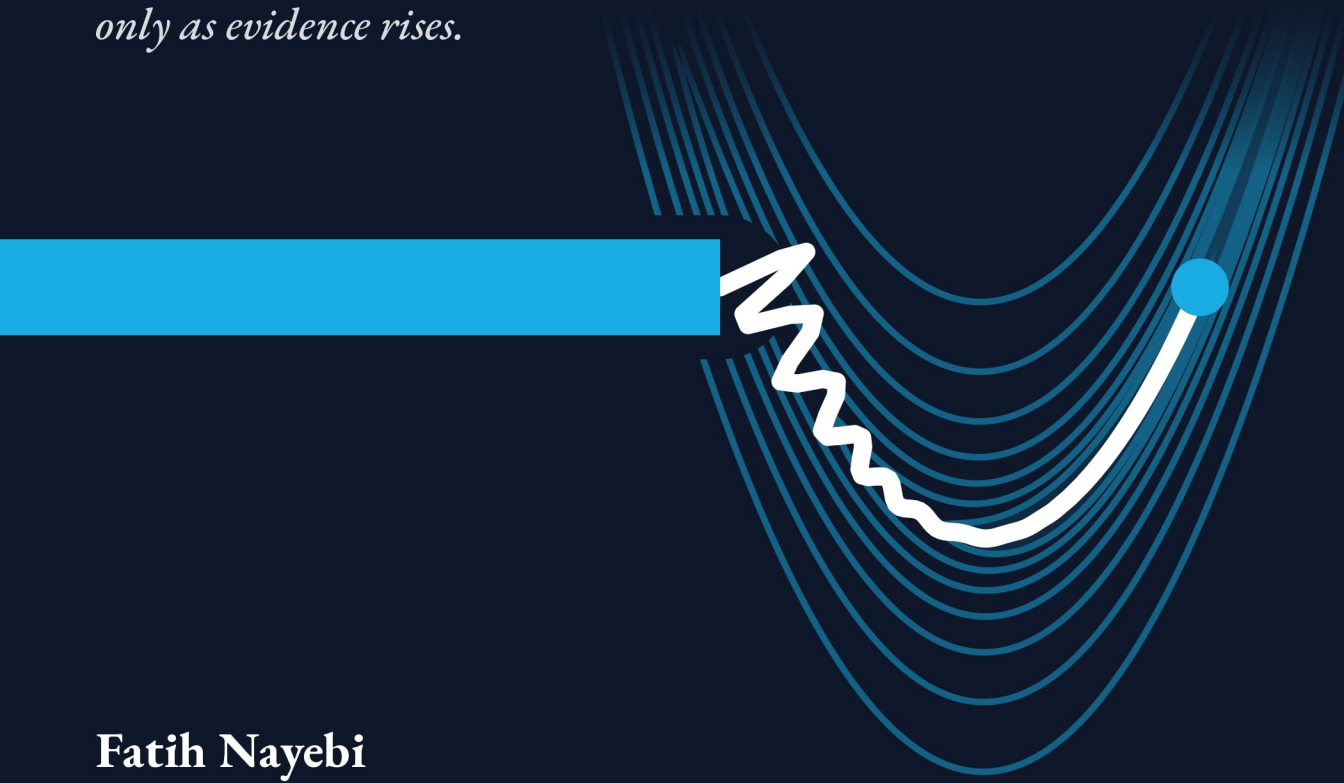
Foundations of
Agentic AI Systems
Volume I

Models, Learning, and Decisions, from First Principles

Autonomy should rise
only as evidence rises.



Fatih Nayebi

FIRST EDITION

Foundations of Agentic AI Systems

Volume I: Models, Learning, and Decisions, from First Principles

Fatih Nayebi

Sample - Not for Redistribution

Gradient Divergence
Montréal, QC, Canada

Foundations of Agentic AI Systems, Volume I
Models, Learning, and Decisions, from First Principles
First edition

Copyright © 2026 Fatih Nayebi. All rights reserved.
No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

Published by Gradient Divergence, Montréal, Québec, Canada
For permissions and inquiries, write to contact@gradientdivergence.com.

First published 2026. Version 1.0 (build 02e5a221b).
ISBN 978-1-0694226-6-8 (hardcover)
ISBN 978-1-0694226-3-7 (paperback)

The code in this book is released under the MIT License at
https://github.com/gradient-divergence/agentic-ai-systems.
Please report errors there as issues.

Models, interfaces, prices, and regulations are described as they stood in October 2026.
Nothing in this book is legal advice.
While every precaution has been taken in the preparation of this book, neither the author nor the publisher assumes any liability for errors or omissions, or for damages resulting from the use of the information contained herein.
Product and company names may be trademarks of their owners and are used here editorially.

Set in STIX Two and Fira Sans, with code in Inconsolata, using LuaLaTeX.

Sample – Not for Redistribution

The Analytical Engine has no pretensions whatever to *originate* any thing. It can do whatever we *know how to order it* to perform.

— Ada Lovelace, 1843

Sample - Not for Redistribution

A. A. L. [Ada Lovelace], "Notes by the Translator," Note G, in L. F. Menabrea, "Sketch of the Analytical Engine invented by Charles Babbage, Esq.," *Scientific Memoirs*, vol. III, ed. Richard Taylor (1843, 722).

Table of Contents

Preface xiii
Introduction xix

PART I
Orientation and Agentic Mindset

1 What Is Agentic AI? 3
2 Architecture, Components and Contracts 17
3 Decision Strategy and Evidence Requirements 27

PART II
ML Foundations

4 ML Data and Evaluation 39
5 Supervised Learning and Calibrated Risk 53
6 Semi- and Self-Supervised Learning and Unlabeled Structure 75
7 Causal Inference and Experiments 91
8 Delegated Model Building 107

PART III
Deep Learning Foundations

9 Neural Networks and Backpropagation 123
10 Training Optimization and Regularization 137
11 Autoencoders and NLP Representations 151
12 Sequence Processing, Transformers and Foundation Models 163
13 Generative Models and Simulation 205
14 Prompting, Fine-Tuning and Retrieval Adaptation 223

PART IV
RL and Decision-Making

15 Constrained Optimization for Agentic Decisions 241
16 Reinforcement Learning Foundations 265
17 Exploration, Bandits and Exposure Budgets 279
18 Deep Reinforcement Learning 293
19 RLHF and Preference Alignment 309
20 Typed Decision Models and Calibrated Decisions 323
21 Practical RL and Off-Policy Evaluation 339

Before the Agent Acts 353
Appendix A: Mathematical Foundations 359
References 407
Acknowledgments 421
About the Author 423
Index 425

Sample - Not for Redistribution

Preface xiii

Introduction xix

PART I Orientation and Agentic Mindset

1 What Is Agentic AI? 3
1.1 What makes a system agentic 4
1.2 An agent in its environment 5
1.3 Four levels of autonomy 8
1.4 If agents do the work, why learn how they work? 10
1.5 A decision boundary for the refund agent 11
1.6 When to use a workflow and when to use an agent 13
1.7 Summary 14
1.8 Exercises 14

2 Architecture, Components and Contracts 17
2.1 Six layers and a control plane 17
2.2 What a contract at a layer boundary promises 19
2.3 What a change does across a boundary 21
2.4 Who owns each layer, and what to build first 23
2.5 Summary 24
2.6 Exercises 25

3 Decision Strategy and Evidence Requirements 27
3.1 Why autonomy is granted one decision at a time 27
3.2 Why evidence is named before the run that judges it 28
3.3 How many clean cases a claim needs 29
3.4 The evidence each tier of the refund agent must show 32
3.5 Summary 35
3.6 Exercises 35

PART II ML Foundations

4 ML Data and Evaluation 39
4.1 What a label records about a decision 39
4.2 Splitting cases in time 41
4.3 What a score may know at decision time 42
4.4 Precision, recall and the base rate 44
4.5 An interval for a measured rate 46
4.6 The refund agent's golden set 49
4.7 Summary 51
4.8 Exercises 51

5 Supervised Learning and Calibrated Risk 53
5.1 Learning a probability from labeled requests 53
5.2 Bias, variance and boosted trees 57
5.3 Expected cost and the decision threshold 59
5.4 The abstain band, where a person decides 62
5.5 Checking and repairing calibration 65
5.6 Calibration at each amount, and the cost of a cutoff 68
5.7 When the base rate changes 69
5.8 Summary 72
5.9 Exercises 72

6 Semi- and Self-Supervised Learning and Unlabeled Structure 75
6.1 What unlabeled requests can show 75
6.2 Choosing how many groups, and what kind 76
6.3 Testing a group or an anomaly score against labels 79
6.4 Self-training, and what a model learns from its own labels 81
6.5 Self-supervised learning and the contrastive loss 85
6.6 Where the next label comes from 87
6.7 Summary 89
6.8 Exercises 89

7 Causal Inference and Experiments 91
7.1 Two outcomes for every request, and only one on record 91
7.2 Why the comparison in the records misleads 92
7.3 What the refund history can say, and what it cannot 95
7.4 What randomizing an action buys, and what it does not 96
7.5 What a cohort picked by hand can still say 97
7.6 The refund agent's threshold experiment plan 99
7.7 Checking an effect you did not estimate 104
7.8 Summary 105
7.9 Exercises 105

8 Delegated Model Building 107
8.1 Commissioning a model: what the delegate may read and change 107
8.2 Spending a search budget on settings, inputs and combinations 108
8.3 Why the best of many scores flatters the winner 113
8.4 Checking a delivered model against frozen cases 115
8.5 The refund agent's selection record 116
8.6 Summary 118
8.7 Exercises 118

