Primary audience: - **System architects and software engineers** designing AI applications with multiple intelligent components - **Technical leaders** making architectural decisions about multi-agent implementations
- **AI engineers** transitioning from individual LLM models to orchestrated agent systems - **Product managers** needing to understand multi-agent capabilities and trade-offs

You'll get the most value if you: - Want to understand **fundamental concepts** rather than just framework-specific tutorials - Are interested in **implementation details from scratch**—critical for evaluating tools and making architectural decisions - Need **practical**,

worked examples you can adapt to real-world problems

Building effective multi-agent systems is roughly **40% theory** (understanding models, evaluation, translating business problems) and **60% systems engineering** (architecture, code, deployment, scaling). This book reflects that balance, providing both conceptual foundations and hands-on implementation guidance.

1.3 What Makes This Book Different

This book takes a unique approach compared to other multi-agent system resources:

- **Thorough treatment of concepts and theory**—Part I provides comprehensive foundations that executives, designers, and technical teams all need to understand multi-agent systems
- **Building production-ready systems from scratch**—You’ll build agents and multi-agent orchestration patterns from first principles, so you understand every component and design decision
- **Focus on evaluation and optimization**—Moving beyond simply applying LLMs to building reliable systems requires evaluation skills often missing from traditional software engineering curricula
- **Framework and technology agnostic**—All examples use Python and are built from scratch, but can be reimplemented in any language or framework of your choice
- **Complete end-to-end approach**—The book covers theory (Part I), implementation (Part II), optimization (Part III), and real-world examples (Part IV), with reusable sample code throughout

1.4 Prerequisites

To get the most out of this book, you should have:

- Basic Python programming experience
- Understanding of machine learning fundamentals (helpful but not required)
- Experience with command-line tools

1.5 Implementation Philosophy

This book takes a **fundamentals-first approach**, guiding you through implementing core multi-agent concepts from scratch. As part of this, we will build a multi-agent

library—picoagents—from scratch and use it to implement a set of complex use cases. You’ll learn to build:

- **Individual agents** and how to configure them with AI models, tools, and memory components to solve tasks
- **Multi-Agent coordination** through both deterministic workflows and autonomous orchestration patterns
- **Evaluation and optimization libraries** for measuring and improving system performance
- **End-user integrations** that seamlessly embed multi-agent capabilities into applications complete with the *right* user experience

By implementing these fundamental building blocks yourself, you’ll gain deep understanding of the design decisions and trade-offs involved in multi-agent systems. This knowledge transcends any specific framework, equips you with the basics on how/when to use a framework, and enables you to architect systems that meet your exact requirements.

Framework Examples: Where relevant, I’ll also demonstrate how similar concepts can be implemented using established frameworks like [AutoGen](#), Google ADK, Pydantic AI, and other popular tools. This helps bridge the gap between fundamental understanding and practical development with existing ecosystems.

1.6 How This Book Is Organized

This book is divided into four parts that systematically build your expertise in multi-agent systems, following a **theory -> build -> optimize -> apply** progression:

1.6.1 Part I: Foundations of Multi-Agent Systems

Part I establishes the theoretical foundation you need to understand multi-agent systems. This part is purely conceptual, focusing on core principles without getting into implementation details:

- **Chapter 1:** Understanding multi-agent systems—what they are, why they matter, and when to use them
- **Chapter 2:** Multi-agent patterns—theoretical frameworks and architectural patterns for agent collaboration
- **Chapter 3:** User experience of multi-agent systems—design principles and interface considerations

This section provides the conceptual grounding needed before diving into hands-on development.

1.6.2 Part II: Building Multi-Agent Systems from Scratch

Part II takes you from theory to practice, teaching you to build multi-agent systems from first principles. This hands-on section is framework-agnostic, focusing on core implementation concepts:

- **Chapter 4:** Building your first agent—fundamental agent creation and basic interactions
- **Chapter 5:** Building multi-agent workflows—coordinating multiple agents to complete complex tasks
- **Chapter 6:** Autonomous multi-agent orchestration—advanced patterns for self-organizing agent teams
- **Chapter 7:** Interface agents—agents that accomplish tasks by driving interfaces (e.g., a browser or desktop UI)

By building agents from scratch, you'll understand every component and design decision.

1.6.3 Part III: Evaluating and Optimizing Multi-Agent Systems

Part III addresses the critical challenges of making multi-agent systems work reliably in practice:

- **Chapter 8:** Evaluating multi-agent systems—testing, benchmarks, and measuring success
- **Chapter 9:** Optimizing multi-agent systems—performance, scalability, and efficiency improvements
- **Chapter 10:** Open challenges—handling security, controllability, and edge cases
- **Chapter 11:** Security, Ethics and responsible AI—implementing guardrails and human oversight

You'll learn to address the questions: How do we ensure our systems work correctly? How do we make them better?

1.6.4 Part IV: Real-World Applications

The final part brings everything together through 3 comprehensive real-world case studies:

- **Chapter 12:** Information processing—building agents that can gather, analyze, and synthesize information from multiple sources

- **Chapter 13:** Data analysis agent—creating intelligent systems for data exploration, visualization, and insights generation
- **Chapter 14:** Software engineering agent—developing agents that can assist with code generation, testing, and development workflows


Complete implementations, deployment strategies, and lessons learned from real-world applications demonstrate how theory translates into practice.

1.7 Code Examples and Resources

All code examples in this book are available in the companion GitHub repository: <https://github.com/victordibia/designing-multiagent-systems>

The repository includes: - Complete source code for all examples - Additional exercises and challenges - Community discussions and updates

Working Code References

Throughout this book, you'll see special **Working Code** callouts like this one that point to complete, runnable examples in the companion repository. These examples demonstrate the concepts in action and provide a foundation for your own implementations. Look for the  icon to find these practical code references.

1.8 References and Academic Sources

The ideas in this book build upon extensive research and development work in multi-agent systems. Throughout the book, you'll find citations to relevant research papers, and many concepts have been explored in blog posts, conference presentations, and open-source projects. Where possible, original sources are cited—consider exploring these materials for deeper theoretical understanding and alternative perspectives.

Stay Connected

The AI agent space is evolving rapidly - and this book is written with that in mind. I plan to have major quarterly releases which will be available as a digital edition on the book website. Also consider following the book GitHub page for code and other updates.

- **Book Website:** multiagentbook.com. The book website also contains a **labs** section with useful resources such as an implementation of [multi-agent usecases](#) using multiple frameworks.
- **GitHub:** [Book Code Repository](#)
- **Author:** [Victor Dibia](#)

Note: If you find any issues or have suggestions, please open an issue in the [book's GitHub repository](#).

1.9 Acknowledgments

Writing this book has been shaped by countless conversations, collaborations, and insights from colleagues, friends, and the broader AI community. This work stands on the shoulders of many researchers and practitioners whose contributions are cited and acknowledged throughout these pages. I am deeply grateful for their insights and generosity in sharing knowledge.

