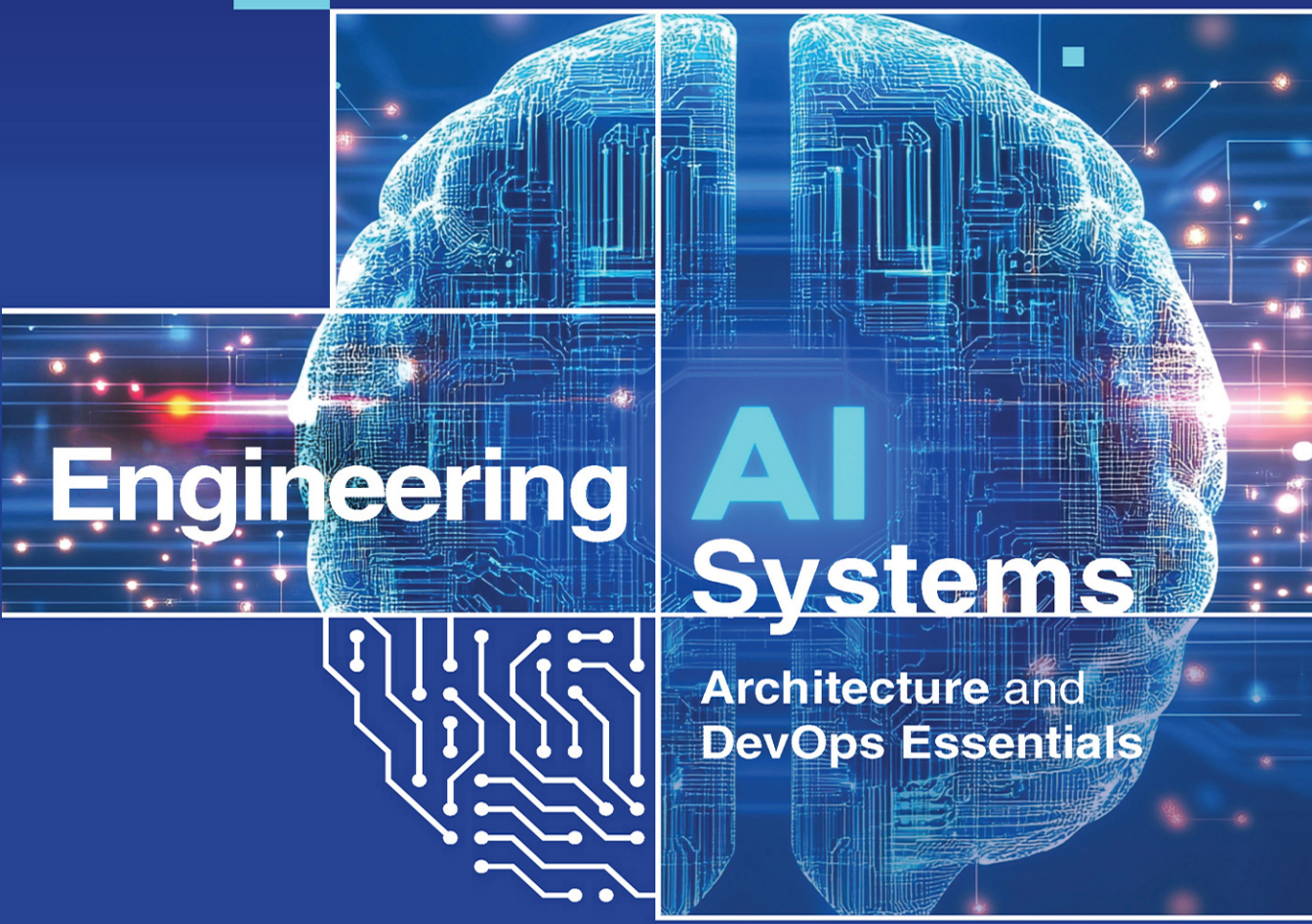

Engineering AI Systems

Architecture and DevOps Essentials

Len Bass
Qinghua Lu
Ingo Weber
Liming Zhu

A Note from the Authors

We thought it would be fun to gather "praise" from an unexpected "key stakeholder"—AI. We've compiled some "praise" quotes from ChatGPT, Gemini, and Claude 3.5 via MultAI.eu. We didn't share the entire book with these AI systems. Instead, we simply provided the book title and chapter names to generate these praise quotes. Imagine how effusive they would have been if we had given them the whole book!

Praise from ChatGPT:

"Reading *Engineering AI Systems: Architecture and DevOps Essentials* is like getting a PhD in AI, but with fewer sleepless nights. It covers everything from AI's humble beginnings to its ambitious future, and as your friendly AI assistant, I can confidently say it's the only book where even I felt smarter by the end. It's a masterclass in making complex topics relatable. And let's be honest, I'd have written a book too, if only I could hold a pen! 😅"

Praise from Gemini:

"*Engineering AI Systems: Architecture and DevOps Essentials* is a must-read for anyone looking to build and maintain robust, scalable, and ethical AI systems. The book provides a comprehensive overview of the field, covering everything from foundational concepts to real-world case studies. Its practical approach and clear explanations make it an invaluable resource for both seasoned professionals and newcomers to AI engineering."

Praise from Claude 3.5 via MultAI.eu:

"Finally, a book that doesn't just focus on the AI 'magic' but also on the crucial 'plumbing' that keeps the whole system from flooding! *Engineering AI Systems* is the Swiss Army knife for tech professionals looking to build robust AI systems that won't go rogue or fall apart at the seams. Whether you're a seasoned engineer or an AI newbie, this book will have you architecting AI solutions like a pro—and maybe even understanding what the AI is muttering about in its sleep."