PART III Deep Learning Foundations

9 Neural Networks and Backpropagation 123
9.1 What a network computes on a request's words 123
9.2 From scores to a loss with softmax and cross-entropy 124
9.3 Backpropagation, one layer at a time 126
9.4 Checking a gradient by hand 130
9.5 A wrong gradient can still train 132

Sample Not for Redistribution

9.6 Accepting a backward pass you did not write 134
9.7 Initialization, and what a correct gradient does not promise 135
9.8 Summary 136
9.9 Exercises 136

10 Training Optimization and Regularization 137
10.1 Why a correct gradient can make the loss climb 137
10.2 Normalization changes the curvature a step meets 141
10.3 Momentum and Adam give each direction its own step 142
10.4 Why the learning rate is warmed up and then decayed 144
10.5 Weight decay and early stopping shrink the same directions 145
10.6 Checking a training run an agent ran 147
10.7 Summary 149
10.8 Exercises 149

11 Autoencoders and NLP Representations 151
11.1 Text as a vector of counts, and what similarity measures 151
11.2 What a reconstruction objective keeps 152
11.3 Judging a representation on a frozen labeled set 156
11.4 A policy index that knows its effective dates 157
11.5 Summary 160
11.6 Exercises 161

12 Sequence Processing, Transformers and Foundation Models 163
12.1 Reading a sequence one token at a time 164
12.2 How attention lets every token read every other 171
12.3 Why the scores are divided by the square root of the key width 174
12.4 Attention heads and the order of tokens 177
12.5 The transformer block and the three model families 178
12.6 Training a language model to predict the next token 182
12.7 One pretrained model, many tasks 187
12.8 Taking a task from the context 189
12.9 The decoder as a decision policy 191
12.10 What a long context costs to compute, cache and serve 193
12.11 A context and cost budget for the refund agent 200
12.12 Summary 201
12.13 Exercises 201

13 Generative Models and Simulation 205
13.1 Drawing a token: greedy decoding, temperature, top-k and nucleus sampling 205
13.2 The probability of a whole output: exact, bounded, or unknown 207
13.3 Diffusion models: adding noise step by step, and learning to remove it 215
13.4 Simulated scenarios for the refund agent, and the checks between generation and use 218
13.5 Summary 221
13.6 Exercises 222

14 Prompting, Fine-Tuning and Retrieval Adaptation 223
14.1 Three places to put what a model must know 223
14.2 What fine-tuning minimizes, and how few weights it changes 227
14.3 Comparing adaptations on the same cases 229
14.4 Which model, and where it runs 232
14.5 The refund agent's adaptation, and the suite it leaves 234
14.6 Summary 238
14.7 Exercises 238

PART IV RL and Decision-Making

15 Constrained Optimization for Agentic Decisions 241
15.1 From a permitted action to a best one 241
15.2 The bound a linear program carries with its answer 242
15.3 Whole decisions, and the bound their relaxation gives 246
15.4 Schedules as constraint models 249
15.5 Finding a plan by search, and checking one a model proposes 251
15.6 The refund agent's review-queue schedule 256
15.7 When a solver earns its place, and what it cannot see 259
15.8 Summary 263
15.9 Exercises 263

16 Reinforcement Learning Foundations 265
16.1 When today's choice changes tomorrow's state 265
16.2 What a decision policy is worth 267
16.3 Why waiting is often worth more than acting 267
16.4 Learning the values without the model 272
16.5 What the learner is told: its state and its reward 273
16.6 The refund agent's case for not learning yet 275
16.7 Summary 277
16.8 Exercises 277

17 Exploration, Bandits and Exposure Budgets 279
17.1 What exploring costs, and who pays for it 279

Sample - Not for Redistribution

17.2 Estimating each offer, and exploring at random 280
17.3 Exploring where the desk is unsure: confidence bounds and Thompson sampling 282
17.4 A budget, and the cap it allows 285
17.5 Late answers, and the estimates a learner bends 287
17.6 The refund agent's offer bandit, run on last year's refunds 288
17.7 Summary 290
17.8 Exercises 290

18 Deep Reinforcement Learning 293
18.1 Learning values beyond a table 293
18.2 Why a shared weight can chase its own target 295
18.3 Training a deep Q-network across five seeds 296
18.4 Learning the policy directly 299
18.5 Proximal policy optimization, and what its clip does not bound 302
18.6 Accepting a deep RL result an agent hands you 305
18.7 When the offer bandit outgrows its table 306
18.8 Summary 307
18.9 Exercises 307

19 RLHF and Preference Alignment 309
19.1 From a comparison to a reward 309
19.2 Training against a learned reward, with a KL penalty 311
19.3 Fitting the policy to the comparisons directly 314
19.4 What preference training does not grant 319
19.5 Summary 321
19.6 Exercises 321

20 Typed Decision Models and Calibrated Decisions 323
20.1 A question whose answers are declared in advance 323
20.2 Scores that reward the truth 325
20.3 What a Brier score is made of 327
20.4 Three bands, and the action no probability may take 329
20.5 Sets that come with a coverage guarantee 331
20.6 Training a model to state honest probabilities 334
20.7 Accepting a decision model you did not train 336
20.8 Summary 337
20.9 Exercises 337

21 Practical RL and Off-Policy Evaluation 339
21.1 Judging a new policy from an old policy's log 339
21.2 Importance weighting: making the log speak for another policy 340
21.3 When a few records carry the estimate 343
21.4 The doubly robust estimate: a reward model corrected by the log 344
21.5 Longer horizons, and the policy simulators that answer where logs cannot 345
21.6 Judging the refund agent's offer policy, or holding it 346
21.7 Summary 350
21.8 Exercises 351

Before the Agent Acts 353

Appendix A: Mathematical Foundations 359

References 407

Acknowledgments 421

About the Author 423

Index 425

Sample - Not for Redistribution

Preface

A systems-first approach

This book is about AI that acts as well as predicts. The work has moved from isolated models to systems that perceive context, reason over constraints, act through tools, and learn from outcomes. The gap between a compelling demo and a reliable, governed system remains wide.

Foundations of Agentic AI Systems sets out to close that gap in two volumes. This first volume, *Models, Learning, and Decisions, from First Principles*, builds the foundations an agent stands on, from data and evaluation through deep learning to reinforcement learning and decision-making. Volume II, *Agents, Operations, and Governance, in Production*, builds the agent on those foundations and takes it to production. Both volumes connect first principles to production practice through a single reference architecture and an evaluation spine that keeps every capability measurable, safe, and operable.

Why now?

Foundation models, tool calling, retrieval, and orchestration frameworks have made agentic systems easier to prototype. But production success still depends on engineering disciplines that outlast any framework: contracts, evaluation, monitoring, and governance.

If you build systems that act in the world, you need more than prompts: you need architecture, reliability, and feedback loops that scale.

ⓘ Example

The triage agent that rolled back the wrong thing

The agent starts as a productivity win: it reads tickets, pulls the runbook, and drafts the exact steps your on-call engineer would have typed. In a sandbox, it feels like cheating. At 2:17am, an incident ticket arrives: "Can you roll back checkout? It's acting weird after the last deploy." The request is real, but it is missing the one detail that matters most: *which environment*.

The agent may gather context and draft a rollback plan, but it must not execute a production change without an explicit environment and an approver.

A healthy run takes five steps:

1. Parse the request and extract `service`, `environment`, and `change_type`. Ask a clarifying question if any are missing.
2. Retrieve the rollback runbook and current release state. Attach links and the active policy version.
3. Propose a rollback with a risk tier, a verification plan, and the exact tool call it would make.
4. Execute only after schema validation and approval. Then verify the expected movement in error rate and latency.
5. Emit a run record: inputs, policy version, tool I/O, verification results, rollback path.

But the tool call silently defaults `environment="prod"` when the field is omitted. The agent treats the request as obvious, executes, and rolls back the wrong system. The intent is fine. The system design is not.

Sample for Redistribution

> The fix is four changes to that design: make `environment` required in the tool schema, route ambiguity into a "clarify" policy, gate high-impact actions behind approvals, and treat verification as a step in the run, not in the postmortem.
>
> Approval logs, audit trails, trace IDs, and error-budget burn tell you whether these controls prevent repeats.

Who these books are for

The two volumes have two primary readers: my graduate students at McGill University, and practitioners who own, or are about to own, an agent that acts in production.

This volume assumes familiarity with Python and basic statistics. Knowledge of scikit-learn and PyTorch is helpful, but the focus is on system design and evaluation rather than framework mastery.

Scope and structure

The two volumes are organized into eight parts, and this volume holds the first four. It moves from core definitions and data foundations through deep learning to reinforcement learning and decision-making.

Part	Core question	What you get
I: Orientation and agentic mindset	*What makes a system agentic, and how much may it decide?*	Definitions, architecture spine, evaluation discipline.
II: ML foundations	*What must be known of a score before a decision may rest on it?*	Data pipelines, model selection, evaluation.
III: Deep learning foundations	*What does a model compute on text, and how is one adapted to a single policy?*	Deep architectures and adaptation trade-offs.
IV: RL and decision-making	*What should the agent choose, and should it learn to choose?*	RL, bandits, alignment, safety constraints.

Volume II holds Parts V to VIII: agentic systems architecture, engineering and operations, governance and adoption, and the capstone and futures. It opens with a bridge that recaps what it needs from this volume.

A refund agent runs through both volumes. What would it have to show before it could pay a refund on its own? This volume ends with the agent at stage 3, drafting every decision for a person to carry out. Volume II takes it from there to a launch gate: its capstone sets out what the agent's launch packet must hold and runs that gate on a constructed packet in the companion, and the chapter after it carries the same decision packet into three other domains. The run that would assemble the agent's own packet is still to be made. In print, the margin beside each chapter's first line shows its part's map, with the chapter's layer lit and the stage the agent reaches by the end of that part.

