# AI Agents & Automation — Engineering Intelligent Workflows

## From LLMs to Reliable Agentic Systems

Dr. Yves J. Hilpisch[1]

February 2, 2026

THE AI
ENGINEER

[1] Get in touch: linktr.ee/dyjh. Web page: theaiengineer.dev. Research, structuring, drafting, and visualizations were assisted by GPT 5.x as a co-writing tool under human direction. Comments and feedback are welcome.

# Preface

*This book is the capstone of a series that starts with Python and mathematics and culminates in building systems that act. Models are components inside agentic workflows that observe, plan, and use tools to achieve goals under constraints.*

Who this is for: Engineers comfortable with Python and large language model (LLM) fundamentals who want to design and ship reliable agentic systems. If you have ever wired a small script to an API and wondered how to turn it into a predictable service, this book is for you.

How we work: Small, testable examples; concise outputs; honest figures. Each chapter states 3–5 learning objectives, includes a deterministic demo, and ends with exercises and acceptance criteria. The case studies in the later parts reuse the same harnesses you meet early on, so improvements compound rather than fragment. See `book_charter.md` for the detailed rubric.

## Why Agents, Why Now

Software automated repetitive tasks for decades. Machine learning raised the bar by learning patterns from data. Large language models added a new twist: they can read, write, and reason across many domains. Agents take the final step: they combine reasoning with action, grounded in tools and memory, and aim for outcomes rather than mere predictions. This book is about engineering those outcomes safely and repeatably, with the same humility and discipline you would apply to any production system.

## A Short History of Agents

It helps to know whose shoulders we stand on.

- **Expert systems (1970s–1990s):** rule bases and inference engines (forward or backward chaining) codified expert knowledge. They worked well where rules were stable and exhaustive, but struggled with uncertainty and scale.
- **Reinforcement-learning agents (1990s–2010s):** systems that learn by acting in an environment, receiving rewards, and optimizing policies over time. They excelled in simulations and games and gave us the vocabulary of state, action, reward, and policy.
- **LLM-powered agents (2020s–):** language models that plan and call tools. They bring broad knowledge and flexible reasoning, but require engineering to avoid hallucinations, keep costs in check, and act within guardrails.

You will see traces of all three in this book: rule-like guardrails, reinforcement-learning style loops and budgets, and LLMs as planners. The goal is not to win a taxonomy debate, but to give you a shared mental model you can use with colleagues in infrastructure, product, and research.

## How to Use This Book

Each chapter starts with a lead paragraph and a trio of callouts titled “You’ll Need”, “At a Glance”, and “You’ll Learn”. Prose anchors concepts; tiny runnable scripts in the companion repository make them real;

github.com/yhilpisch/agecode

exercises cement understanding. The Sanity Box in each chapter lists quick checks so you know you are on track. You can read linearly, but it is common to jump between code and text: run a snippet, then return to the page that explains why it behaves that way.

## What’s Inside

- Part I (Ch. 1–3): mental model, anatomy, and engineering mindset for agents.
- Part II: building blocks—language models as brains, memory, tools, and multi-agent workflows.
- Part III: frameworks and deployments, from local harnesses to cloud services.
- Part IV: applications in finance, knowledge work, and industry settings.
- Part V: safety, governance, and the road ahead, including scenario planning and signals to watch.

Turn the page when you can explain the loop in Chapter 1 out loud. That sentence becomes your compass for the rest of the book; later parts simply refine how you observe, plan, act, and learn under tighter budgets and in richer environments.

# Contents

Preface i

I Foundations of AI Agents 1

1 The Evolution Towards Agents 3
1.1 Why It Matters . . . . . 4
1.2 Key Ideas . . . . . 4
1.3 Visual Overview . . . . . 4
1.4 Worked Example—Simple Calculator Agent . . . . . 6
1.5 From Code → Concepts . . . . . 7
1.6 Exercises (with Acceptance Criteria) . . . . . 7
1.7 Further Reading . . . . . 8
1.8 Where We Are Heading Next . . . . . 8