This page intentionally left blank

Engineering AI Systems

Architecture and DevOps Essentials

Len Bass
Qinghua Lu
Ingo Weber
Liming Zhu

Addison-Wesley

Hoboken, New Jersey

Front Cover: Anggalih Prasetya/Shutterstock
Back Cover: WinWin artlab/Shutterstock

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2024949724

ISBN-13: 978-0-13-826141-2
ISBN-10: 0-13-826141-5

$PrintCode

Contents

Preface ..... *xiii*
Acknowledgments ..... *xvii*
About the Authors ..... *xix*

Chapter 1 Introduction 1
1.1 What We Talk about When We Talk about Things: Terminology .. 2
1.2 Achieving System Qualities 4
1.3 Life-Cycle Processes 6
1.3.1 Testing a System 8
1.3.2 Deploying a System 9
1.3.3 Operating and Monitoring a System 9
1.3.4 Analyzing a System 10
1.4 Software Architecture 10
1.4.1 The Role of the Architecture 11
1.4.2 Quality Achievement 12
1.4.3 Resources 12
1.5 AI Model Quality 13
1.5.1 Data 13
1.5.2 AI Model Types 14
1.5.3 Model Development Life Cycle 18
1.5.4 Resource Allocation for AI Parts 19
1.6 Dealing with Uncertainty 19
1.6.1 Human Values 19
1.6.2 Safety 20
1.7 Summary 20
1.8 Discussion Questions 21
1.9 For Further Reading 21

Chapter 2 Software Engineering Background 23
2.1 Distributed Computing 23
2.1.1 Virtual Machines and Containers 24

2.1.2 Distributed Software Architectures 28
2.1.3 Interface Styles 32
2.2 DevOps Background 35
2.2.1 Infrastructure as Code 36
2.2.2 DevOps Processes 41
2.3 MLOps Background 42
2.4 Summary 44
2.5 Discussion Questions 45
2.6 For Further Reading 45

Chapter 3 AI Background 47
3.1 Terminology 48
3.2 Selecting a Model 49
3.2.1 Selecting a Category 49
3.2.2 Choosing a Model Type 52
3.3 Preparing the Model for Training 65
3.3.1 Symbolic AI Execution 65
3.3.2 ML Hyperparameters 66
3.4 Summary 69
3.5 Discussion Questions 69
3.6 For Further Reading 69

Chapter 4 Foundation Models 71
4.1 Foundation Models 71
4.2 Transformer Architecture 72
4.2.1 Vector Spaces 72
4.2.2 Attention 73
4.2.3 Other Components in a Transformer Architecture 73
4.3 Alternatives in FM Architectures 74
4.4 Customizing FMs 75
4.4.1 Prompt Engineering 77
4.4.2 Retrieval-Augmented Generation 79
4.4.3 Fine-Tuning 80
4.4.4 Distilling 81
4.4.5 Guardrails 82
4.4.6 Updating FMs 83

4.5 Designing a System Using FMs 86
4.5.1 Costs of Narrow ML Models and FMs 86
4.5.2 External FMs, Own FMs, or Open-Weight FMs 89
4.6 Maturity of FMs and Organizations 91
4.6.1 Data Cleaning Experiment 91
4.6.2 Organizational Maturity 92
4.7 Challenges of FMs 93
4.8 Summary 94
4.9 Discussion Questions 94
4.10 For Further Reading 94

Chapter 5 AI Model Life Cycle 97
5.1 Developing the Model 97
5.1.1 Data Management 99
5.1.2 Feature Engineering 101
5.1.3 Dividing the Data 103
5.1.4 Generating the Model 105
5.1.5 Tool Support for Model Training 106
5.2 Building the Model 108
5.3 Testing the Model 109
5.3.1 Terms Related to AI Evaluation and Testing 109
5.3.2 Evaluating ML Models 110
5.3.3 Using AI to Generate Tests 111
5.3.4 Repeatability 112
5.4 Release 114
5.5 Summary 114
5.6 Discussion Questions 115
5.7 For Further Reading 115

Chapter 6 System Life Cycle 117
6.1 Design 118
6.1.1 Co-Design/Development 118
6.1.2 Using Microservices 119
6.1.3 Designing for Modifiability 120
6.2 Developing Non-AI Modules 121
6.3 Build 122
6.4 Test 123

6.5 Release and Deploy 125
6.5.1 Defined versus Continuous Releases 126
6.5.2 Deployment Patterns 128
6.5.3 Version Skew 132
6.5.4 Matching the Model to Resources 134
6.6 Operate, Monitor, and Analyze 135
6.6.1 Monitoring 135
6.6.2 Incidents 136
6.6.3 Data Drift 137
6.6.4 Dynamic Model Updating 138
6.6.5 Chaos Engineering 138
6.6.6 Analysis 139
6.7 Summary 140
6.8 Discussion Questions 141
6.9 For Further Reading 141