My understanding of Human-AI interaction, AI agents, and multi-agent orchestration patterns has also been profoundly shaped by conversations with members of the Human/AI Experiences (HAX) group at Microsoft Research: Saleema Amershi, Adam Fourney, Gagan Bansal, Jingya Chen and Hussein Mozannar—thank you for the many collaborations! I am equally grateful to the leaders of the AI Frontiers lab at MSR, Ece Kamar and Ahmed Awadallah, who have been incredible mentors and encouraged me to write this book.

Special thanks to Gonzalo Ramos, Steven Druker, Dan Marsal, Dave Brown, and Nathalie Riche of the Visualization Group at Microsoft Research, with whom I shared early excitement and collaborated on generative AI for data visualization ideas. Thank you to the many reviewers who took the time to provide feedback on early drafts of the book including Piali Choudhury, Sasa Junosovic, and many others.

I am deeply grateful to the AutoGen community — from Chi Wang and Qingyun Wu who started the AutoGen project, to Jack Gerrits and Eric Zhu, who stepped up to help lead it, brought engineering rigor, and helped make it a success. Thank you to all the contributors, early adopters, and community members who have shared ideas, reported issues, provided feedback, and contributed code. The broader open-source AI community has been instrumental in shaping the ideas in this book through countless discussions, thousands of issues on GitHub and numerous community calls. Thank you!

Finally, I am grateful to my family. This book is as much yours as it is mine — to my wife

Hermine for all the support and brilliant feedback along the way, and to my little boy Liam whose hugs provided the energy I needed when things got tough. Thank you!

Victor Dibia

Part I

Part I: Foundations of Multi-Agent Systems

Chapter 2

Understanding Multi-Agent Systems

This chapter covers:

- The complexity challenge: understanding when individual AI models become insufficient and why agents are needed
- What defines an agent: core capabilities (reason, act, communicate, adapt) and essential components (model, tools, memory)
- Complex tasks that motivate multi-agent systems: planning, diverse expertise, extensive context, and adaptive solutions
- When to use (and when not to use) multi-agent approaches: Multi-agent systems are not always the *right* solution; know when to explore or use them.

2.1 Introduction

Generative AI (GenAI) models today have demonstrated remarkable abilities in modeling complex relationships within the vast amount of data on which they are trained. For example, models such as GPT-4, Claude, Gemini, etc., also known as large language models (LLMs) (Vaswani et al. 2017), excel at text processing tasks (for example, summarizing passages, extracting entities, generating code, etc.). Given the capabilities of these models, they have now been applied directly within applications to solve tasks in various industries. For instance, LLMs have been integrated into products for generating marketing copy, reviewing legal

documents for compliance, providing advanced natural language understanding to chatbots and virtual assistants, and assisting researchers in the scientific discovery process. And adoption is growing rapidly across both individual users and enterprises. A recent report states that 75% of knowledge workers are already using AI in the workplace today (Microsoft AI Blog 2024), and a rising number of startups are building applications that leverage these models to solve real-world problems.

i Quantifying the Shift to AI Agents

It's widely acknowledged that interest in AI agents is growing rapidly, but by how much? To answer this question with concrete data, Chapter 10 demonstrates how to build a multi-agent workflow that analyzes startup data from Y Combinator—one of North America's largest startup incubators.

As a preview of both the methodology and results: by analyzing descriptions from 5,622 Y Combinator companies, we found that startups building AI agents grew from 6.1% in 2020 to 47.7% in 2025—a 7.8 -fold increase in just five years. This dramatic growth signals where the industry sees opportunity: moving beyond AI-assisted tools to fully autonomous systems that can handle complex, multi-step tasks.

This analysis itself exemplifies the multi-agent workflow patterns we'll explore in Chapter 3, with the complete implementation detailed in Chapter 10. Feel free to jump ahead if you're curious about the methodology!

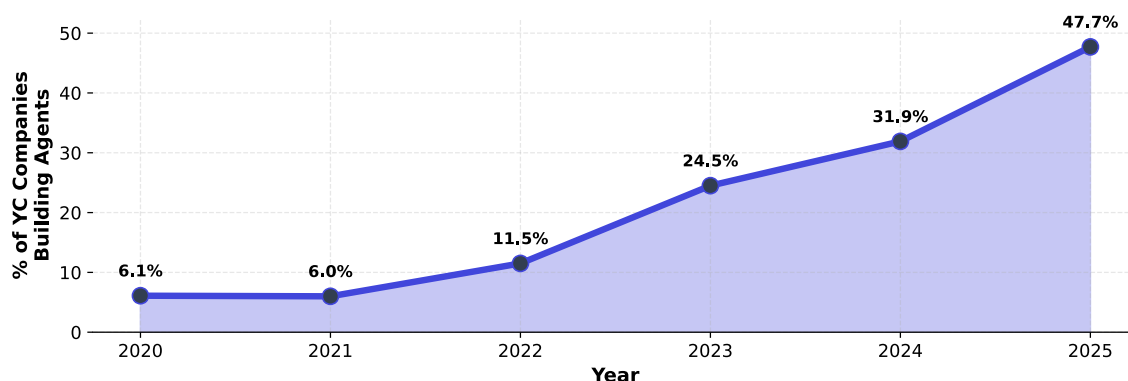


Figure 2.1: Growth in Y Combinator startups building AI agents (2020-2025). This dramatic trend demonstrates the industry shift toward autonomous AI systems.

However, as tasks become more *complex* — those that are long-running, involve multiple steps, require consideration of various perspectives, and necessitate exploration of a dynamic solution space — these models alone are insufficient. To illustrate this progression and motivate the need for different types of AI systems, consider the following task examples (shown in Figure 2.2), representing three distinct levels of complexity:

Complexity Level	Task Example	Description
Model-Level	“What is the height of the Eiffel Tower?”	Direct information retrieval from training data
Agent-Level	“Tell me the stock price of NVIDIA today.”	Requires current data, planning, and tool use
Multi-Agent	“Build a mobile application that helps users view stock prices, buy stock and file taxes.”	Involves multiple domains of expertise and iterative development

i Definitions: Agents, Multi-Agent Systems, and Tools

Agent: An AI system that can reason, act with tools, communicate, and adapt
 Multi-Agent System: Multiple agents working together, each with specialized capabilities
 Tools: External capabilities like APIs, code execution, web search that extend agent abilities

Task 1 can be reliably addressed by retrieving general knowledge facts from a model’s training data or existing systems such as web search engines. This represents what a **model** can accomplish - straightforward information retrieval without the need for planning, tools, or adaptation.