2 The Anatomy of an AI Agent 9
2.1 Why Components Matter . . . . . 10
2.2 Policy: Brains with a Checklist . . . . . 11
2.3 Tools: Hands with Safety Gloves . . . . . 11
2.4 Memory: The Agent's Notebook . . . . . 12
2.5 Environment: Where Work Lands . . . . . 12
2.6 Putting the Loop Together . . . . . 12
2.7 Exercises (Acceptance Criteria) . . . . . 13

3 The Engineering Mindset for Agents 14
3.1 Observability: Seeing the Loop Clearly . . . . . 14
3.2 Budgets: Envelopes for Time, Cost, and Risk . . . . . 15
3.3 Guardrails: Safety by Default . . . . . 15
3.4 From Demo to Service . . . . . 15
3.5 Exercises (Acceptance Criteria) . . . . . 16

II Building Blocks of Agentic Systems 17

4 Language Models as Agent Brains 19
4.1 Why LLMs Make Good Planners (and When They Don't) . . . . . 20
4.2 Prompt Patterns That Produce Actions . . . . . 20
4.3 Minimal Planner Stub (No API Required) . . . . . 20
4.4 Exercises (with Acceptance Criteria) . . . . . 21

5 Memory and State 23
5.1 Why Memory? 23
5.2 A Tiny Memory Store (Local Only) 24
5.3 Integrating Memory into the Loop 25
5.4 Exercises (Acceptance Criteria) 26
5.5 Further Reading 27

6 Tools and APIs 28
6.1 A Minimal Tool Adapter (Local Only) 29
6.2 Putting It Together: Adapter in the Loop 30
6.3 Sandboxing Side-Effecting Tools 31
6.4 Exercises (Acceptance Criteria) 32
6.5 Further Reading 32
6.6 Where We Are Heading Next 32

III Orchestration and Collaboration 33

7 Workflows and Orchestration 35
7.1 Why Workflows for Agents? 35
7.2 A Tiny Workflow Engine (Local Only) 36
7.3 Putting It Together: A Small ETL Flow 37
7.4 Exercises (Acceptance Criteria) 37

8 Multi-Agent Collaboration 39
8.1 Why Teams of Agents? 39
8.2 A Minimal Chat Between Planner and Critic 40
8.3 Putting It Together: A Small Specialist Team 41
8.4 Exercises (Acceptance Criteria) 41
8.5 Further Reading 42
8.6 Where We Are Heading Next 42

IV Frameworks and Platforms 43

9 Agent Frameworks 45
9.1 What Carries Over Between Frameworks 45
9.2 A Minimal Framework Adapter (Local Only) 46
9.3 Side-by-Side: Plain vs. Framework Runner 47
9.4 When to Use a Framework 48
9.5 Exercises (Acceptance Criteria) 48
9.6 Further Reading 48

10 Agents in the Cloud and on the Edge 49
10.1 Cloud-Style Request Handler (Local Stub) 49
10.2 Tiny Edge Loop (Local Only) 50
10.3 Sandboxing State and Files 51
10.4 Statelessness and Idempotency at the Edge 51
10.5 Exercises (Acceptance Criteria) 51
10.6 Further Reading 52
10.7 Where We Are Heading Next 52

**V Applications of AI Agents 53**

**11 Agents in Finance 55**
11.1 A Minimal Research → Signal → Report Pipeline . . . 56
11.2 Designing the Pipeline . . . 56
11.3 Data and Assumptions . . . 56
11.4 Designing Signals That Age Well . . . 57
11.5 Risk, Compliance, and Scope . . . 57
11.6 Observability and Audit . . . 57
11.7 Failure Modes and Fallbacks . . . 57
11.8 From Toy to Real Data . . . 58
11.9 Exercises (Acceptance Criteria) . . . 58

**12 Agents in Knowledge Work 59**
12.1 Extracting Fields from Text . . . 59
12.2 Building a One-Line Summary . . . 60
12.3 Document Types and Scope . . . 61
12.4 Extraction Strategies . . . 61
12.5 Quality Checks and Fidelity . . . 61
12.6 Scaling Up Thoughtfully . . . 62
12.7 Exercises (Acceptance Criteria) . . . 62