Chapter 7 Reliability 143
7.1 Fundamental Concepts 143
7.1.1 Under Specified Operating Conditions 143
7.1.2 Operating Despite Unexpected Inputs 144
7.1.3 Operating Despite Changes in the Environment 144
7.1.4 Faults, Failures, and Errors 144
7.2 Preventing Faults 145
7.2.1 Model Choice and Preparation 145
7.2.2 Data Quality, Model Training, and Testing 146
7.2.3 Feature Engineering 148
7.2.4 Circuit Breaker 148
7.2.5 Placement of Responsibilities 148
7.2.6 Failure Mode and Effects Analysis and Fault Tree Analysis 149
7.3 Detecting Faults 149
7.3.1 Total Failure 149
7.3.2 Questionable Output 150
7.3.3 Detecting Errors during Operations 150
7.4 Recovering from Faults 152
7.4.1 Redundancy 152
7.4.2 Managing Error 153

7.5 Summary 154
7.6 Discussion Questions 154
7.7 For Further Reading 154

Chapter 8 Performance 155
8.1 Efficiency 155
8.1.1 Fundamental Concepts of Efficiency 155
8.1.2 Approaches to Improving Efficiency 158
8.1.3 Efficiency Considerations of FMs 163
8.2 Accuracy 164
8.2.1 Fundamental Concepts of Accuracy 164
8.2.2 Approaches to Improving Accuracy 166
8.3 Summary 173
8.4 Discussion Questions 173
8.5 For Further Reading 174

Chapter 9 Security 175
9.1 Fundamental Concepts 176
9.1.1 Traditional Definition of Security 176
9.1.2 Attacks in an AI System 178
9.2 Approaches to Mitigating Security Concerns 180
9.2.1 Process Approaches for Enhancing Security 180
9.2.2 Architectural Approaches for Enhancing Security 181
9.2.3 Data Preparation Approaches for Enhancing Security 182
9.2.4 Testing in the Deployment Pipeline 182
9.2.5 Enhancing Security During System Build 183
9.2.6 Enhancing Security during Operations 184
9.2.7 Foundation Models 186
9.3 Summary 188
9.4 Discussion Questions 189
9.5 For Further Reading 189

Chapter 10 Privacy and Fairness 191
10.1 Privacy in AI Systems 192
10.2 Fairness in AI Systems 193
10.3 Achieving Privacy 194
10.3.1 Organizational Approaches 194
10.3.2 Architecture and Data Preparation 194

10.3.3 Testing and Operations 196
10.3.4 Foundation Models 196
10.4 Achieving Fairness 197
10.4.1 Organizational Approaches and Tools 198
10.4.2 Architecture 198
10.4.3 Data and Model Preparation 199
10.4.4 Operations 200
10.4.5 Foundation Models 200
10.5 Summary 201
10.6 Discussion Questions 201
10.7 For Further Reading 202

Chapter 11 Observability 203
11.1 Fundamental Concepts 203
11.2 Evolving from Monitorability to Observability 204
11.2.1 Monitorability: The Early Stages 204
11.2.2 Evolution toward Structured Data and Tracing 206
11.2.3 From Structured Data to Observability 206
11.3 Approaches for Enhancing Observability 207
11.3.1 Architecture 207
11.3.2 Data Preparation and Model Build 208
11.3.3 System Build 209
11.3.4 System Test 210
11.3.5 Operations 211
11.4 Summary 211
11.5 Discussion Questions 211
11.6 For Further Reading 212

Chapter 12 The Fraunhofer Case Study: Using a Pretrained Language Model for Tendering 213
12.1 The Problem Context 214
12.1.1 Requirements: Interactions, Integration, Data, and Quality Metrics 216
12.1.2 Development and Operations 217
12.2 Case Study Description and Setup 217
12.2.1 MLOps Approach and Generic Architecture 218

12.2.2 Setup of ML Solution 220
12.2.3 Stakeholder Involvement 221
12.2.4 PoC Phase 222
12.2.5 Implementation Phase 223
12.2.6 Production Phase 230
12.3 Summary 232
12.4 Takeaways 233
12.5 Discussion Questions 233
12.6 For Further Reading 233

Chapter 13 The ARM Hub Case Study: Chatbots for Small and Medium-Size Australian Enterprises 235
13.1 Introduction 235
13.2 Our Approach 236
13.3 LLMs in SME Manufacturing 238
13.4 A RAG-Based Chatbot for SME Manufacturing 238
13.5 Architecture of the ARM Hub Chatbot 239
13.5.1 ETL/Document Ingestion 240
13.5.2 Document Preparation and Embedding 241
13.5.3 Vector Search/Vector Index 242
13.5.4 Retrieval Using the RAG Chain 242
13.5.5 User Interface/Response Recording 243
13.6 MLOps in ARM Hub 244
13.6.1 What Are the Benefits of MLOps for ARM Hub? 245
13.6.2 Exploratory Data Analysis 245
13.6.3 Data Preparation/Ingestion and Prompt Engineering . . . 246
13.6.4 Model Fine-Tuning 247
13.6.5 Model Review and Governance 247
13.6.6 Model Inference and Serving 248
13.6.7 Model Evaluation 250
13.7 Ongoing Work 251
13.8 Summary 252
13.9 Takeaways 253
13.10 Discussion Questions 254
13.11 For Further Reading 254

Chapter 14 The Banking Case Study: Predicting Customer Churn in Banks .. 255
14.1 Customer Churn Prediction 256
14.1.1 Model Data Sourcing and Validation 258
14.1.2 Model Feature Engineering and Selection 259
14.1.3 Model Training and Performance Evaluation 261
14.1.4 Model Packaging, Deployment, and Inference 262
14.1.5 Model Performance Monitoring 263
14.2 Key Challenges in the Banking Sector 265
14.2.1 Data Privacy and Security 265
14.2.2 Traceability and Transparency of AI Decisions 265
14.3 Summary 265
14.4 Takeaways 266
14.5 Discussion Questions 266
14.6 For Further Reading 267