Task 2, however, reveals where models alone become insufficient. The stock price of NVIDIA **today** represents information that an LLM is unlikely to have seen during training; thus, the results it generates are likely to be incorrect (hallucination). Task 2 requires understanding the query, *planning* (e.g., decomposing the task into steps such as fetching current stock data and analyzing the data to answer the question), *action* (executing each step using tools like web search or APIs), and presentation of final results. This is where we need an **agent** - a system that combines the reasoning capabilities of these models with tools that enable them to act on our behalf. Current GenAI applications explore this approach by building prescribed

solution pipelines that include Large Language Models (LLMs) which drive tools for action.

Task 3 represents a level of complexity that goes beyond what even a capable single agent can handle effectively. It involves multiple types of expertise (interface design, Android development, API usage, integration testing, etc.) and requires an iterative solution where actions are taken and resulting outcomes determine the next steps toward a solution (e.g., retrieving appropriate SDK documentation, writing multiple versions of each module in the app, testing them to identify errors, repairing and integrating until a working application is crafted). This is where **multiple agents** become essential - each bringing specialized capabilities while collaborating to solve tasks that exceed any single system's scope.

In this book, we will focus on exploring how to address these complex tasks through the development of applications that implement elements of planning, action and communication across *multiple agents*. In turn, these applications hold potential to enable radically new, and general-purpose digital interfaces, solve new classes of tasks or address existing tasks in a manner that reduces human toil and effort.

The potential extends beyond generic task automation. When agents understand user context (current activity, environment, recent interactions) and preferences (explicitly defined settings and behavioral patterns), they can provide highly personalized assistance with tasks such as scheduling meetings, drafting email responses, booking flights, making e-commerce purchases, and filing taxes.

To realize this potential, we need to understand the fundamental building blocks: What exactly makes a system an “agent”? How do we build systems where multiple agents work together effectively? Let's start by examining these core concepts.

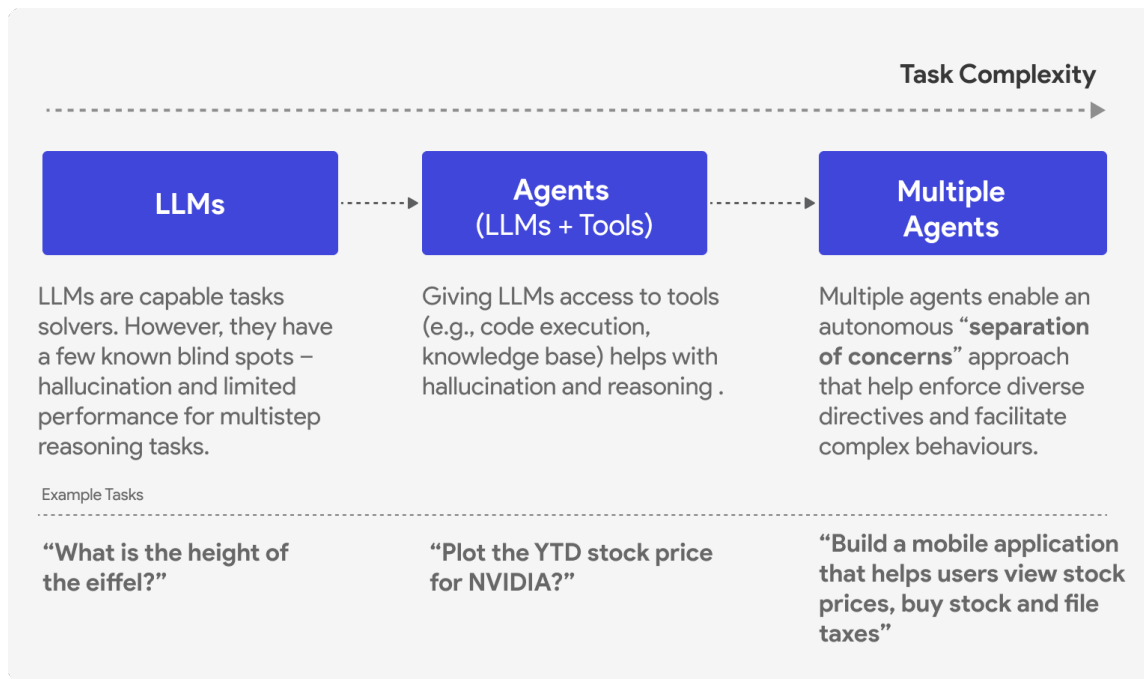


Figure 2.2: Generative AI models (e.g. Large Language Models) can address a wide range of tasks including answering general knowledge questions, translating text, summarizing passages, generating code, etc. However, as tasks become more complex, requiring reasoning, planning, and action, the limitations of these models become apparent. We can address these increasingly complex tasks by providing LLMs with access to tools that enable them to act as agents on our behalf and enabling collaboration across multiple agents.

2.2 A Primer on Generative AI

Generative models are a subset of deep neural networks skilled at *identifying complex patterns within datasets, enabling them to generate new, similar data points*. These models are distinct from discriminative models, which focus on differentiating between data types or classifying them. Generative models can be trained on data from various modalities. For instance, models trained on written text in human languages, such as the GPT series, are known as Large Language Models (LLMs); those trained on images, like the DALL-E series, are termed Image Generation Models (IGMs); and models trained on both, such as GPT4-V, are often referred to as Large Multimodal Models (LMMs). This book will concentrate on Generative Models that generate text output, including both large language models and large multimodal

models.

The fundamental concept for training generative models is relatively simple. In the case of LLMs, these models are trained to predict the next word in a sequence or to fill in blanks for masked or hidden sections of text - aka *sequence prediction*. By applying this training objective to a sufficiently large dataset, the models learn representations of the world as depicted through text. In turn, the models can leverage these representations in generating text that is coherent, relevant, and contextually appropriate.

To *apply* these models to solve tasks, early efforts focused on framing multiple tasks as sequence prediction problems. For example, to classify text, a description of the classification problem is provided, followed by a list of labels and a prompt asking, “Which of the provided labels is the correct class?”. Similarly, to summarize a given passage, given a prompt structured as “The summary of the passage above is ..”, the model is induced to generate a summary of the passage. It turns out that framing tasks as sequence prediction problems allows the models to leverage knowledge gained during training to perform a wide range of tasks (e.g., language translation, sentiment analysis, question answering, dialogue generation, named entity recognition, syntax parsing, code synthesis, paraphrasing, grammar correction etc.) beyond their initial training objectives. This background is important to understand the capabilities and limitations of these models when applied to complex tasks. For example, the *reasoning* capabilities observed in these models (e.g. $2 \text{ apples} + 10 \text{ apples} = 12 \text{ apples}$) are often limited to scenarios well represented in training data, with failures to inherently generalize to rare or unseen scenarios (e.g. solving linear equations in base 3).