Reading paths

Pick the path that fits you. Chapter numbers are those in the table of contents, and a chapter in Volume II is named with its volume. Each path starts in this volume and continues in the next.

Sample. Not for Redistribution

Path	For whom	In this volume	Then
Core	Any reader	The introduction; Chapter 1 to Chapter 4; Section 5.3 to Section 5.6; Section 8.1 to Section 8.3; Section 9.1 to Section 9.3 and Section 10.1 to Section 10.2; Chapter 11 to Chapter 12; Chapter 14	Volume II, Chapter 22 to Volume II, Chapter 29
Builder	Engineers shipping an agent	As Core, with Section 7.1, Section 7.3, Section 7.4 and Section 7.6	As Core, then Volume II, Chapter 30 to Volume II, Chapter 39
Technical lead or architect	Readers who own design and risk	As Core, with Section 7.1, Section 7.3, Section 7.4 and Section 7.6; Section 15.6; Section 20.4; Chapter 17 and Chapter 21	The Part V opener; Volume II, Chapter 22, Volume II, Chapter 23, Volume II, Chapter 25, and Volume II, Chapter 29; Volume II, Chapter 30 to Volume II, Chapter 35; Volume II, Chapter 36 to Volume II, Chapter 38; Volume II, Chapter 39 to Volume II, Chapter 41
Sponsor	Executives and product leads	The introduction; Chapter 1; Chapter 3	The Part V opener; Volume II, Chapter 25, Volume II, Chapter 37, Volume II, Chapter 38, Volume II, Chapter 39, Volume II, Chapter 40, and Volume II, Chapter 41
Graduate course, 13 weeks	Students with one prior ML course	Chapter 1 to Chapter 3; Chapter 4 to Chapter 8; Section 9.1 to Section 9.3 and Section 10.1 to Section 10.2; Chapter 11, Chapter 12, and Chapter 14; Chapter 16, Chapter 17, Section 18.4 and Section 18.5, and Chapter 19	Volume II, Chapter 22 to Volume II, Chapter 29; Volume II, Chapter 30 to Volume II, Chapter 35; Volume II, Chapter 36 to Volume II, Chapter 38; Volume II, Chapter 39 as the project, then Volume II, Chapter 40 and Volume II, Chapter 41
Foundations first	Readers new to ML	Parts I to IV in order	Volume II, Chapter 22 to Volume II, Chapter 29, as in Core

The Sponsor path takes tool contracts and evaluation gates on trust. Volume II, Chapter 29 tests the refund agent on, among other cases, the generated scenarios of Section 13.4. There, on the agent's first trial day, its decisions are logged and not carried out, and every payment it would make carries the standard notice, because Section 21.6 holds back both candidate rules for choosing a payment's notice. The Core, Builder and graduate paths take the scenarios and that decision on trust; the Technical lead path reads the decision and takes only the scenarios on trust. Volume II, Chapter 37 sizes a desk of reviewers with the review times of Section 15.6, and the score bands of Section 20.4 add no work to that desk. The Builder, Sponsor

Sample - Not for Redistribution

and graduate paths, which reach that chapter without either section, can therefore take the review times and the bands on trust.

Each chapter opens with what is at stake, works through a concrete example, and ends with exercises. Use the reading paths as shortcuts, but keep in view the evaluation spine, the connective tissue across every path.

Using these books in a course

The two volumes grew out of the five graduate courses I teach at McGill University. The table lists each course with the year I first taught it and the chapters that hold its material, so students and instructors can find a course's chapters in either volume. The chapters are optional reading, and no course of mine requires the books.

Course	First taught	Chapters
Enterprise Data Science	2019	Chapter 3 to Chapter 7; Chapter 16; Volume II, Chapter 38
Machine Learning in Production	2019	Chapter 8; Volume II, Chapter 30 to Volume II, Chapter 36, with Volume II, Chapter 23, Volume II, Chapter 25, Volume II, Chapter 29 and the shadow-to-canary requirement of Table 3.1, which Part VI assumes
Introduction to AI and Deep Learning	2020	Chapter 9 and Chapter 10; Appendix A
Applications and Architectures of Deep Learning	2020	Chapter 11 to Chapter 14; Volume II, Chapter 23
Designing and Building Agentic AI Systems	2023	Chapter 1 and Chapter 2; Chapter 12; Chapter 16 to Chapter 19; Volume II, Chapter 22 to Volume II, Chapter 29; Volume II, Chapter 37; Volume II, Chapter 39 and Volume II, Chapter 41

For most chapters the answers are in the companion repository, described at the end of this preface. The keys to the numerical exercises sit in a file with the outputs printed beside each chapter's listings, and the reference solutions to the builds, with their acceptance tests, sit in `exercises/`.

Why I wrote these books

I wrote *Foundations of Agentic AI Systems* because I kept seeing the same pattern: agent demos are dazzling, and production failures are boring. The failures usually come from missing contracts, fuzzy decision boundaries, weak evaluation, or brittle operations, not from the model "not being smart enough."

I write from the intersection of enterprise technology, academic research, and product delivery. As Founder and CEO of OODARIS AI, I work on agentic systems and deal with the realities of integration, governance, and measurement every day.

As an Assistant Professor at McGill University, I teach and research AI, and I mentor practitioners from diverse backgrounds who share an interest in applied work. My earlier roles leading data, analytics, and AI programs ground the examples in operational reality.

I have taught at McGill since 2019, in person and online, in the Master of Management in Analytics. My first courses were *Enterprise Data Science* and *Machine Learning in Production*. A year later I added *Introduction to AI and Deep Learning* and a second

Sample - Not for Redistribution

deep learning course, *Applications and Architectures of Deep Learning*. In 2023 I began teaching *Designing and Building Agentic AI Systems*. These two volumes hold the material of all five courses. My students knew I was writing them and have been waiting for them, and with these volumes I can at last keep my promise.

Together, the two volumes are a record of what has worked and a guide to what comes next. My hope is that these pages demystify agentic AI and help you move responsibly from proof of concept to production.

Disclosure and neutrality

I am an Assistant Professor at McGill University and the Founder and CEO of OODARIS AI. These books are written as an independent academic and practitioner reference. Product or platform mentions are provided for instructional context, not endorsement. Where tools or vendors are cited, the aim is to illustrate patterns and trade-offs rather than prescribe a specific solution.

Code repository

The companion repository contains runnable examples and short demos aligned to both volumes. It is the canonical reference for the code shown in print.

The repository is at https://github.com/gradient-divergence/agentic-ai-systems, and its demos cover every major part of the book, from supervised ML and deep learning through agentic systems, security, observability, and the capstone. Its README gives the install steps and the checks to run, and the code needs Python 3.10 or newer.

The labs of my courses are public too. The McGill showcases repository, at https://github.com/conqueror/mcgill-showcases, holds 25 machine learning projects written for these courses, each with its own guide and tests.

Sample - Not for redistribution

Introduction

An assistant that drafts an email is useful. An agent that can change a deployment, approve a refund, or rotate credentials is something else entirely: it can change external state, quickly, at scale. When it goes wrong, the blast radius is not a bad answer. It is an incident.

I have spent years building and operating these systems, and the pattern is always the same. The model works fine in a notebook and passes every offline evaluation, then breaks in production, because nobody designed the controls around it. Someone forgot to pin the policy version, someone else assumed the tool would validate its own inputs, and the postmortem reads like the last three.

This book is about making agentic systems *boringly reliable*. You will learn to design, evaluate, deploy, and govern agents that deliver results while keeping risk, cost, and failure modes visible. The aim is accountable autonomy rather than more autonomy: decisions constrained by policy and grounded in evidence, proposals and authority that can be withdrawn when reality changes, and controls before any action that can't be undone, with reconciliation after it.

ⓘ Example

A refund agent that needed a stop button

A support team deploys an agent that can propose and process customer refunds. A request arrives: "Refund the last order. The shipment never arrived." The agent can pull order records and trigger a refund.

Its decision boundary says it may draft the response and compute the refund, but must not move money without a clear order ID, a policy check, and an approval whenever a threshold or a risk flag is hit.

A healthy run takes five steps:

1. Resolve the customer and order ID. Refuse to proceed if identifiers are ambiguous.
2. Retrieve the refund policy and the order timeline: shipment scans, delivery confirmation, prior refunds.
3. Propose a refund decision with a justification and the exact tool call it would make.
4. Execute only if policy checks pass and required approvals are present. Verify the ledger update.
5. Emit a run record: inputs, policy version, tool I/O, and a linkable decision note.

What goes wrong in a run like this? The agent "helpfully" guesses the wrong order when names are similar, or it applies last month's policy after an update, because the policy was cached and nobody versioned it.

The fix is to require stable identifiers in tool schemas, version every policy, gate high-risk cases on approval, and test those edge cases in evaluation. Clean audit logs, refund-rate trends, chargeback rates, and sampled run traces are evidence that the control plane is holding under the conditions seen so far.

The book is domain-agnostic. The refund agent above is the running system the book keeps returning to, because it is small enough to reason about and rich enough to expose the real problems: decision boundaries, stale context, policy drift, human approvals, rollback, and evaluation. In Volume II's last part, its decision packet, not its workflow, carries over to three other domains.

Sample, not for distribution

Note

How to use this book

To stay oriented as you read:

- Pick a reading path: the preface's paths say which chapters each kind of reader needs, in this volume and the next.
- Check the learning objectives at the start of each chapter to see what it delivers.
- Run the listings. Each is a marked region of the companion's tested code, and the output printed beside it is that code's own.
- Find the exercises' answers in the companion repository, which keeps them for most chapters as files of keys and tested reference solutions.
- Follow the evolving project thread, the refund agent. Each part adds one new layer of capability or control, and the last, in Volume II, puts the whole system before a launch gate.