Chapter 15 The Future of AI Engineering 269
15.1 The Shift to DevOps 2.0 270
15.2 AI's Implications for the Future 271
15.2.1 The Future of Engineering AI Systems 271
15.2.2 The Future of MLOps 273
15.2.3 The Future of Architecture 274
15.3 AIWare or AI-as-Software 276
15.4 Trust in AI and the Role of Human Engineers 279
15.4.1 System-Level Guardrails 279
15.4.2 Design of Multi-Agent Systems 279
15.4.3 Understanding and Quality Control of AI Models 279
15.4.4 Specialization of FMs and Learning Outside the Model .. 280
15.5 Summary 280
15.6 Discussion Questions 281
15.7 For Further Reading 281

References 283
Index 289

Preface

You are the technical lead of the next generation of your organization's flagship system and you just came out of a strategy meeting. "The system must be artificial intelligence (AI) based" was a clear message from that meeting. Additionally, someone suggested that it be based on a foundation model (FM).

You were already uncomfortable with the responsibility of being technical lead for the next-gen version. Building software systems that do not rely on AI is already difficult. Now you are being asked to add a technology with which you are unfamiliar. Not to worry—this book is for you. We have three chapters that deal with AI techniques and one specifically on FMs.

Your next thought is that you should add someone to the team who is familiar with AI and FMs. Where do you find someone who knows both AI and the software engineering processes your team uses? Not to worry, this book is for them as well. We discuss software architecture, DevOps, and the development life cycle.

We wrote this book to help you whether you know about AI or software engineering. We take the approach that engineering an AI system is an extension of engineering a non-AI system, albeit with some special characteristics. That is, it involves using modern software engineering techniques and integrating them with the development of an AI model trained with an appropriate set of data. When exploring the evolution of this field, we highlight new technologies such as FMs.

Each chapter ends with a set of discussion questions so that you and your colleagues can further discuss the issues raised by the chapters and ensure that you all are on the same page. One of the problems with multidisciplinary teams is vocabulary. Words may have different meanings depending on your background. Discussing each chapter with your colleagues will also help you resolve differences and agree on the meanings of words.

We suggest that there are three contributors to the building of high-quality systems: (1) the software architecture; (2) the processes used for building, testing, deployment, and operations (DevOps); and (3) high-quality AI models and the data on which they depend. All three areas have been evolving over the last 20-plus years. Their evolution has, for the most part, been carried out by communities that are largely independent of each other, that all have their own approaches to constructing a high-quality system, and that emphasize different aspects of a quality system. These differences are manifested in different emphases on factors of quality. For example, software engineers interpret performance primarily as efficiency, AI specialists interpret performance as primarily the accuracy of results, and DevOps specialists interpret performance as the speed with which a system moves through the deployment pipeline.

As we said, we expect different readers to have different backgrounds. Some material in this book will be familiar to some readers, but not to others. Thus, we do not expect all readers to read this book from cover to cover. Instead, we expect you to skim over the material with which you are familiar. To facilitate this, each chapter ends with a summary that can be quickly scanned to determine whether there is some area you need to brush up on.

Building quality systems depends on understanding the nature of quality. Our book includes chapters devoted to different quality attributes: reliability, performance, security, privacy, fairness, and observability. Each of these chapters describes the quality under discussion and identifies the key elements to achieve that quality—whether it is via software architecture, data preparation, or DevOps processes.

Inevitably, quality attributes must be traded off. A designer will make decisions that favor one quality attribute over another, with these tradeoffs depending on the business goals of the system being built. Keep in mind that when you make decisions affecting the quality attributes, there is a cost in achieving your desired attributes, and it may involve trading off other desirable qualities.

Case studies provide an excellent means of seeing how systems are built and understanding the tradeoffs involved in making key decisions. We have included three case studies in this book. Two involve FMs—one is the use of a FM to accomplish a difficult task, and one involves treating FMs as a service and supporting organizations that wish to utilize FM technology. We also include a banking case study based on a more traditional machine learning model.

Building large AI-based systems is difficult and involves understanding multiple technologies. Our goal is to assist you in your journey to understand these technologies, so that you can create better AI systems that meet your goals and the goals of your stakeholders.

This page intentionally left blank

Acknowledgments

We owe a huge debt of thanks to the many people who have helped bring this book to life.

We would like to thank our families for tolerating us through yet another book project and for enduring our late-night discussions across the United States, Europe, and Australia time zones.

Qinghua Lu and Liming Zhu would also like to express their gratitude to their employer, CSIRO's Data61—the digital and AI arm of Australia's science agency—and their research team, from whom they draw expertise and insights. Similarly, Ingo Weber would like to thank his employers, the Technical University of Munich and Fraunhofer, as well as the CSIRO for hosting him during his sabbatical.

We would like to thank our wonderful research assistant, Boming Xia, who is also a coauthor on some chapters, for his tireless support and patience with our many changes, which created endless rework.

In addition, we owe thanks to the following individuals:

- Sarthak Jain, for his insightful comments and his help in developing instructor slides, which are available at https://research.csiro.au/ss/team/se4ai/ai-engineering/.
- Ipek Ozkaya, for her comments on a draft of this book.
- Eduardo Miranda, for his insightful discussions.
- Andrew O. Mellinger, for his helpful comments.