The practice of creating sequences that increase the likelihood of successfully completing a task is known as *prompt engineering*. Techniques such as few-shot prompting (Brown et al. 2020), where examples of the task and solution are included in the sequence, chain-of-thought prompting (Wei et al. 2022), where examples of tasks broken down into solution steps are included, and ReAct prompting (Yao et al. 2022), which combines reasoning and acting in language models, have become leading prompt engineering approaches. Beyond prompt engineering, fine-tuning models on task-specific instruction data and explicitly collected human feedback data (Ouyang et al. 2022) have proven to enhance model alignment - efforts aimed at ensuring that generated outputs align with human intentions, preferences, and values.

Finally, while LLMs can be instructed to generate text sequences that solve tasks, the amount of text they can process at a time is fixed, constrained by what is known as the *context window*. The context window size defines the maximum number of tokens (words or word pieces) the model can handle in a single input and output sequence. This limit is dictated by the model’s architecture, particularly its computational and memory constraints. Understanding the context window is crucial for developers as it affects how input data should be structured

and how tasks should be framed. Strategies like truncation, sliding window techniques, summarization, chunking, and optimized prompt engineering are essential to operate effectively within these constraints, maximizing the model’s potential in various applications while balancing computational resources.

2.3 Why Multiple Agents? Complex Task Characteristics

While a single agent can effectively handle tasks like retrieving current stock prices, even capable agents encounter fundamental limitations when facing highly complex challenges. Consider our Task 3 from Figure 2.2: “Build a mobile application that helps users view stock prices, buy stock and file taxes.” A single agent attempting this task would need to understand application requirements across multiple domains, design user interfaces and user experiences, write Android development code, integrate stock market APIs, implement tax filing functionality, and handle testing, debugging, and deployment.

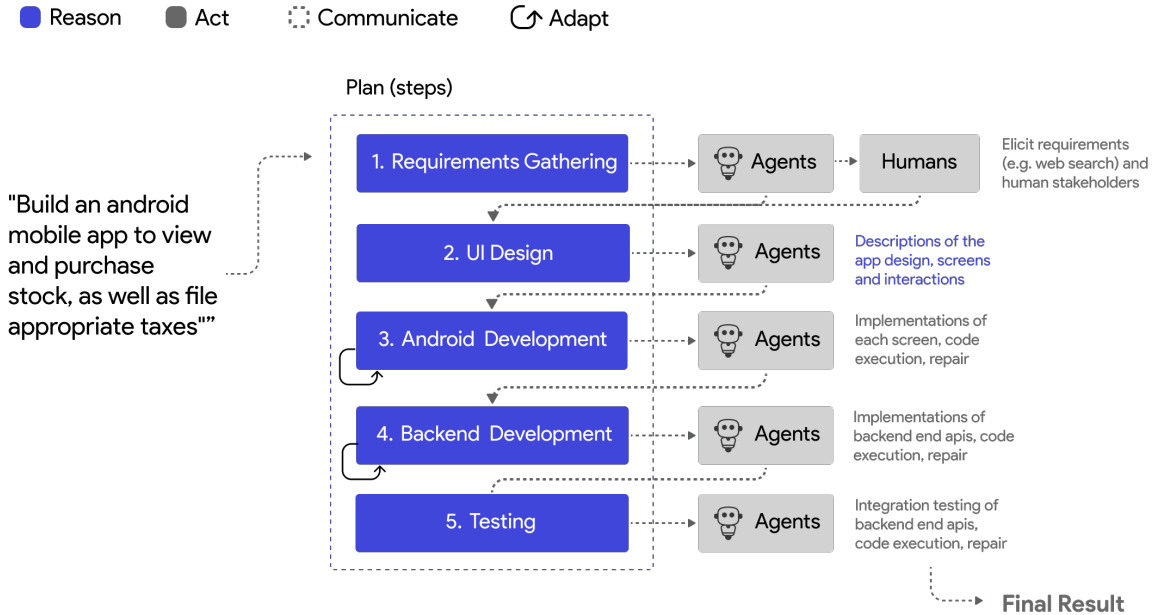


Figure 2.3: Consider the following complex task - “build an app to view and purchase stock, as well as file appropriate taxes”. To address such a task, a multi-agent application must derive a plan (e.g., gather requirements, design an interface, write android code etc), optionally delegate steps in the plan to various entities with specialized relevant capabilities (agents and humans), communicate effectively and adapt (recover from errors, explore new approaches) to solve the task.

Even with access to powerful models and comprehensive tools, a single agent faces several challenges. The extensive context required quickly exceeds practical limits - existing studies (Liu et al. 2024) have shown that while LLMs can process increasingly longer text, they are likely to attend to instructions at the beginning and end of the text, while neglecting data in the middle. More fundamentally, this task demands diverse types of expertise that are difficult to encode effectively in a single agent’s instructions.

Complex tasks that motivate multi-agent approaches typically exhibit four key characteristics, as shown in Figure 2.4. While the presence of any one of these characteristics can increase the complexity of a task, it is often their combination that presents the greatest opportunity for multi-agent systems.

What makes a task complex?

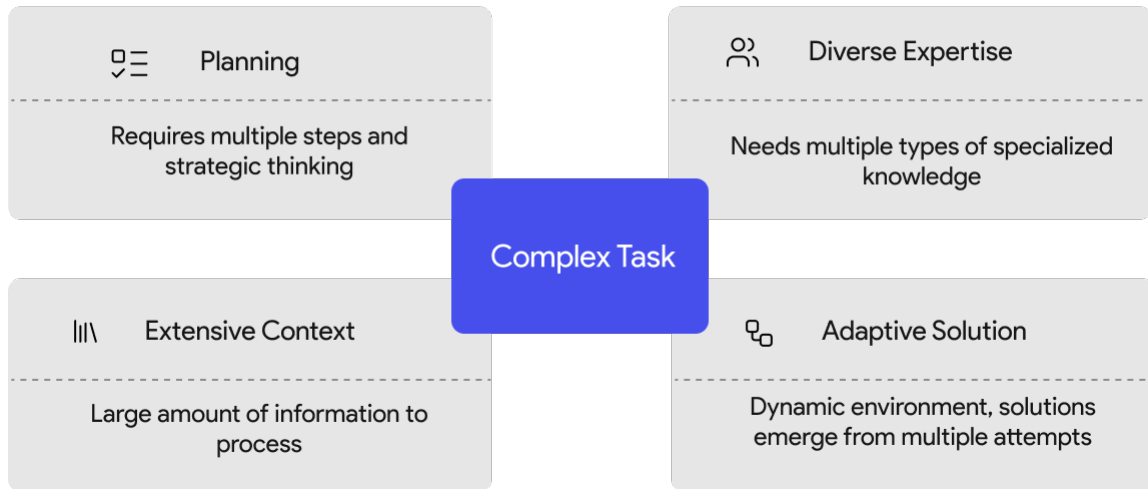


Figure 2.4: Complex tasks exhibit four key characteristics that make them suitable for multi-agent systems: Planning (requiring multiple steps and strategic thinking), Diverse Expertise (needing specialized knowledge across different domains), Extensive Context (processing large amounts of information), and Adaptive Solutions (where solutions emerge from multiple attempts). Tasks become increasingly complex when they combine multiple characteristics, making them ideal candidates for multi-agent approaches.