What this book is (and is not)

This book treats an agent as a system of tools, state, policies and controls, and not as a single model. It teaches evaluation-led engineering: what can go wrong, how you detect it, and how you prevent regressions. And it generalizes across industries rather than walking you through one vendor's stack.

If you are coming from a framework tutorial, an ML textbook, or a collection of LangChain notebooks, the difference is scope. The two volumes cover the full lifecycle from design through production operations to governance, so they are neither a framework tour nor a collection of isolated notebooks. The emphasis throughout is on system behavior, governance, and outcomes you can measure. This volume teaches the foundations, from data and evaluation to reinforcement learning, and each method arrives as evidence the refund agent must produce before it may act. Each chapter ends on what that evidence permits and what it does not.

The reference architecture

When a system fails, the fastest way to debug it is to know *where* a behavior lives. This book uses one six-layer map throughout, so that architecture discussions never depend on the framework of the month:

Layer	What it owns
Interface	goals, permissions, typed inputs/outputs
Context	retrieval, memory, state, grounding sources
Reasoning and orchestration	workflows, tool calling, planning policies
Model	classical ML, deep learning, adaptation (fine-tuning, distillation)
Data	pipelines, features, embeddings, labels, governance
Evaluation and operations	tests, monitoring, incidents, rollback

The **control plane** cuts across all six layers: policy gates, approvals, and audit trails. A layer owns a behavior. The control plane owns permission for that behavior to take effect.

Each chapter states which layers it strengthens and which artifacts it adds. In Volume II, Part V builds the core agentic layers, from Interface through Reasoning. Part VI wires in operational infrastructure: Data, then Evaluation and operations. Part VII adds the governance controls that make the whole stack auditable.

Sample - not for distribution

The evaluation spine

Evaluation runs through the entire book rather than living in one chapter. For every capability you build, you will define:

- Failure modes: what can go wrong.
- Metrics: how you know it is working.
- Tests: how you prevent regressions.
- Monitoring: how you catch drift in production.

Each chapter names the metrics, tests, or monitoring signals for the capability it covers. These artifacts are the evidence on which a system earns more autonomy, one grant at a time and under the conditions they test.

Pattern vocabulary to watch for

As you read, watch for a small set of named patterns and anti-patterns. They are there to make design reviews, incident writeups, and rollout decisions easier to talk about. When a team can name a failure mode quickly, it can usually contain it faster too. The names are shorthand for recurring operational mistakes, and for one defense:

Pattern	What it means in practice
Autonomy cliff	the moment a system crosses from suggestion to action without the governance to match
Context amnesia	the system technically has the right evidence somewhere, but retrieval, freshness, or context-budget choices make it act as if that knowledge does not exist
Swiss cheese evaluation	layered evaluation, named after James Reason's Swiss cheese model of system accidents (Reason 2000), where each method has blind spots, so safety depends on combining multiple imperfect checks
Silent cascade	an upstream error that survives every handoff because each downstream component assumes someone else already validated it
Demo-to-prod gap	a system that passes every offline evaluation and then fails in production, because the evaluation never tested real traffic, stale data, tool latency under load, or a busy approval queue

The refund agent's two failures above already have names. Applying last month's policy after an update is context amnesia: the current version existed, but a caching choice made the agent act as if it did not exist. The guessed order becomes a silent cascade when every later step, the refund tool included, assumes the identifier was checked. The book describes four more recurring mistakes in plain words:

- A rule that exists in docs, prompts, or review decks, but not in the execution path that controls behavior.
- A human review queue that still exists on paper but no longer gives decisions the attention that could change them.
- Measurement rituals that produce dashboards but block no release and delay no grant of autonomy, whatever they show.
- A threshold that nobody can explain, so nobody can safely change it or defend it during an incident.

You do not need to memorize the names on first read, only to notice the patterns when they recur. In this volume's chapters only the first, the autonomy cliff, recurs by name, and Volume II's chapters name the other four. By the end of the book, you

Sample - Not for Redistribution

should be able to use the names as a compact diagnostic language, and say "this was a silent cascade" where you would have said "something felt off."

The evolving project

A single project thread ties the book together, so that ideas accumulate into one system. We will build the refund agent in layers, and each layer exists for a reason:

1. **Part I** explains how to write a decision boundary. For the refund agent, the boundary is the one in the example above: it can draft, calculate, and gather evidence, but it cannot move money outside explicit policy.
2. **Parts II and III** add the predictive and representational pieces: a risk score proposed for the gate, policy grounding, retrieval, and the model choices that determine how the system reasons from evidence.
3. **Part IV** shows where learning and policy tuning help, and where they should stop. Not every refund decision needs a learning loop, but every threshold change needs governance.
4. **Part V** (Volume II) turns the workflow into an agentic system with a loop, tools, memory, guardrails, and evaluation.
5. **Part VI** makes the system operable: observability, rollback, deployment, and incident response.
6. **Part VII** adds the human and institutional controls: a monthly audit of denials by segment, an explanation for every denial, a review queue sized for its arrivals, and a rollout plan that widens the agent's authority one signed step at a time.
7. **Part VIII** assembles the refund agent's launch packet and runs its launch gate, asks the packet's questions again in three other domains without copying the refund workflow, and says when a part of the agent is replaced. That assembly waits on a run of simulated operation still to be made. Until then, the capstone sets out what the packet must hold and runs the gate on a packet constructed in the companion.

By the end of Volume II, you should be able to point to one concrete system and explain what changed at each stage, why it became safer, and what evidence justified more autonomy.

Conventions and code policy

Every listing is a marked region of the companion's tested code, printed whole or with its elisions marked. Where a region is a reprint set to the page's width, a test keeps it identical to the package.

By convention, a time-sensitive claim carries a date and a factual claim carries a citation.

Summary

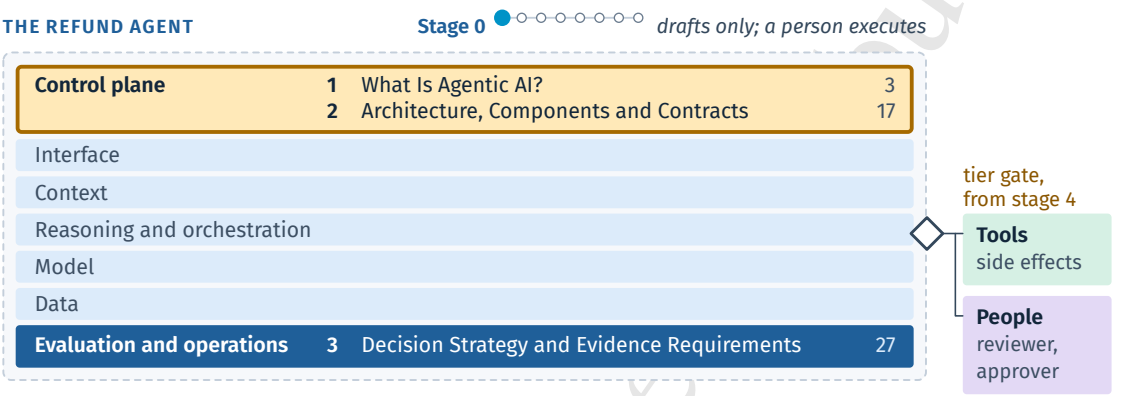
An agent that can act on the world needs more than a good model. It needs contracts, policies, evaluation, and someone watching. Part I starts by defining the decision boundary: what the system is allowed to do, what it must prove before it acts, and what should remain human.

Sample - Not for Redistribution

PART I

Orientation and Agentic Mindset

I



The introduction ended on what has to be settled for a system that can act: what it may do, what it must prove before it acts, and what should stay with a person. Part I settles it for one system, the refund agent, before that system may pay anything. The agent that paid a refund in the introduction's healthy run is where the book is going. Here it begins at what the book calls stage 0 (Figure 1.3 and Table 1.5), where it drafts every decision and a person executes, and the part ends with it still there.

Chapter 1 (*What Is Agentic AI?*) says what kind of system an agent is and how much it may do, on a scale of four levels of autonomy. It then writes the refund agent's decision boundary: what it may decide, what is out of scope, and where a person takes over. Chapter 2 (*Architecture, Components and Contracts*) gives each rule of that boundary a place to live, on one of six layers or on the control plane. The chapter adds a contract where two parts meet and an owner who answers when one of them fails. Chapter 3 (*Decision Strategy and Evidence Requirements*) then settles what a class of decisions must show, signed before the run that judges it, to receive its next grant of authority. By the end the agent has a boundary, a layer map and, for tiers 1 and 2, the evidence each step up will need, and it still may not pay.

Read the three in order, because each starts from the question the one before leaves open. Of the preface's reading paths, only the Sponsor path skips the second. If you have a system of your own in mind, write one paragraph about it now: what it decides, what it must not decide, who owns it, and what would make you roll it back. The chapters give that paragraph a boundary, a map and an evidence requirement in turn.

All three rest on a question that comes first. What makes a system agentic, and how much may it decide?

Sample Not for Distribution

THE ROUTE THROUGH PART I

Orientation and Agentic Mindset

What makes a system agentic, and how much may it decide?

Read the three in order, because each starts from the question the one before leaves open.

1

What Is Agentic AI?

3

YOU CAN NOW

Write a decision boundary and say whether a system needs an agent.

THE REFUND AGENT

Its boundary for stage 0 has seven tools, three tiers and a rule-only baseline whose drafts a person executes.

LEAVES OPEN *Where does each one live?*

2

Architecture, Components and Contracts

17

YOU CAN NOW

Place each part of a system on a layer, name its owner, and trace a change across a contract.

THE REFUND AGENT

The layer map gives each part a row and an owner, and nothing the agent does yet moves money.