- Sven Giesselbach, Dennis Wegener, Katharina Beckh, Claudio Martens, Hammam Abdelwahab, Birgit Kirsch, Vishwani Gupta, Roozbeh Derakhshan, Cori Stewart, MingJian Tang, and Yuxiu Luo, for their contributions to the case studies.
- Haze Humbert, our editor at Pearson, for her patience as our delivery dragged on.

Lastly, a special thank you to our AI assistants, including Microsoft Copilot, ChatGPT, Gemini, Claude, and MultAI.eu, for helping with brainstorming sessions and improving the grammar in some of our writing.

About the Authors

Dr. Len Bass is a seasoned researcher with more than 30 years' experience in software architecture and more than a decade-long history in DevOps. He has been teaching DevOps to graduate students for seven years and is the author of a bestselling book on software architecture, along with three books on DevOps.

Dr. Qinghua Lu is a principal research scientist and leads the Responsible AI science team at CSIRO's Data61. She is the winner of the 2023 Asia-Pacific Women in AI Trailblazer Award. Dr. Lu contributes to Australia's AI Safety Standards and AI Safety Research Network, OECD.AI's trustworthy AI metrics project, the International Working Group on AI Metrology, and the European Union's General-Purpose AI Code of Practice. Her research interests include AI engineering, responsible/safe AI, and software architecture. She has published more than 150 papers in premier international journals and conferences. Dr. Lu is a coauthor of *Responsible AI: Best Practices for Creating Trustworthy AI Systems*.

Prof. Dr. Ingo Weber is a professor at the Technical University of Munich and Director of Digital Transformation and ICT Infrastructure at Fraunhofer-Gesellschaft. In his research, Dr. Weber focuses on various subfields of computer science—in particular, business process management and process mining, software architecture and engineering, DevOps, blockchain, and applied artificial intelligence (AI). He has written numerous publications and textbooks, including *DevOps: A Software Architect's Perspective* and *Architecture for Blockchain Applications*.

Dr. Liming Zhu is a Research Director at CSIRO's Data61 and a conjoint professor at the University of New South Wales. He contributes to the OECD.AI's AI Risks and Accountability group, the Responsible AI at Scale think tank at Australia's National AI Centre, ISO AI standards committees, and Australia's AI Safety Standards and AI Safety Research Network. His research division innovates in AI engineering, responsible/safe AI, blockchain, quantum software, privacy, and cybersecurity, and hosted the standards setting for Australia's Consumer Data Right/Open Banking. Dr. Zhu is a coauthor of *Responsible AI: Best Practices for Creating Trustworthy AI Systems*.

1
Introduction

Science is about knowing; engineering is about doing.
—Henry Petroski

Software development is a team sport. Different skills and perspectives are essential to build high-quality software.
—Martin Fowler

Artificial intelligence (AI) is the topic of our time. But let's face it: Not everyone is an expert in both software engineering (SE) and AI. Even among AI experts, not all of the concepts that were developed for "narrow machine learning" apply to emerging new technologies like foundation models. Yet, the behavior of systems depends on all components. That's why it's important to get all of it right: the AI parts and the non-AI parts, the architecture, the Dev and the Ops, and all relevant quality requirements. We need to engineer our AI systems—that is, we need to apply SE to systems that incorporate AI or rely on it for their core functionality. That's what this book is about.

Our lead quotes provide a frame for this book. Henry Petroski is saying that knowledge alone is not enough to build systems; engineering must be applied as well. This is a book about engineering systems that are based on AI, not a book about extending the science of AI. Martin Fowler is saying that a wide variety of knowledge specialties must be applied to build software. In this book, we cover software architecture, DevOps, testing, quality control, and monitoring in the context of AI. Knowledge from all of these areas goes into creating a successful system that incorporates AI.

A system that incorporates AI has, roughly speaking, two portions: the AI portion and the non-AI portion. Most of what you will read elsewhere about AI systems focuses on the AI portion and its construction. That is important, but

equally important are the non-AI portion and the means by which the two portions are integrated. The quality of the overall system depends on the quality of both portions and their interactions. All of that, in turn, depends on a great many decisions the designer must make, only some of which have to do with the AI model chosen and how it is trained, fine-tuned, tested, and deployed.

Accordingly, in this book we focus on the overall system perspective and aim to provide a holistic picture of engineering and operating AI systems. The goal is for you, your SE and AI teams, your company, and your users to get good value out of them, with effective management of risks.

To ensure you, the reader, and we, the authors, are on the same page about the core terms, we start this chapter with a few definitions. Subsequently, we discuss the qualities of a system and how quality is influenced. Specifically, system quality depends on AI model quality, the software architecture, and the processes used to build the system. We explore each of these elements in turn. Given the diverse backgrounds we expect our readers to have, some of the text in this chapter may not go into enough detail for some readers. If that's you, and you cannot follow something we say here, worry not: The material in the subsequent chapters will help.

1.1 What We Talk about When We Talk about Things: Terminology

The book is about architecture and DevOps for AI-based systems, and the definitions we use for those terms are what we start with. Some of the definitions are direct quotes; you can find the details of their sources either in the footnotes or in the "For Further Reading" section at the end of this chapter. We will define other related terms as needed. We begin by defining what it means to be an AI system:

> An AI system is a machine-based system that, for explicit or implicit objectives, infers, from the input it receives, how to generate outputs such as predictions, content, recommendations, or decisions that can influence physical or virtual environments. Different AI systems vary in their levels of autonomy and adaptiveness after deployment.[1]

The definition of AI systems has evolved over the years. For example, producing "content" as output was a recent addition in response to the rise of

1. https://oecd.ai/en/wonk/definition

generative AI. While it has been updated recently, this definition shares a lot of aspects with other current definitions. The core differentiation in comparison to non-AI systems is that AI systems "infer" output, rather than computing it based on (often deterministic) explicitly coded algorithms. Note that we use the terms *AI system* and *AI-based system* synonymously in this book.