2.3.1 Planning

Complex tasks often require a high level plan, which involves decomposing the task into steps that must be completed successfully. Given some context, a plan prescribes a set of actions that should be executed to achieve some target success state. This is a well-known and studied property in the robotics planning literature and we will borrow concepts from this domain across this book.

2.3.2 Diverse Expertise

Decomposing complex tasks into multiple steps often results in steps that can benefit from specific expertise. For instance, consider the app development task where the objective is to build a mobile app for viewing and purchasing stocks and filing taxes. This task may be broken down into steps such as analyzing app requirements, designing the user interface,

implementing required functionalities, integrating necessary APIs for stock data and tax filing, and testing and deploying the app. In turn, these tasks map into specific roles that can be addressed by dedicated agents (e.g. a UX agent, Android App agent etc), each possessing specialized knowledge and skills.

The separation of expertise allows for a separation-of-concerns approach that single-agent systems may struggle to achieve following a generic instruction set. From the application development perspective, creating agents with specific directives provides a useful abstraction for cleanly mapping responsibilities to specific entities, enabling domain-driven design of applications.

2.3.3 Extensive Context

Complex tasks often require extensive context that need to be processed to solve the task. For example, in our app development task, the agents may conduct web search queries as well as ask for human feedback to assemble requirements, across multiple turns to assemble the right initial context. Such lengthy instructions or context present a significant challenge for single-agent systems. Existing studies (Liu et al. 2024) have shown that while LLMs can process increasingly longer text, they are likely to attend to instructions at the beginning and end of the text, while neglecting data in the middle. This perspective is also informed by theories from cognitive science, such as cognitive load theory (Sweller 1988), which suggests that the human brain has a limited working memory capacity. When presented with lengthy or underspecified instructions, this capacity becomes overwhelmed, leading to decreased comprehension and performance.

A multi-agent perspective can help address this limitation by selective context provisioning - i.e., only sections of available context (for example, a history of actions performed so far) relevant to a task are provided to an agent focused on that task. By structuring instructions and managing context in a way that minimizes unnecessary cognitive load, we can significantly improve task performance in AI systems.

2.3.4 Adaptive Solutions

Complex tasks are often situated in dynamic environments where the exact solution is unknown or uncertain until some actions are taken. For example, specific data sources may be unavailable at a given time requiring the exploration of other data sources or each action may have unexpected side effects etc. For example, in the app development task, the initial request to the API may fail (e.g., due to a network issue, incorrect argument format or general changes to the API), requiring the agent to adapt by retrying the request with different parameters based on the *feedback* (e.g., error message received from the API request

or seeking explicit human assistance). In these cases, the solution emerges from *adaptation* - iteratively reasoning across some initial plan, actions and outcomes of agents as they solve the task.

This property is also inspired by insights from metacognition studies, which suggest that an agent’s awareness and regulation of their own cognitive processes can significantly influence the problem-solving process. Metacognitive skills, which are challenging for single-agent systems today, enable agents to reflect on, evaluate, and adjust their strategies in real-time, leading to more adaptive and potentially innovative solutions within the emergent framework of multi-agent collaboration.

Given these adaptive requirements, it becomes more challenging to write deterministic pipelines (e.g., specific prompts, tools), hence the need for agents that can self-orchestrate (with some autonomy) to address the task.

2.3.5 Examples of Complex Tasks

These characteristics aren’t just theoretical—the 47.7% of Y Combinator startups building AI agents in 2025 are tackling exactly these types of complex challenges, with concentrations in productivity (25.5%), software (18.4%), finance (15.6%), health (11.6%), and e commerce (6.8%) where planning and diverse expertise are essential. See the in-depth analysis in Chapter 10.

Table 2.2 shows some examples of complex tasks and how they map to each of these properties.

Table 2.2: Examples of complex tasks and their defining characteristics: Planning, Diverse Expertise, Extensive Context, and Adaptive Solutions.

Task	Planning	Diverse Expertise	Extensive Context	Adaptive Solutions
Web App Development	Requirements, design, compliance, APIs	Developers, designers, security, data experts	Legacy systems, docs, workflows, standards	Security updates, feedback, changes
Financial Reporting	Data collection, analysis, reporting	Analysts, statisticians, writers, domain experts	Market data, tools, standards, trends	New sources, feedback, market shifts

Task	Planning	Diverse Expertise	Extensive Context	Adaptive Solutions
Tax Filing	Data gathering, analysis, filing, compliance	Tax advisors, analysts, legal, jurisdiction experts	Records, tax codes, regulations, structures	Law changes, queries, corrections
Presentation Design	Content review, design, interactivity	Designers, editors, presentation experts	Content, audience, goals, guidelines	Feedback, content updates, format changes

Reflecting on these patterns provides context for understanding *when* and *why* a multi-agent approach is beneficial. When tasks are complex, dynamic, and require diverse expertise, a multi-agent approach can offer the adaptability and collaborative problem-solving capabilities necessary to manage these tasks effectively. Recent research studies have also highlighted benefits of multi-agent approaches. For instance, (Du et al. 2023) find that a *society of mind* approach (Minsky 1986) - a theory proposing that human intelligence arises from the interaction of multiple simple agents working together - improves reasoning and factuality when multiple agents debate outcomes over multiple rounds of conversation turns. Similarly, (Liang et al. 2023) demonstrate that using multiple agents with separate roles (e.g., agents that perform a task and agents that adjudicate the quality of the task) can improve the diversity and quality of generated outcomes.

2.4 What is an Agent?

Definitions of agents vary across the AI literature, from early concepts in robotics to more recent ideas around entities driven by AI models. A good starting point is the *classic* work of Russell and Norvig (Russell and Norvig 2020) that define an agent as anything that perceives its environment through sensors and acts upon that environment through actuators, operating through an agent function that maps perceptions to actions. This classical definition emphasizes perception-action *loops* driven by logical reasoning.

To illustrate this classical approach, consider a robotic vacuum cleaner: it perceives its environment through sensors (detecting obstacles, dirt levels, battery status), applies logical rules (if obstacle detected, turn; if dirt detected, increase suction; if battery low, return to dock), and acts through actuators (motors for movement, suction mechanisms, brush rotation). The agent function follows predetermined logic: obstacle -> avoid, dirt -> clean,

low battery -> recharge. This creates a continuous perception-action loop where each sensor reading triggers specific programmed responses.

We can build on this foundation but extend it for the generative AI era. While classical agents operate through predefined functions, generative AI agents maintain the same perception-action cycle but add sophisticated capabilities: complex reasoning powered by large language models, dynamic tool use, natural language communication, and adaptive behavior based on outcomes. These enhanced capabilities make them particularly suitable for the collaborative problem-solving we explore in this book.