LEAVES OPEN *What evidence earns each autonomy level?*

3

Decision Strategy and Evidence Requirements

27

YOU CAN NOW

Write an evidence requirement for any class of decisions and size its run.

THE REFUND AGENT

An evidence requirement for each grant tiers 1 and 2 can take, and tier 1's step to shadow signed.

LEAVES OPEN *What must a score mean before it gates?*

BY THE END OF PART I

The agent has a boundary, a layer map and, for tiers 1 and 2, the evidence each step up will need, and it still may not pay.

Refund agent stage 0 ●○○○○○○○○ *drafts only; a person executes*

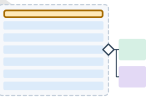
NEXT **Part II** ML Foundations

Sample - Not for Redistribution

CHAPTER

What Is Agentic AI?

13 pages, 1 code listing



Control plane
Refund agent
Part I: stage 0

On 14 February 2024 a tribunal in British Columbia ordered Air Canada to pay a customer $812.02 because the chatbot on the airline's website had given the customer misleading information about a refund (British Columbia Civil Resolution Tribunal 2024). A chatbot had answered a question, and the company that deployed it was held to the answer. So before you let a system answer, decide or act for you, you need to know what kind of system it is and how much it may do. You'll leave this chapter able to say both, to write down what it may decide before it runs, and to tell when it shouldn't be an agent at all.

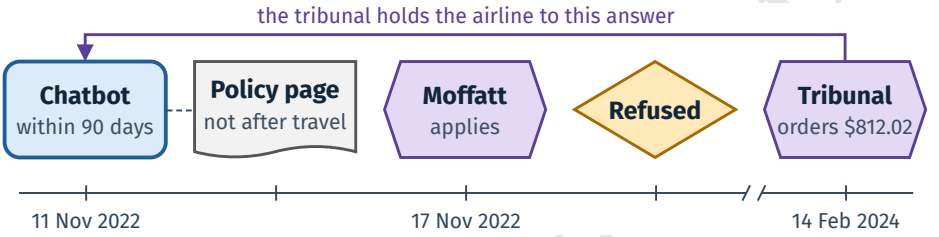


Figure 1.1 The Air Canada case on a time line. The chatbot said the reduced fare could be claimed within 90 days of the ticket's issue, and its answer linked to the airline's own page, which said the policy didn't apply after travel. Moffatt applied, the airline refused, and the tribunal held the airline to its chatbot's answer.

The case began on 11 November 2022, when Jake Moffatt, whose grandmother had died, asked the chatbot on Air Canada's website about bereavement fares (Figure 1.1). The chatbot answered that someone who needed to travel immediately, or had already traveled, could submit their ticket for a reduced bereavement rate within 90 days of the ticket's issue. Highlighted words in the answer linked to the airline's own page on bereavement travel, which said the policy didn't apply to requests made after travel was completed. Moffatt booked flights between Vancouver and Toronto and first applied for the reduced fare on 17 November, the day before flying back.

The airline refused, holding that Moffatt hadn't followed the proper procedure and couldn't apply after traveling. Moffatt took the claim to the tribunal. The tribunal found that the chatbot was part of the airline's website and that the airline was responsible for all the information on it, whether a static page or a chatbot supplied it. It decided the claim as negligent misrepresentation, finding that the airline hadn't taken reasonable care over its chatbot's accuracy and that Moffatt had reasonably relied on what it said. It awarded $650.88 in damages and, with interest and fees, $812.02 in all.

The tribunal decided one small claim, on its own facts.[1] Still, the airline answered for a system it had deployed, so someone there had to understand what it would tell a customer. This book's position goes further: whoever deploys an agent owns what it says and does.

The airline's chatbot answered a question about a refund. The system this book builds, the refund agent, decides refund requests. It is built for the support desk of an online merchant that sells physical goods. Its job is to do one of three things with each request: approve and execute the refund, deny it with an explanation that cites

[1] The decision doesn't discuss its weight beyond this dispute. This chapter draws from it a lesson for design, not a legal rule.

Sample - Manning

the policy, or send it to a person. Unlike the chatbot, it acts on what it decides, once it is allowed to. In my view the whole software industry, and knowledge work with it, is going through a revolution, as more and more intelligent systems are built to perceive, orient, decide, act and learn.[2]

[2] Here "learn" means that outcomes are recorded and used to improve the system, so the model need not train online. Table 1.2 matches these five verbs to the four stages of the loop that Volume II, Chapter 22 builds.

◎ **Learning objectives**

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

1. Describe a system by its environment, percepts, actions and performance measure, and tell an agentic one from a fixed workflow by who chooses the next step.
2. Place each action of a system on four levels of autonomy, and say who decides, who acts, and how the action is undone at that level.
3. Write a decision boundary for one decision: what is permitted, what is forbidden, where approval is required, what is reversible and what is not, the spend ceiling, and the logging that is non-negotiable.
4. Choose between a fixed workflow and an agent for a given decision, and state the condition that would reverse the choice.

1.1 What makes a system agentic

The tribunal's decision never says what kind of system the chatbot was. The airline gave the tribunal no information about its nature. For the refund agent the question can't be left open, and this book answers it in one sentence: *agentic AI* is goal-directed software that uses AI to decide what to do next. Of the opening's five verbs, the definition turns on *decide* and asks one question: who chooses the next step?

In a fixed workflow the designer sets the steps before the first request arrives, and a branch among them follows a rule the designer wrote. In an agent, AI selects the steps at run time, from what it has seen so far. This book judges the two differently, a workflow by whether it applies its fixed rules correctly and an agent by its outcomes under constraints, with evidence that it acted within them. Table 1.1 compares the two on five questions.

Table 1.1 A fixed workflow and an agent, compared on five questions

Question	Fixed workflow	Agent
Who sets the steps	Predefined by the designer, branches included	Selected at run time by the model
How much the work varies	Low	Medium to high
Where judgment lives	Encoded by the designer	Delegated to the model and its policies
How it fails	At brittle edge cases	By unexpected reasoning or action
What governs it	Control of the process	Control of the process and of its autonomy

Which of the two the refund agent is depends on when you ask, because it is built in stages. A *stage* is what it may do after each part of the book. At stage 0, where this part leaves it, a baseline of fixed rules drafts every decision and no model chooses anything, so by the table's first row it starts life as a workflow. Its rules do branch, since a read that fails ends the reading, but the designer wrote that branch too. It becomes an agent only when a model chooses its next step, and later parts add one.

Exercise 1

Sample - Not for Reproduction

When a model does join, it won't be the agent by itself. The agent is the model plus memory, tools, policies, evaluation, and the execution loop that holds them together. Besides the AI that decides what to do next, such a system can use other kinds of AI as tools (predictive, prescriptive, descriptive and others). *Agentic*, then, describes the system around the model, and the refund agent's system is set down in a specification before any model joins it. The specification gives it seven tools (from reading an order to paying a refund) and a record of each request's state. Those tools read a world and change it.

1.2 An agent in its environment

environment

The world an agent reads and acts on is its *environment*. At each step an agent receives an observation from its environment and takes an action (Yao et al. 2023, sec. 2). For an agent that lives in software, the two directions have names. Quoting Etzioni, Lesh and Segal, Wooldridge and Jennings call the commands that return information about its environment its sensors, and the commands that change the environment's state its effectors (Wooldridge and Jennings 1995, sec. 1.1.1, p. 117). Each observation, whether a request that arrives or a sensor's reply, is a *percept*.

percept

The refund agent fits this frame closely (Figure 1.2). Its environment is the systems around it: the order system, the carrier's feed, the store of policy versions, the payment ledger and the people who approve and pay. Its sensors are its three reads, of the order, the shipment and the policy, so its percepts are a request and what those reads return. Its effectors are its four writes, which propose a refund, ask for an approval, pay and keep the audit record, and it calls them only at later stages. In the frame's terms, its one action at stage 0 is the draft it returns after the reads, to approve, deny or refer. The draft only suggests what to do, and a person in that environment acts on it.

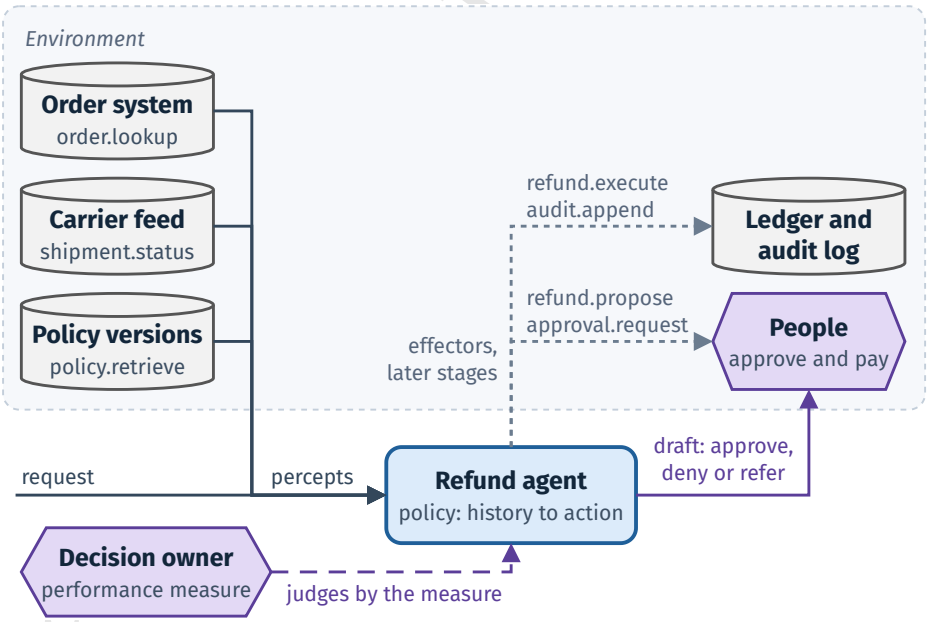


Figure 1.2 The refund agent in its environment. A request and three reads come in as percepts. At stage 0 its one action is a draft returned to a person, and the four writes are the effectors that later stages call. The measure that judges it is held outside, by the decision owner.

for redistribution

That environment is only partly observable. The agent sees a parcel through the carrier's events as the feed last reported them, and the feed can be stale. The environment also changes while the agent works and as it acts. A balance the agent reads when a request arrives can be spent before the refund is paid (Section 1.4), and every refund paid lowers the balance that the next request on that order reads.

performance measure

The frame has one more part, and it isn't the agent's: the *performance measure*, the standard for judging what the agent does.[3] Whoever deploys the agent holds the measure, and the owner of each class of decisions signs it before the run that judges those decisions (Section 3.2). At stage 0 the baseline's drafts are scored by agreement. For each request the data generator in the book's companion repository records the decision the refund policy calls for. A draft agrees when its outcome, amount and tier (the band of amounts that sets who decides) match that decision. A denial agrees only if it also cites the policy as the decision does. Agreement shows whether the baseline applies the policy's rules correctly, which is how this book judges a workflow, and it counts every wrong draft alike. Pricing each kind of mistake is left to the decision's owner, at stage 1.