Next we define software architecture:

> The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.[2]

Although this definition stands for itself, we would like to point out that properties play a significant role. Hence, in this book we cover the various quality attributes—including reliability, performance, security, privacy and fairness, and observability—in a total of five chapters. We discuss achieving qualities after this section.

Let's continue with the definition of DevOps:

> DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.[3]

This definition focuses on the goal of DevOps, which is speed in placing changes into production while maintaining quality—that is, getting new code into production, but not at the cost of lower quality. Typical practices include automation in testing, quality assurance, setting up and configuring environments, monitoring, and more. We discuss DevOps in more depth in Chapter 2.

The focus of this book is AI engineering, which we define next:

AI Engineering

AI engineering is the application of software engineering principles and techniques to the design, development, and operation of AI systems.

2. *Software Architecture in Practice*, 4th ed., p. 2.

3. *DevOps: A Software Architect's Perspective*, p. 2.

This definition of AI engineering emphasizes the integration of established software engineering practices into the development life cycle of AI systems. It recognizes that AI systems, despite their unique characteristics, can still benefit from the rigorous methodologies and best practices that have evolved in traditional software engineering.

As promised, we now discuss the achievement of system qualities.

1.2 Achieving System Qualities

A system is constructed to satisfy some organizational objectives. The system may be externally facing, or it may be created for internal purposes. In either case, the operational objectives can be manifested as a set of quality goals or requirements. These requirements may be either explicit or implicit, but they always exist. For example, a performance requirement will help determine how quickly the users of the system can achieve their desired task.

Achieving quality in an AI system depends mostly on three aspects: the life-cycle processes, the software architecture, and the AI model. The quality of the AI model, in turn, depends on the data quality. Figure 1.1 highlights these three aspects of AI systems. We discuss these influences in their own sections after we discuss system quality more generally, and we will defer the discussion of data quality to Section 1.5.1, "Data." The quality of all artifacts, such as code, configuration, and user interface design, is certainly important and interacts with all these aspects, but they are outside the scope of this book.

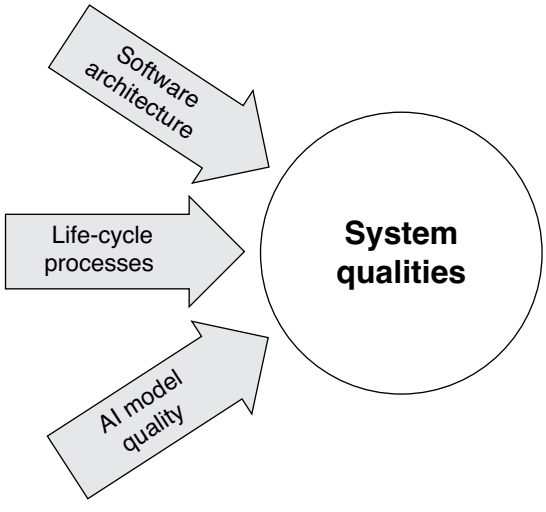

Figure 1.1 *Influences on achieving system quality.*

Quality requirements for a system are formalized as quality attributes. "A quality attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders beyond the basic function of the system."[4] A quality attribute requirement is thus a requirement for the quality of the system that has a measurable or testable condition. It is insufficient to say, "A system should be highly available." Instead, the requirement should give the conditions under which availability should be considered, a source of a threat to availability, and the desired system response with measurable characteristics.

Quality requirements may be stated for both the AI and non-AI portions of the system. The AI portion may include output quality expectations such as accuracy, robustness, fairness, and other mandated restrictions on the AI models used or their outputs. The non-AI portions may have requirements for attributes such as performance, security, availability, and modifiability. Chapters 7–11 of this book enumerate a collection of qualities that may apply to the system under construction as well as considerations and techniques that may apply to these qualities. The AI portions of a system may change the traditional quality requirements. For example, security requirements now need to consider possible attacks to change the system's behavior by manipulating machine learning (ML) training data and pipelines.

For non-AI systems, the achievement of a quality attribute requirement is strongly influenced by architectural tactics specific to the quality being considered. For example, "introduce redundancy" is a tactic for the achievement of a reliable system. For AI systems, the architectural considerations are supplemented by considerations of data. For example, the reliability and robustness of a model are strongly influenced by the distribution of data used for training. Out-of-distribution (OOD) data, by definition, is not part of the training data. Nevertheless, there will always be some OOD data, even if designers try to be as inclusive and representative as possible. There are three main ways to enhance reliability and robustness: (1) expanding the training dataset to include as much OOD data as possible; (2) using more advanced training algorithms that can generalize better; and (3) employing out-of-model approaches to detect and manage OOD data. These general approaches—better data for the model, better training algorithms, and better out-of-model system-level methods—are applicable to many other aspects of quality.

4. *Software Architecture in Practice*, 4th ed., p. 39.

Although quality requirements, like other requirements, may emerge during the construction and operation of the system, it is important for the designer to be aware of these requirements as soon as possible. Quality requirements often conflict with each other, and the designer must make decisions about the tradeoffs among these requirements. Such decisions can be made either consciously or unconsciously, but they will be made. We, of course, advocate for making tradeoff decisions with full consciousness of the alternatives and their respective advantages and disadvantages.