For this book, we define agents as entities that can reason, act, communicate, and adapt to solve problems. Consider our NVIDIA stock price example (Task 2 from Figure 2.2): an effective agent must understand the information request, reason about obtaining current stock data, take action using appropriate tools (web search or financial APIs), and adapt if the initial approach fails. This adaptation might involve trying alternative data sources or adjusting the query strategy.

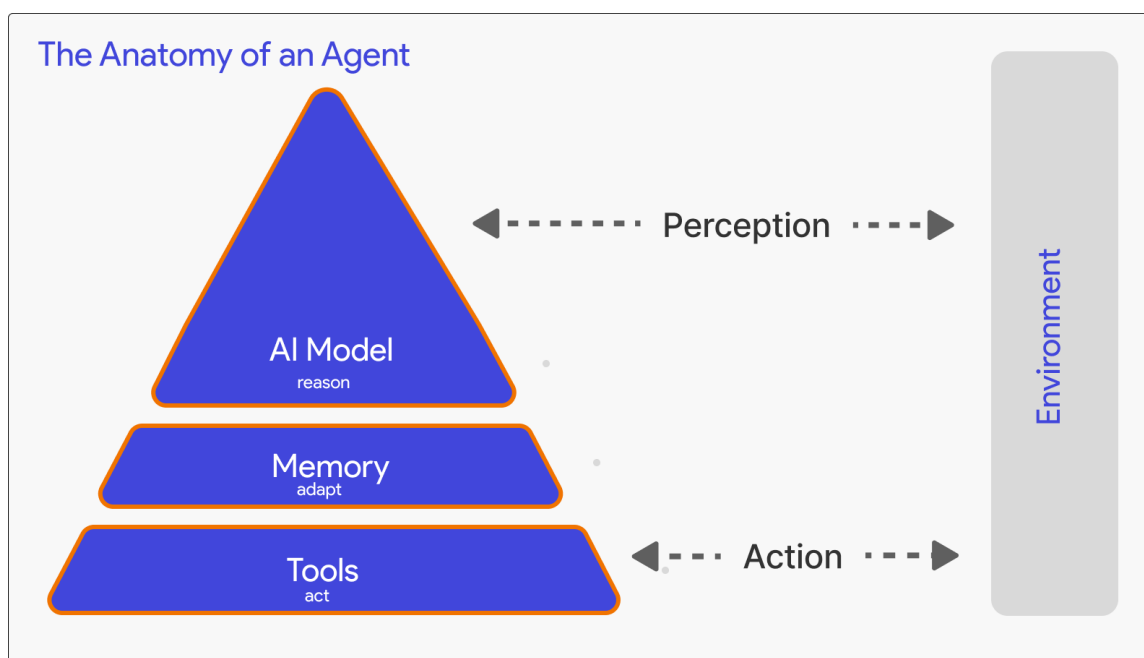


Figure 2.5: An agent is a software entity that possesses core components - a generative AI model that enables reasoning, tools that enable the agent to act, and memory that enables the agent to recall and reuse information.

2.4.1 Core Capabilities

An agent possesses four fundamental capabilities that distinguish it from basic models or traditional programs:

Reason: Agents can synthesize new information by applying rules or logic to available context. This reasoning may be deductive, inductive, or abductive and can be driven by a Generative AI model, custom processing functions, or a combination of both. In our stock price example, reasoning involves understanding that “NVIDIA today” requires current market data, not historical information from training data.

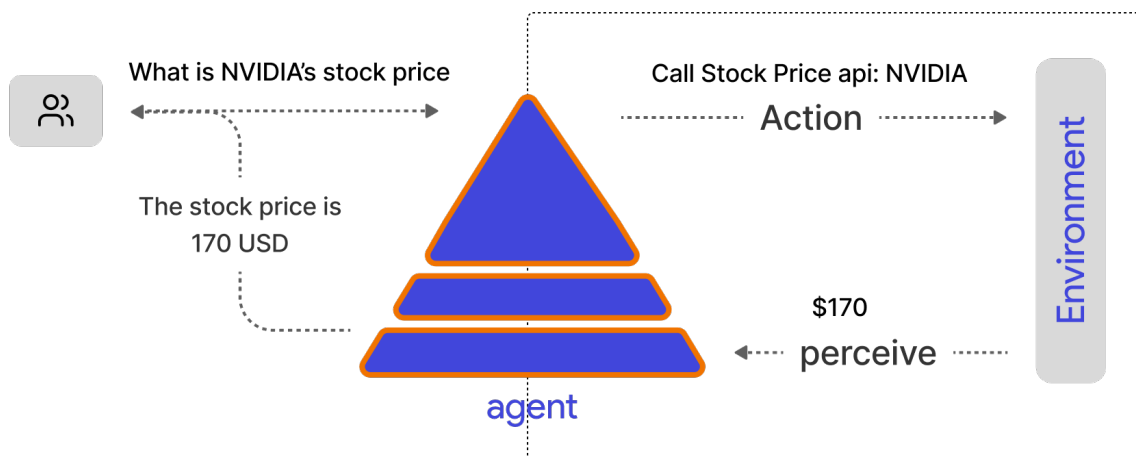
Act: Agents can take concrete actions to affect their environment or gather information. This goes beyond generating text responses - agents can execute code, call APIs, search the web, or interact with external systems. For the stock price task, acting means actually calling a financial data API or performing a web search to retrieve current pricing information.

Communicate: Agents can effectively exchange information with users, other agents, and external systems. This includes understanding natural language inputs, formatting appropriate responses, and knowing when and how to request additional information or clarification.

Adapt: Agents can modify their approach based on feedback, changing conditions, or new information. If an initial API call fails due to rate limiting, an effective agent might wait and retry, switch to an alternative data source, or adjust its approach based on the error message received.

2.4.2 How Agents Work

Agent Action Perception Loop



Agents take **action**, and then **perceive** the results over an iterative loop. The response may be an error that the agent has to handle with a **retry** or error response to the user.

Figure 2.6: The agent action-perception loop: Agents operate through iterative cycles where they take action (such as calling an API), perceive the results (processing the response), and adapt their approach based on outcomes. When successful, agents provide natural language responses; when errors occur, they may retry with different parameters or inform users of limitations.

Agents operate through a fundamental **action-perception loop** shown in Figure 2.6. Unlike models that generate single responses, agents work through iterative cycles—taking action, perceiving results, and adapting based on outcomes until the task is resolved. Multi-agent systems build upon this principle, with multiple agents running coordinated action-perception cycles to solve complex tasks that exceed any single agent's capabilities.