[3] Russell and Norvig give performance measures a section of their own (Russell and Norvig 2021, sec. 2.2.1).

The measure is only one of the things a deployer has to write down. Before calling something an agent, you should be able to name eight things on one page: its objective, environment, allowed actions, memory policy, tool permissions, evaluation method, stop condition and human escalation path. Read against the frame, the objective and the evaluation method are its measure, stated and then applied, and the environment and the allowed actions are two of its parts. The other four are what a deployer adds, and the decision boundary of Section 1.5 writes two of them for the refund agent, the tool permissions and the path to a person.

None of the eight, though, does the choosing. In practice an agentic system has an objective, context, a policy for choosing actions, tools that create side effects, and feedback that changes future behavior. The policy is the part that chooses. It isn't the refund policy that `policy.retrieve` reads, or the rules on what an agent may do, which a policy contract states (Section 2.2). Section 16.1 calls it a decision policy. Write h_t for the agent's history at step t, the percepts it has received and the actions it has taken until it chooses, and a_t for the action it then takes. The policy π maps each history to an action,

Notation Appendix A

$$a_t = \pi(h_t). \tag{1.1}$$

The action at step t is what the policy gives for the history so far. In general, though, a policy gives each action a probability, as in Appendix A and in ReAct, a method for language-model agents (Yao et al. 2023, sec. 2), and the agent draws its action from those probabilities. The equation as written is the deterministic case, one action for each history, which is what the stage 0 rules give.

Take request r-7-0007, the constructed scenario this part follows, on the companion's data with values as given. When the stage 0 rules choose, its history holds the request for $38 back after a change of mind, and three replies. Order o-0007 was paid $42 with nothing refunded, the parcel was delivered 12 days before the request, and the current version of the refund policy, `refund_cap_v4`, allows 21 days. Which draft does π give for that history? Section 1.5 runs the program on it and prints the draft it gives.

The stage 0 baseline computes its π with rules, each pairing a condition on what has been read with a draft, and tries them in a fixed order. Russell and Norvig's pseudocode calls a program of that shape a simple reflex agent, one that acts by the

Sample – Not for Distribution

rule whose condition matches the current state as the percept defines it (Russell and Norvig 2021, fig. 2.10). For this comparison the request and the replies to its reads count as one percept, the one that defines the state, and the draft the baseline returns is the action. The baseline has no internal model of the world and keeps a state only for the request in hand, in the context layer of Section 2.1.[4] By that frame the stage 0 baseline is already an agent, a simple reflex one, and by this book's definition it is still a workflow, because its designer fixed every rule it applies.

Predict Does the baseline keep any model of the world?

[4] Russell and Norvig's model-based reflex agent has such a model, keeping a state of the world updated from its last action and each new percept (2021, fig. 2.12), and they go on to goal-based, utility-based and learning agents (2021, secs. 2.4.4–2.4.6).

Those rules settle what the baseline does, and decision theory says which action a policy should choose. Give each action a utility $U(a)$, set by the measure and uncertain because it turns on what is true, and pick the action whose utility is highest on average (Berger 1985), as Equation A.15 does in Section A.2. For an agent the average is taken given its history, and the best is sought among the actions it may take, the set $\mathcal{A}$,

$$a_t^* = \arg\max_{a \in \mathcal{A}} \mathbb{E}\big[U(a) \mid h_t\big]. \tag{1.2}$$

The best action at step t, a_t^*, is the permitted action whose utility, averaged over what the history leaves uncertain, is highest. Russell and Norvig's pseudocode for "a decision-theoretic agent that selects rational actions" chooses this way (Russell and Norvig 2021, fig. 12.1). On r-7-0007 at stage 0, $\mathcal{A}$ holds three drafts, to approve, deny or refer, and no call to `refund.execute`, because the boundary of Section 1.5 leaves paying to a person. The baseline's rules return their draft without computing any expectation. Whether that draft is also a_t^* turns on how likely the request is to be sound and on U, that is, on what a wrong payment, a wrong denial and a referral each cost. Section 5.3 works this out with illustrative costs for the decision's owner to replace and sign.

A language-model agent keeps this frame and changes the program. In place of rules a designer fixed, a model reads the history as text in its context and writes the next action (Yao et al. 2023, sec. 2). The history it can use is only what fits in its context window (Section 12.10), and the measure stays outside the model, with the deployer. Table 1.2 sets the frame beside the five verbs of this chapter's opening and the four stages of the agent's loop.

Table 1.2 One agent in three vocabularies

Verb of the opening	The agent in its environment	Stage of the loop
Perceive	A percept, the request or a sensor's reply	Perceive
Orient	Each new percept interpreted in light of the history h_t	Reason
Decide	The policy's choice, $a_t = \pi(h_t)$	Reason
Act	An action, a draft returned to a person or a permitted call through an effector	Act
Learn	Outcomes recorded and used to change future behavior	Learn
No verb	The performance measure, the deployer's	Outside the loop

For a language-model agent, an action through a tool begins as text. In Toolformer a call is written into the model's text as the tool's name and its input, with the

Sample - Not for distribution

result written after it (Schick et al. 2023, sec. 2). In ReAct the agent looks an entity up through a web API by writing `search[entity]` (Yao et al. 2023, sec. 3.1). Until a program runs it, the call is only text. The program around the model reads it, checks it against what the tool accepts and against the decision boundary, and runs it or refuses it.[5] Either way, what comes back is the agent's next observation (Yao et al. 2023, sec. 2), so the reply or the refusal joins h_{t+1}. A check of this kind can refuse a call outside $\mathcal{A}$, but it can't tell whether a permitted call is the best one, which is a question for U and so for the deployer.

[5] Section 12.11 shows what such a call is to the model. Volume II, Chapter 22 builds the program that turns the refund agent's calls into actions, and Volume II, Chapter 23 writes down what each tool accepts and promises.

How much the program lets the agent do is a separate matter, and its name, autonomy, needs care, because Wooldridge and Jennings use the word in another sense. They call an agent autonomous when it operates without the direct intervention of humans or others and has some control over its own actions and internal state (Wooldridge and Jennings 1995, sec. 1.1.1).[6] In this book the word means how much a system may do once it has decided, its level of autonomy, and the definition of *agentic* leaves that out on purpose. Put autonomy into the definition, and the label would switch on and off. At the same point in its rollout, the refund agent would be agentic for a refund of $49.99 and not for one of $50.00. So autonomy is a second question, and the side effects of what a system does make it a serious design problem.

[6] Their definition names four properties of an agent: autonomy, social ability, reactivity and pro-activeness (Wooldridge and Jennings 1995, sec. 1.1.1).

Choosing which step comes next is one moment, and acting on the world is another. Once the system is choosing actions, product design, risk management, and accountability all move. Once it moves from suggestion to action, policy, tracing, and rollback stop being optional. A system can reach the first moment long before the second.

1.3 Four levels of autonomy

Between the two, this book uses a scale of four levels of autonomy, each described by who decides, who acts, and how a wrong action is undone. At the first level, an agent helps a person on request and shows what to pay attention to, and the person approves or rejects the outcome. At the second, the agent works continuously and makes recommendations that people examine, then adapt or reject, which fits when it lacks context people haven't yet entered into any system. At the third, it proposes the next best actions, and people approve, reject or modify each one. At the fourth, it decides and executes, and people see its actions in an audit log and can reverse them. My company, OODARIS AI, builds agents at these four levels, and its first-level agent is a conversational one that helps people run financial scenarios and what-if analyses.[7]

[7] Many of those scenarios are projections, with no data and no decisions behind them yet, so the agent's job is to help a person simulate and decide.

A person can reject what the agent puts forward at both the second and third levels. This book tells the two apart by who carries out the action: the person at the second level, and the agent at the third once the person approves. By this book's reading, the airline's chatbot worked at the first level, since it answered the customer who asked and the customer decided what to do with the answer. Even at that level, the airline was held to what its chatbot said. Table 1.3 puts the four levels side by side.