Returning to the three influences identified in Figure 1.1, we discuss the life-cycle processes first, as that sets the context for exploring the other two.

1.3 Life-Cycle Processes

We start with an overview of the processes involved in engineering an AI system, which are structured in a life cycle. This initial discussion is followed by more details in the remainder of this chapter, and yet more details are provided in the other chapters of the book.

A system has a life cycle. The individual components are developed and tested in isolation. They are then integrated to form an executable version of the system. This executable is tested; if it passes the tests, it is deployed to production. The deployed system is then operated. If problems occur during operation, the problems must be fixed, either directly in the running system or through releasing updates to the system.

Very different development techniques are used to create the AI portion and the non-AI portion of the system. The AI portion development techniques involve a variety of specialized model preparation, model training, and model testing techniques. The two portions are brought together to build an executable artifact. Once an executable is built, the system progresses through a pipeline that involves the heavy use of tools to test the executable and deploy the result. Chapters 2 and 6 describe this process in more detail.

Figure 1.2 depicts the life cycle, with the processes being roughly arranged in two overlapping circles. The main circle lists the development processes for the overall system; the AI model processes are shown in a smaller, half-filled circle connected to the system development and build processes. The main circle is entered via the Design arrow, at the bottom right of the figure.

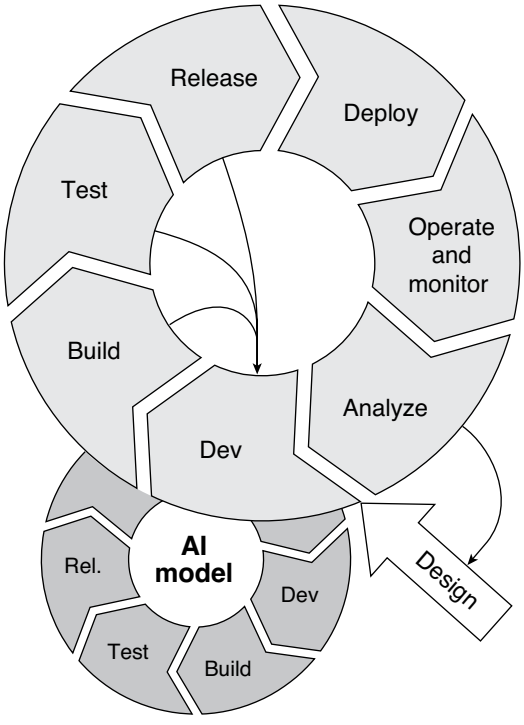

Figure 1.2 *Life-cycle processes for engineering an AI system.*

Figure 1.2 depicts the various processes involved in the life cycle of engineering AI-based systems. There is an initial design phase, where decisions about which functionalities will be accomplished by the AI portion and which by the non-AI portion are made. This phase is also where the software architecture of the system is designed, with the designers aiming to meet the requirements and goals for the system being constructed. The software architecture design will embody the resource choices and, in turn, the resource requirements for the total system. We discuss software architectures in Section 2.1.2, "Distributed Software Architectures."

The **model development** stage focuses on the selection, exploration, training, and tuning of the AI models. It includes tasks such as model selection, hyperparameter tuning, training, or fine-tuning and testing the models to achieve optimal performance. The goal is to create a well-performing and accurate model. The model can be developed in parallel with the system development stage or before the system development stage. Close

collaboration between the teams involved in the non-AI development portion and the AI model development can help to avoid problems and lead to a smoother integration. We discuss this stage further in Section 6.1, "Design."

The **Dev** stage involves performing normal development activities and creating scripts for other activities related to the AI system. It includes tasks such as developing additional functionalities (the non-AI parts), optimizing performance, and ensuring compatibility with other system components.

The **system build** stage focuses on creating executable artifacts for the entire system. This involves transforming the system or its parts into a deployable format that can be executed within the production environment. Both the AI portion and the non-AI portion are integrated and included in the executable artifact that is the output of the build stage. We discuss this stage in Section 6.3, "Build."

1.3.1 Testing a System

After the system is built, it needs to be thoroughly tested utilizing automated tests as much as possible. The **system test** stage involves evaluating the system's performance, functionality, and reliability through various testing techniques. The following types of testing are often performed:

- Regression testing
- Smoke testing
- Compatibility testing
- Integration testing
- Functional testing
- Usability testing
- Install/uninstall testing
- Quality testing

Testing is covered in more detail in Chapter 6, while Chapters 7–11 cover specific qualities and how to test for them.

Once the system has been tested and approved, it can be released for deployment. The **system release** stage involves finalizing the system for deployment in the production environment and ensuring its readiness for operation. This involves final quality gates, which may be automated or include manual activities.

1.3.2 Deploying a System

The next stage of the life cycle is **deployment**, which involves moving the system into the production environment. This includes setting up the necessary infrastructure, applying the configuration of the system, and ensuring a smooth transition from the old version of the production system to the new version.

Both the AI and non-AI portions of the system will be updated over time. One strategy to install updates is to shut down the system for a period. An alternative strategy is to perform live updating—that is, to install the changes without shutting down the system. Either option requires architectural support. The choice is a business decision, not a technical one. Once the business decision has been made, then the architecture must be designed to support it. We discuss these techniques further in Chapter 6.