These capabilities are enabled by the three core components shown in Figure 2.5 working together:

2.4.2.1 Model

The reasoning engine that enables decision-making and planning. This is typically a large language model (LLM) or large multimodal model (LMM) that can understand context, generate plans, and determine appropriate actions. The model serves as the “brain” of the agent, processing inputs and deciding what to do next.

For agents to successfully accomplish tasks, they must reason over the task state and determine the appropriate actions to take. This core decision-making capability can be driven by generative AI models, predefined logic, or explicit human input. Besides generative AI models, agents can also make use of custom logic (e.g., logic to call an API whenever a message is received based on some heuristics), or explicit human input to drive their actions and reasoning. Implementations that intelligently combine multiple drivers—using generative AI models to reason over the task state and requesting just-in-time human input to provide feedback—can significantly enhance the quality and adaptability of agent responses.

2.4.2.2 Tools

Tools, also known as skills or plugins, are specific implementations of logic designed to carry out particular tasks. They serve as the primary method for agents to *act* on tasks. Tools can be grouped into two categories: *general-purpose* tools and *domain-specific* tools.

General-purpose tools enable a broad range of capabilities, such as a code executor that allows agents to complete any task expressible as code, or a UI interface driver that allows agents to carry out tasks formulated as a sequence of UI interactions. In contrast, domain-specific tools are designed to address a specific task or a set of related tasks (e.g., calling a weather API with particular parameters). Providing agents with access to tools can significantly impact the range (diversity) and complexity of tasks that agents can address.

2.4.2.3 Memory

For agents to effectively perform tasks and improve over time, they need memory—the ability to recall and reuse information from past interactions. This enables agents to learn from experience and apply lessons to future tasks.

Short-Term Memory acts as working memory for the current task, tracking recent actions, conversation history, and temporary information needed to complete the immediate objective. In multi-agent systems, short-term memory often includes shared context so agents can coordinate effectively.

Long-Term Memory stores accumulated knowledge, successful strategies, and learned pat-

terns that persist across different tasks and sessions. This allows agents to build expertise over time and apply past experience to new situations.

Memory implementation varies from simple conversation logs to sophisticated knowledge bases, with the choice depending on the agent's intended use and complexity requirements. We explore specific memory architectures and implementation strategies in detail in later chapters.

These components work together in a continuous cycle: the model reasons about the current state and determines what action to take, selects and uses appropriate tools to execute that action, observes the results, updates its memory with new information, and adapts its approach for the next step.

i Conversational Programming

The action-perception loop described above raises a practical question: how do we implement such iterative reasoning and action-taking in code?

A powerful approach is *conversational programming*, where each step in the agent cycle—reasoning, acting, and processing results—is represented as messages in a conversation. This creates a structured sequence that captures both the agent's internal reasoning and the actions taken (tools used).

This message-based approach aligns naturally with chat-fine-tuned models, which are designed to process conversation histories and generate contextually appropriate responses. By representing agent actions as conversation turns, we leverage the model's existing strengths in understanding context and maintaining coherent dialogue.

Conversational programming enables agents to seamlessly blend natural language interaction with concrete actions like code execution or API calls. This paradigm forms the foundation for multi-agent frameworks like AutoGen (Wu et al. (2023)), where agents coordinate through message passing—a pattern we'll explore in detail in later chapters.

2.4.3 Agent vs. Model: A Clear Distinction

While both agents and models can process natural language and generate responses, agents differ fundamentally in their ability to interact with the world beyond text generation:

- **Models** excel at text processing, analysis, and generation based on their training data
- **Agents** can take actions, use tools, maintain context across interactions, and adapt their behavior based on results

Our NVIDIA stock price example (Task 2 from Figure 2.2) perfectly illustrates this distinction: a model might generate a plausible-sounding but potentially incorrect stock price based on its training data, while an agent would recognize the need for current information, use appropriate tools to fetch real-time data, and provide an accurate, up-to-date response.

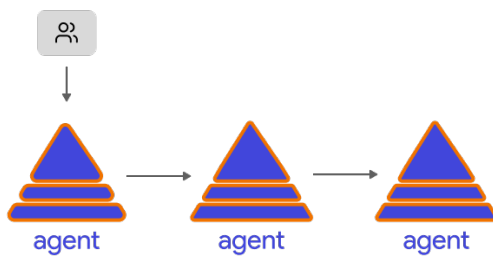
This ability to move beyond text generation into action-taking and adaptation is what makes agents powerful tools for addressing complex, real-world tasks that require more than just information synthesis.

2.5 What is a Multi-Agent System?

Building on our agent definition, we define multi-agent systems as applications that involve a *group of agents*, each with diverse capabilities and specialized objectives, *collaborating to solve tasks*. While these agents could be embodied in physical robots, this book focuses on agents that exist primarily in the digital world.

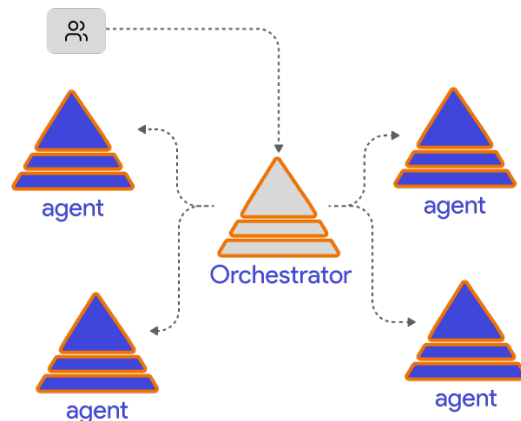
Multi-Agent Orchestration Workflows

Workflow (Graph) Orchestration



Control flow is predefined (typically modelled as a graph)

Autonomous (AI Driven) Orchestration



Control flow is Driven by an AI model at runtime across several patterns

GroupChat

Plan Based

Handoff

...

Figure 2.7: Two approaches to multi-agent orchestration: predefined workflows (left) versus AI-driven autonomous orchestration (right). The key difference lies in whether control flow is predetermined or emerges dynamically from AI-driven decisions at runtime.

i Multi-Agent System

A multi-agent system (or multi-agent application) is a collection of agents that collaborate to solve tasks. Each agent maintains specific capabilities—reasoning, acting, and communicating—and can adapt to changes in the task or environment. The key distinguishing feature of multi-agent systems is their *orchestration mechanisms*: the patterns that determine how agents communicate, when they act, and how they share data and control flow during task execution.

i Orchestration

Across this book, we will repeatedly use the term orchestration and it is rather helpful to define what we mean by the term. **Orchestration** refers to the mechanisms and patterns that enable multiple agents to work together effectively toward shared goals. This encompasses two main aspects - how they share information, and who controls the flow of execution (the order in which they act).

While academic literature and emerging practices sometimes uses “coordination” and “orchestration” interchangeably, we use “orchestration” as the primary term throughout this book to align with industry standards and common usage in AI/ML frameworks.