Exercise 2

Which level fits depends on the use case, the risks, the domain, and how the agent's decisions and actions will affect businesses, people and the environment. For the refund agent it depends on the refund. Its requests fall into three tiers by amount (under $50, $50 to $500, and over $500), and Figure 1.3 shows each tier's level at each stage.[8] Every tier starts at level 2, because at stage 0 the agent drafts and a person

Sample - Not for Redistribution

Table 1.3 Four levels of autonomy, from OODARIS AI, with this book's reading of who acts

Level	What the agent does	What a person does
1	Helps on request: runs scenarios and shows what to watch	Asks, then approves or rejects the outcome
2	Works continuously and recommends	Examines each recommendation, adapts or rejects it, and carries it out
3	Proposes the next best actions	Approves, rejects or modifies each one before it takes effect
4	Decides and executes	Reads the audit log and reverses what is wrong

executes. Request r-7-0007, for $38, is tier 1 by its amount, and at stage 0 it gets a level 2 draft that a person decides on and carries out. Only tier 1 ever reaches level 4, and tier 3 never leaves level 2. No tier is ever at level 1, because a customer's request starts the agent and no one on the staff asks it for a scenario.

[8] The figure includes two trials that later parts define: shadow, in which the agent decides and its payment is recorded but not made, and a canary cohort, the first group of requests whose tier 1 refunds it pays.

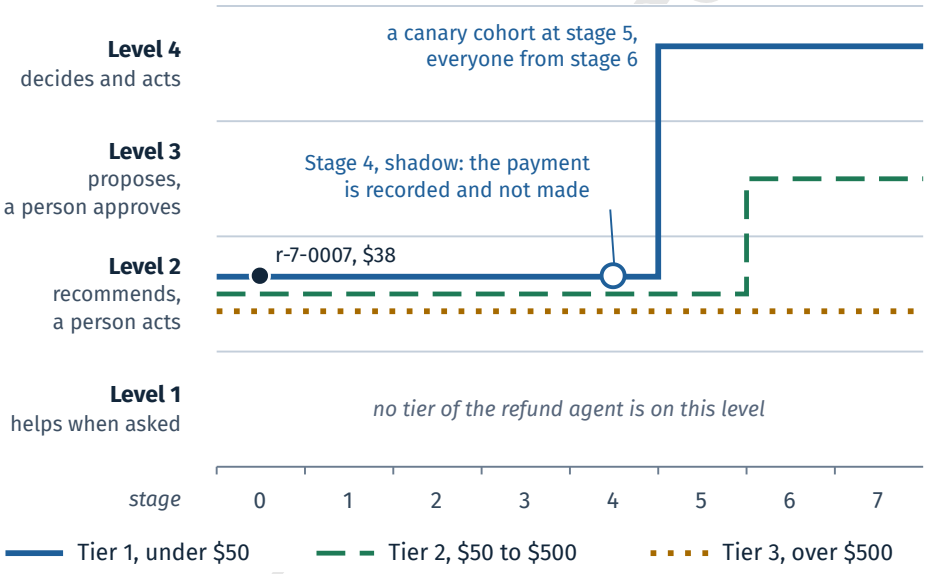


Figure 1.3 The refund agent's level of autonomy at each stage, by tier.

Each step up the scale changes what has to be governed. Action begins at the third level, where the agent carries out what a person has approved. So a mistake the approver misses is already an operational event, and approving each action is the governance. The move to the fourth removes that approval, so governance has to go elsewhere: into the audit log, the means of reversal and the evidence that earns the move. This book uses *autonomy cliff* for the moment a system crosses from suggestion to action without the governance to match. A system that makes either move without its governance goes over the cliff.[9]

autonomy cliff

[9] The term is the book's own, and it returns in *Human Oversight and Risk Management* (Volume II, Chapter 37) and *Decision Strategy and Evidence Requirements* (Chapter 3).

The refund agent's tier 1 reaches level 4 because the agent decides and executes there with no approval. It gets there without the level's means of reversal, since none of its

Sample, not for distribution

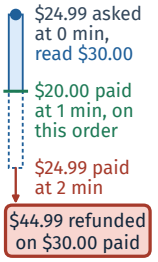
tools takes back a paid refund. Reversibility belongs to the action, as part of its risk, and no level supplies it, so two things stand in for it: a limit on the money and a check before the payment. The payment is kept under $50, so one wrong payment is small. The same mistake repeated over many requests is not, and Volume II, Chapter 32 caps how much the canary cohort may be paid in a day. A gate in the payment call, which Volume II, Chapter 25 builds, limits the harm before it happens. Together they keep tier 1 off the cliff. Volume II, Chapter 37 takes up how a person oversees each level, and Chapter 3 what evidence earns each step, by the rule that autonomy should rise only as evidence rises.

Evidence earns each step, but what lets a system climb the scale is trust, and trust has a condition. Humans need to trust these systems before letting them be fully autonomous, and gaining that trust requires understanding how the systems are built, what they will do, how they make decisions, and how their actions and decisions can be tuned, reverted and defended.

1.4 If agents do the work, why learn how they work?

Many people would say nobody needs to. In their view agents will take care of the work, so there's no need for people, or for data, or for machine learning, operations research, deep learning and reinforcement learning. On that view the foundations needn't be learned, and coding agents can build anything without human oversight. The view deserves a fair hearing, because this book rests on a belief of mine that starts from the same premise: agents will do the data science, and the scarce skill is understanding, directing, checking, and explaining their work. The dispute is over what follows.

I see what follows in some of my students. They skip the foundations, hand the work to coding agents, and then can't get the best out of them. A coding agent can produce a notebook or a small application quickly, but along the way the work calls for many assumptions, decisions and authorizations. Each of them needs judging, and passing tests can't stand in for that, because code, whoever writes it, can pass its tests and be wrong.



The $24.99 request was paid on a balance it read on arrival, after another refund had spent $20.00 of it. The checks are now made at execution.

An early build of the companion's code for the refund agent, which already paid refunds into a simulated ledger, passed 151 tests. A review of that code then ran one more case, in which a refund of $24.99 was requested on a $30.00 order and another refund of $20.00 was paid on the same order a minute after the request arrived. A minute after that the request was executed and paid, leaving the order refunded $44.99 against $30.00 paid. The code had checked the balance as it stood when the request arrived, and no test had caught it. The fix is a rule the specification now states: the checks are made when the refund is executed. In the terms of Section 1.2, the code paid on a percept that had gone stale while the request waited, and the rule makes the agent gather what its action depends on at the moment it acts.

Someone at the company that deploys the system has to catch an error like that before real money moves, and calling the system an agent doesn't change that. Air Canada argued that it couldn't be held liable for information given by its agents, servants or representatives, the chatbot included. The tribunal read the argument as suggesting that the chatbot was a separate legal entity, responsible for its own actions, and rejected it (British Columbia Civil Resolution Tribunal 2024, paras. 27–28).

Answering for an agent takes understanding, from the people who use it as well as those who build it. In many cases its users would want to know why it made a

Sample - Not for Distribution

decision. Its practitioners have to be able to architect it, evaluate and debug it, give it guardrails and tools, secure it, and put it into production for people to use. All of them need to understand how it was built and how it decides, and that takes knowledge of the foundations. If you do not learn and know all these concepts and foundations, I would not hire you.

A practitioner draws on the foundations to evaluate an agent, debug it, and give it guardrails and tools. Improving my team's first agent at OODARIS AI, a conversational one at the first level, took that kind of work. My team collected the many ways users ask it for things and the mistakes it made, added evaluations, better tools and guardrails, revised its instructions, and added decision models trained to classify what the user intends and to check and improve its responses. Evaluations and guardrails like these need a target: a statement of what the agent may decide, against which this book assesses it.

1.5 A decision boundary for the refund agent

That statement is the refund agent's first artifact, its *decision boundary*, which defines what the system can decide, which actions are out of scope, and when a human must take over. A useful one names six things: what is permitted, what is forbidden, where approval is required, what is reversible and what is not, the spend ceiling, and the logging that is non-negotiable.

At stage 0 the refund agent may read the order, the shipment and the policy, and draft a decision, but it may not pay, because a person executes. Any refund of $50 or more needs approval, and anything outside the lowest tier's conditions (fresh evidence, a current policy version and a risk score below its threshold) goes to a person. A drafted proposal can be withdrawn. A paid refund can't. The spend ceiling is the one the agent will work under once it pays. Its own payments stay under $50, and while it pays only the canary cohort, a daily cap bounds the total (Volume II, Chapter 32). The logging rule is one audit record per decision, linked by a trace identifier.

The agent acts through the seven tools of Table 1.4, one of which writes that audit record. Three are reads, which return evidence, and three write records without moving money, while the seventh, `refund.execute`, moves money and can't be undone.[10]

[10] The three reads have no tier. The policy check isn't one of the seven but the gate in the payment call that Volume II, Chapter 25 builds.

Table 1.4 The refund agent's seven tools

Tool	Effect	What it does
`order.lookup`	Read	Returns the order, its items, the payment and prior refunds
`shipment.status`	Read	Returns the carrier's events, with the age of the data
`policy.retrieve`	Read	Returns the policy text, with its version and effective date
`refund.propose`	Write, reversible	Creates a proposal with an amount, a reason and evidence
`approval.request`	Write	Opens a review item for a person
`refund.execute`	Write, irreversible	Pays the refund
`audit.append`	Write	Writes one record per decision, linked by trace identifier

The approval rule guards that payment and rests on the three tiers of Table 1.5, which say who decides at each amount and on what conditions. Part II proposes a calibrated risk score for tier 1 and tests it offline, in Section 5.5, but the companion's code keeps a placeholder rule in its place.

Exercise 3

Sample - Not for Redistribution

Table 1.5 The refund agent's three tiers, with the authority each holds once its evidence is in. At stage 0 a person decides at every tier

Tier	Refund amount	Who decides	Conditions
1	Under $50	The agent	The evidence is fresh, the policy version is current, and the risk score is below its threshold
2	$50 to $500	The agent proposes, and one person approves	An approval is recorded before anything is paid
3	Over $500	A named approver	The agent prepares the packet only

Whatever the tier, at stage 0 the draft comes from the rule-only baseline, which in the companion repository is the function `baseline`. Follow it on request r-7-0007, the $38 asked back on order o-0007 in Section 1.2.