**13 Agents in Industry 63**
13.1 A Tiny Queue Runner with Metrics . . . 63
13.2 Queue Design and Statuses . . . 64
13.3 SLA Budgets and Backpressure . . . 64
13.4 Retries, Idempotency, and Fallbacks . . . 65
13.5 Metrics and Alerts . . . 65
13.6 Human in the Loop . . . 65
13.7 Exercises (Acceptance Criteria) . . . 65

**VI Case Studies: End-to-End Systems 67**

**14 Case Study — Research & Reporting Assistant (RAG + Planning) 69**
14.1 Problem Framing and Acceptance . . . 70
14.2 Local Retrieval Stub (Dependency-Free) . . . 70
14.3 Planning and Drafting . . . 71
14.4 Fact-Checking Pass (Minimal) . . . 72
14.5 Observability and Audit . . . 72
14.6 Swapping in a Framework (Optional) . . . 72
14.7 Exercises (Acceptance Criteria) . . . 72

**15 Case Study — Financial Signals & Compliance 74**
15.1 Shape of the Pipeline . . . 74
15.2 Local Pipeline (Dependency-Free) . . . 75
15.3 Audits and Reports . . . 76
15.4 Scaling Safely . . . 76
15.5 Exercises (Acceptance Criteria) . . . 76

**16 Case Study — Ops Ticket Triage & Automation 77**

16.1 Shape of the Loop . . . 77

16.2 Tiny Multi-Role Triage (Local) . . . 78

16.3 Logs and Metrics . . . 79

16.4 Optional: Service Wrapper . . . 79

16.5 Exercises (Acceptance Criteria) . . . 79

**VII Frontiers and Challenges 80**

**17 Safety, Alignment, and Governance 82**

17.1 Threat Modeling for Agents . . . 82

17.2 A Minimal Policy Guard (Local Only) . . . 83

17.3 Red-Team and Canary Harness . . . 84

17.4 Governance: Policy, Provenance, and Review . . . 84

17.5 Privacy and Responsible Data . . . 84

17.6 Exercises (Acceptance Criteria) . . . 84

**18 The Future of Agents 86**

18.1 Durable Patterns and Likely Trajectories . . . 86

18.2 A Tiny Scenario Playbook (Local Only) . . . 87

18.3 Tracking Experiments and Decisions . . . 87

18.4 Signals to Watch . . . 88

18.5 Exercises (Acceptance Criteria) . . . 88

**Epilogue 89**

**Glossary of Agent Terms 91**

**Agent Engineering Cheat Sheet 94**

**Framework Comparison Tables 96**

**Starter Templates 97**

# List of Figures

1.1 Keeping each phase within its latency budget forces instrumentation and logging before adding more tools or autonomy. . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.1 Chapter 2 insists every component connection is explicit so you can trace where information is created, transformed, or persisted. . . . . . . . . . . . . . . . . . . . 10

3.1 Chapter 3 insists every incident loops through shared telemetry, codified budgets, and human playbooks before agents regain autonomy. . . . . . . . . . . . . . . . . 16

4.1 Token budgets (gray line) create a ceiling the planner must respect, forcing concise reasoning. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

5.1 Chapter 5 insists on explicit read/write paths so planners know exactly which facts they can trust. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

6.1 High-capability, high-risk tools (top-right) demand sandboxing, human review, or throttles before production. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

7.1 States cascade left to right; downstream nodes wait until upstream work is finished. 37

8.1 Alternating turns keep ownership clear: planners propose, critics approve or request revisions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

9.1 Adapters pass the same events forward so observability stays intact after migration. 48

10.1 Cloud handlers tolerate higher latency but more resources; edge loops demand strict budgets. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52

11.1 Each boundary enforces schema validation before artifacts move downstream. . . 55

12.1 Balanced extraction prevents blind spots before summarization. . . . . . . . . . . 61

13.1 Points above the gray SLA line trigger escalations or more capacity. . . . . . . . 65

# Contact

AI Agents & Automation — Engineering Intelligent Workflows

The AI Engineer

Get in touch:

https://linktr.ee/dyjh

https://theaiengineer.dev

© 2026 Dr. Yves J. Hilpisch — All rights reserved.