As shown in Figure 2.7, multi-agent systems can be organized through two distinct approaches. Multi-agent workflows (left) use predefined control flow where agents follow established sequences and handoffs, creating predictable and reliable processes. Autonomous multi-agent systems (right) use AI-driven orchestration where the control flow is determined dynamically at runtime, enabling adaptive responses to complex or unpredictable tasks. While both approaches use agents with the same core anatomy—model, memory, and tools—they differ fundamentally in how orchestration decisions are made.

2.5.1 Two Approaches to Multi-Agent Orchestration

Multi-agent systems can be organized through two fundamentally different approaches, each suited to different types of problems:

Multi-Agent Workflows (Defined Orchestration) These systems follow pre-defined collaboration patterns where each agent has clearly specified roles, responsibilities, and handoff points. The orchestration logic is explicitly programmed, creating predictable and repeatable processes. For example, a document processing workflow might have agents that specialize in text extraction, analysis, and formatting, working in a predetermined sequence with defined inputs and outputs for each stage.

Autonomous Multi-Agent Systems (AI-Driven Orchestration) These systems use AI models to drive orchestration decisions, allowing agents to dynamically negotiate responsibilities and adapt their collaboration based on task requirements and intermediate results. The orchestration emerges from agent interactions rather than being pre-programmed. This approach is particularly valuable for complex tasks where the optimal solution strategy cannot be predetermined and must evolve through exploration and adaptation.

The choice between these approaches directly impacts system behavior: workflows provide predictability and reliability, while autonomous systems offer adaptability and innovation. We

explore the specific patterns within each approach in detail in Chapter 2, including sequential and supervisor patterns for workflows, and group chat and handoff patterns for autonomous systems.

An Adaptive Multi-Agent Application Example

Consider this scenario: You need to create an application that books flights to specific destinations at the best price. However, your target airline, *specialairlines.com*, doesn't offer an API—only a web interface and mobile app designed for human users.

This creates several challenges beyond simply finding and booking flights:

Interface Navigation: The system must understand interface content (HTML elements, visual layouts), determine appropriate actions (clicking buttons, filling forms), and verify success (checking for confirmation messages). Each action changes the interface state, affecting subsequent possible actions.

Dynamic Adaptation: The system must handle interface changes (button relocations, updated form fields) and recover from errors (failed bookings, network issues) through iterative problem-solving.

Multi-Agent Orchestration: Success requires collaboration between specialized agents—perhaps a navigation agent for interface interaction, a payment agent for transactions, and a monitoring agent for verification—along with communication with human users when assistance is needed.

This scenario illustrates a task where the solution cannot be predetermined but must emerge through a series of adaptive actions. It exemplifies the complex problems that require reasoning, acting, adapting, and communicating across multiple agents—tasks that exceed the capabilities of current single-agent systems and represent ideal candidates for multi-agent approaches.

Distributed Agents: Agents that live across multiple organizational boundaries

Depending on the background of an engineer, the term “multi-agent systems” can evoke different mental models. For some, the focus is on software entities that collaborate within a single application/thread/process with similar permission structures. For others, the focus is on a *distributed internet of agents* where agents are primarily unaware of each other (requiring some discovery protocol), permissions (security and access) are markedly different, and communication must support both synchronous and asynchronous modes at scale.

In this book, we will focus primarily on the first scenario. While the second is important,

it is frequently the case that an early focus on distributed agents is overengineering—an increase in complexity—and there are standard methods that can help us transition to a distributed setup as the need arises. In the version 0.4 rewrite/redesign of AutoGen, we specifically provided a runtime concept that could be single process or distributed, allowing developers to create agents that could run in either mode without changing the agent logic itself. We found that the vast majority of use cases were well served by single process/thread applications.

Specifically, in a later chapter, we will discuss emerging protocols for distributed agent-to-agent communication, the design choices, and how existing agents can be adapted to work in a distributed setting.

2.6 Why Now?

Multiple factors highlight the importance of multi-agent Generative AI applications today, including advances in AI reasoning capabilities, economic opportunities through time arbitrage, self-improving systems, opportunities to tackle complex tasks, the increasing demand for reliable automation, platform economics, and the growing need for ethical AI deployment. These elements emphasize the significance of multi-agent systems in addressing contemporary challenges.

2.6.1 Advances in Generative AI Reasoning Capabilities

The general premise of multi-agent systems is not new. This topic has been extensively researched both from the perspective of understanding how humans collaborate to solve tasks (collective intelligence, crowdsourcing) as well as how artificial agents collaborate with themselves and humans (robot planning, robot navigation, human-robot collaboration, swarm intelligence, etc.). However, the development of artificial agents has been limited due to the lack of a *reasoning engine* (or *artificial brain*) that can adapt to context, synthesize plans that address tasks, and drive actions required as part of such plans. Recent advances in the demonstrated reasoning capabilities of Generative AI models, such as GPT-3.5 and GPT-4, change that status quo and now provide a critical component that enables new experiments and progress in creating truly autonomous multi-agent systems.

2.6.2 Economic Value Through Time Arbitrage

Multi-agent systems offer compelling economic advantages through time arbitrage. While an agent might take longer to complete a task than a human in absolute terms, the economics

favor delegation when considering the value of human attention. For example, a task that costs you \$30-100 of time (1 hour at your hourly value) might cost \$1-2 for an agent to complete over 2-3 hours. As GenAI inference costs continue to drop while human time becomes more valuable, this equation only improves.

2.6.3 The Promise of Self-Improving Systems

Unlike traditional software that requires engineering effort to improve, multi-agent systems *can* automatically benefit from advances in underlying AI models. This creates a unique opportunity to build systems that compound in value over time with minimal additional investment.

Note

Across my career, I have seen multiple instances where the performance of systems where capabilities are delegated to AI agents improve as the underlying model improves. A good example is my work with LIDA - a workflow based multi-agent system for automatically generating visualizations from data. The initial version of this system built with the davinci model family from openai had a 20% error rate on the initial evaluation harness. Simply switching to GPT 3.5 turbo which became available a few months after led to an error rate reduction from 20% to 3% across the board. I still see these sort of system level improvements as new advanced models become available.

2.6.4 Opportunity to Address Tacit Knowledge Tasks

Many repetitive problems today have been difficult to automate or support with reliable software tools. Existing research suggests that some of the automation challenges arise because these tasks require *tacit knowledge*— a form of knowledge that is challenging to articulate or codify but more readily transferred through experience or practice. However, groups of agents enabled by LLMs that encode vast amounts of knowledge and can collaborate with humans and other agents provide a new opportunity to address these complex tasks. This approach can help automate tasks that were previously considered infeasible to automate, thus reducing human effort and potentially errors.

2.6.5 Increasing Demand for Reliable Automation

Businesses and individuals seek AI solutions that can autonomously execute sophisticated tasks while maintaining high levels of reliability and safety. Humans should still remain in