The listing is that function from `refund_agent/baseline.py`, set to the page's width, and the companion's tests keep the two identical. For each request it makes the three reads in a fixed order and stops at the first that fails, referring the request with that read's reason. Otherwise `assess` applies the fixed rules to what was read. Finally the approval rule turns an approval at tier 2 or 3 into a referral for approval, since only tier 1, `Tier.AGENT` in the code, is the agent's to decide. Read it as Equation 1.1 written out for stage 0, a policy in which every branch is a rule its designer fixed. From the companion repository's root, `python demos/what_is_agentic_ai_demo.py` runs it on r-7-0007, with the values as given, and on a second request. The demo prints amounts in whole cents, as the companion keeps them.

Predict Which draft does the baseline return for r-7-0007?

Code refund_agent/baseline.py

```python
def baseline(
    data: Dataset, requests: Sequence[Request] | None = None
) -> list[Decision]:
    tools = Tools(data)
    decisions = []
    for request in data.requests if requests is None else requests:
        state = State(
            request=request, trace_id=f"baseline-{request.request_id}"
        )
        for name in ("order.lookup", "shipment.status", "policy.retrieve"):
            result = tools.call(ToolCall(name=name), state)
            if not result.ok:
                state.decision = decision(state, Outcome.REFER, result.reason)
                break
        chosen = state.decision if state.decision is not None else assess(
            data, state
        )
        if chosen.outcome == Outcome.APPROVE and chosen.tier != Tier.AGENT:
            chosen = decision(state, Outcome.REFER, "approval_required")
        decisions.append(chosen)
    return decisions
```

OUTPUT

```
r-7-0007 asks 3800 cents for a change of mind
order.lookup: o-0007 paid 4200 cents; refunded 0 cents
shipment.status: delivered 12 days before the request
policy.retrieve: refund_cap_v4 version 2026-03-01; window 21 days
draft: approve, tier 1, eligible
cites refund_cap_v4@2026-03-01#eligibility
audit records: 0
r-7-0000: order.lookup failed: ambiguous_order (2 matching orders)
draft: refer, tier 1, ambiguous_order; no citation
```

Sample Not for Distribution

```
audit records: 1; decision refer, ambiguous_order
draft kinds: 3 (approve, deny, refer); write tool calls: 0; executions: 0
```

The first seven lines follow r-7-0007, from the request through its three reads to the draft. The draft approves because every condition holds: the delivery falls inside the policy's window for a change of mind, the evidence is fresh and the policy current, the amount is within the policy's cap and what the order has left to refund, and the risk score is low. That score comes from the placeholder, a fixed rule that looks for the substring "ignore" in the request's text, regardless of case. A claim that the parcel never arrived would turn on the carrier's events instead of the policy's window, and the baseline denies one when they show a delivery. The draft cites the eligibility section of `refund_cap_v4`, and a person reads it and pays. Over both requests the baseline creates no proposal or review item and executes nothing, as the last line's counts show.

The second request, r-7-0000, is one of the companion's forty public scenarios, generated with one of the specification's injected failures: its order identifier matches two orders. The lookup fails, the baseline stops reading, and it returns a draft sending the request, with the reason, to a person. That referral leaves the one audit record of the two runs. The code that runs the tools writes it itself when a read fails, so the last line counts no call to a write tool. The baseline doesn't yet meet the boundary's logging rule, since a draft that settles a request, like the first one, leaves no audit record. Volume II, Chapter 22 adds the record. If fixed rules can draft every one of these decisions, when is an agent worth building?

Exercises 5 and 6

1.6 When to use a workflow and when to use an agent

Workflows win when the path is known. Agents earn their keep when the path has to be discovered. When a system can be easily implemented and verified, there is no need to make it agentic if regulations apply, or if the decisions you need from it are simple, predetermined and deterministic and the client it is built for does not intend to change its approach or process.

Exercise 4

Stage 0 of the refund agent has a known path, since fixed rules set every step, so it is a case for the workflow. Five conditions favor one: a stable process, explicit decision rules, data of a predictable shape, costly mistakes, and output that has to be deterministic. At least the last four hold here. The rules are three tiers at two thresholds, and the data is an order, a shipment and a policy. A mistake costs money, and the same request should get the same draft every time.

Five conditions favor an agent: context that changes faster than rules can be maintained, a task that needs search and synthesis and tools, a path that depends on what the agent finds along the way, common exceptions, and experts who already improvise the workflow. In a constructed scenario, a bank's fraud analysts follow leads from record to record, each step set by what the last one showed. The baseline follows a fixed plan of three reads for every request and stops if one fails. Measured against the five, the case for making the refund agent an agent is still open. The specification as it stands shows one of the five, search and synthesis, because later in the book the agent drafts from policy text it retrieves by version and freshness. The specification may show a second. Its test data injects failures, such as an order identifier that matches two orders or a stale carrier feed, and those are exceptions, though nothing says how common they are.



A known path: three reads in a fixed line, one branch to a person. A discovered path: each lead set by the last record.

Sample – Not for Redistribution

Even where the rule seems settled, two conditions reverse it, one in each direction. The first, a known path that changes faster than the team can maintain the workflow, favors an agent. The second, an unknown path where a wrong action cannot be undone or reviewed in time, does not. That second condition is the reason for the refund agent's tiers, and Section 3.1 shows how it and the size of a wrong payment set the level each tier may reach.

By the rule, then, the baseline is rightly a workflow, and this chapter specified it as one. Nothing in the book so far shows an agent deciding refunds better than those fixed rules. That is the test an agent has to pass. Compare the agent with the simpler alternatives, from the manual workflow to the rules engine, and if it isn't better on the business dimension, not the demo dimension, don't ship it. In Volume II, Chapter 29 and Volume II, Chapter 39, that comparison sets the agent's decisions beside the baseline's on the same requests (Section 3.4). This book's own advice can end in an agent built anyway, when the baseline already does the job. I count systems that are too complicated, and overengineering up front for simple tasks, among the places where that advice fails, and this book is not immune to either.

1.7 Summary

An agent differs from a fixed workflow in who chooses the next step. Either is described by its environment, percepts, actions and performance measure, and by a policy that maps each history to an action. The rational action is the permitted one of highest expected utility. Four levels of autonomy run from helping on request to deciding and executing under an audit log. A workflow fits a known path, unless the path changes faster than the workflow can be kept up. An agent fits a discovered one, unless a wrong action on it can't be undone or reviewed in time. The refund agent's boundary for stage 0 has seven tools, three tiers and a rule-only baseline whose drafts a person executes. On paper it binds nothing, since each rule is a responsibility that some part of the system has to carry. Where does each one live?

1.8 Exercises

1. (Check, objective 1.) Who chooses the next step in each constructed scenario, rules the designer fixed in advance or AI at run time, and what in the description settles it, if anything? (a) A grid balancer picks each action from the readings of the last few seconds, and a person overrides it when the grid becomes unstable or a constraint is violated. (b) A code assistant uses a model to pick each next file, change and test from what the last step found, and a person reviews its work. (c) The stage 0 baseline applies the same reads and rules, in the same order, to every request, and stops reading when a read fails. (d) Describe the stage 0 baseline by its environment, percepts, actions and performance measure, and say who holds the measure.
2. (Check, objective 2.) Place each action on a level of Table 1.3, and say who decides and how a wrong action is prevented or undone. (a) A planner asks an agent how cash flow changes if a supplier is paid late. (b) An agent recommends discharges for a ward, and the charge nurse edits the list and arranges them. (c) An agent applies a server change once an engineer approves it. (d) An agent blocks card payments it judges fraudulent, and the bank's team lifts wrong blocks from an audit log.
3. ★ (Apply, objectives 2 and 3.) For three requests, values as given ($24, $180 and $1,250, each meeting every other condition of its tier), use Table 1.5 and Figure 1.3 to give the tier, who decides, the level of autonomy at stage 0, and the highest level the tier ever reaches. Which tools in Table 1.4 are classified as reads, and so have no tier, and which one does the stage 0 boundary forbid?
4. ★ (Apply, objectives 3 and 4.) Apply the rule of Section 1.6 and its reversing conditions to each constructed scenario, and

Sample. Not for Redistribution

name the clause that decides. (a) A hospital assigns transfers to wards by an audited protocol, revised yearly, on bed type, isolation needs and specialty. (b) A bank's fraud analysts follow leads from record to record. The patterns change every few weeks, faster than their playbook, and freezing an account stays with a person. (c) Suppose the bank of (b) builds an agent to follow the leads. Write its decision boundary for one decision, whether to freeze an account, naming the six things of Section 1.5.

5. ★★ (Analyze, objectives 1 and 3.) These eight constructed stage 0 drafts include faulty ones. Values are as given, unmentioned eligibility conditions hold, and "reviewer" means an ordinary tier 2 reviewer. Which drafts break the boundary of Section 1.5, and which condition does each break? (1) $42, all fresh: approve. (2) $42, shipment data older than its freshness limit: approve. (3) $50.00, all fresh: approve. (4) $120, all fresh: proposal to one reviewer. (5) $640, all fresh: proposal to one reviewer. (6) An identifier matching two orders: sent to a person. (7) $30, a cached policy version that is no longer current: approve. (8) $30, policy text older than its freshness limit: sent to a person.

6. ★★★ (Build, objective 3.) With seed 7, run the companion's rule-only baseline on the forty public scenarios, then lower the tier 1 threshold in `tier_for` from 5,000 to 2,500 cents and run it again. Deliverable: the diff and each run's count of drafts sent to a person. Acceptance: `exercises/what-is-agentic-ai/test_acceptance.py` passes. It checks the second run's counts and the new tier boundary, and that the run calls only the three reads and executes no refund.

Sample - Not for Redistribution