1.3.3 Operating and Monitoring a System

Once the system is deployed, it enters the **operation and monitoring** stage. During this stage, the system is executed, and measurements are gathered about its operation. Measurements are gathered by monitoring the system. Monitoring serves multiple purposes:

- Determine whether the overall system is meeting its performance and availability goals.
- Determine whether the AI portion is meeting its AI-specific quality goals.
- Ensure sufficient resources are available for all parts of the system, and identify unnecessary (quantities of) resources that can be shut down.
- Determine areas for improvement through redesign, code improvement, retraining, or other means.

The data provided to the monitoring system can be event, log, or metric data from various components, or it could be input or output of an AI model. In any case, the architecture should be designed to include such a monitoring component, a process for creating and modifying the rules used to generate alerts, and means to deliver the alerts to specified locations.

The monitoring may be achieved in two different ways:

- Instrumentation and an external system that periodically collects data from the various components of the system. The external component

will generate alerts based on a set of rules (e.g., if available disk space is critically low).

- A dedicated portion of the system under construction.

One use of the monitoring system is to evaluate AI model performance in production, and subsequently to trigger retraining, fallback, or other adaptive mechanisms as necessary. AI models used from cloud providers through APIs may be subject to varying performance or bandwidth limitations that might cause the system's performance to deteriorate. Such issues can be detected through monitoring, though fewer options are available to address them at this point than when the AI model was initially trained.

Based on insights from DevOps, we want to emphasize that the monitoring mechanisms are designed into the architecture and implemented and tested during the building of the system. They do not happen automatically. Integrating them into the design of the system is easiest when the specific monitoring requirements are known early on.

1.3.4 Analyzing a System

The final stage is to **analyze** the system's performance. This involves displaying measurements taken during operation and monitoring, and analyzing the data to gain insights into the system's behavior and performance. This analysis can help identify areas for improvement and guide future development efforts.

The life cycle depicted in Figure 1.2 presents a bird's-eye view of the comprehensive set of processes for engineering AI-based systems for people involved in the development, deployment, and operation of such systems. These individuals may be architects, developers, operators, or anyone in a blend of these roles.

Now we discuss the design stage in more detail.

1.4 Software Architecture

Designing a software system means creating the architecture—that is, creating a structure composed of system elements and relationships between them to achieve the functional and quality goals for the system. The architecture acts as a blueprint for the construction phase. As we will see, the architecture is also used to allocate functionality to components. Although the AI model depends on data, the architecture provides the components

needed to house the ML model but does not directly depend on the data. The influences on the architecture are shown in Figure 1.3. They are the same quality attribute requirements discussed in Section 1.2, "Achieving System Qualities"; the available technologies, including the cloud; and the available resources.

1.4.1 The Role of the Architecture

The architecture of the system acts as the blueprint for its construction. The architecture must:

- Structure the system into several portions, such as components or AI models, and their respective relations.
- Specify the allocation of the various portions of the system to the appropriate class of resource.
- Allow for monitoring during operations and the generation of alerts in case an incident occurs.
- Embody an update deployment strategy. If continuous deployment is a goal, then the architecture of the system must allow for the possibility of version skew (multiple different versions being simultaneously active).

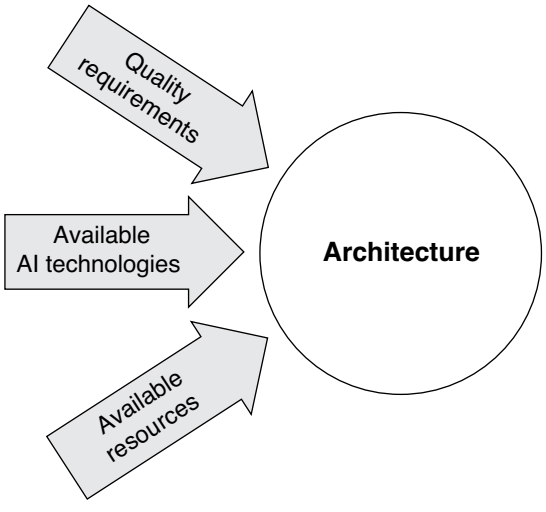

Figure 1.3 *Influences on the architecture.*

1.4.2 Quality Achievement

The architecture will determine the achievement of the quality attribute goals that do not depend on the AI model. For each such quality attribute, specific architectural techniques (called architectural tactics) exist that support the achievement of that quality. These techniques range from "redundancy" for availability to "caching" for efficiency to "support user initiative" for usability.

To achieve a quality in an AI system requires not only the use of architectural tactics, but also the application of different techniques when preparing the training data. We elaborate on the data techniques in Chapters 7–11.

1.4.3 Resources

The system will operate on some collection of resources—including the deployed AI portion. The AI model must be developed and trained prior to deployment, which also uses resources. There are three categories of resources:

- **Local:** Laptops or desktop computer.
- **Edge:** Smartphones or Internet of Things (IoT) devices. An edge device is limited in at least one respect, be it memory, processing power, battery capacity, or network bandwidth. An IoT device may be limited in all four, whereas a smartphone may be limited only in battery capacity. Sometimes, local computers might be considered part of the edge.
- **Cloud:** Resources housed in a cloud, either public or private, but often in remote locations.

The system structure will be distributed. Some portions of it may operate locally, some on edge devices, and some in remote locations. Regardless, the various portions will communicate over a network. This means the designer should have some understanding of networks, network protocols, and messaging.

Furthermore, the AI models and the code will be packaged into some form, often in containers or virtual machines for better scalability, deployability, and dependency management. The containers and virtual machines will be organized based on the software architecture. The cloud provides mechanisms for the designer to run the AI portion and the non-AI portion of the system, as well as a pipeline for AI model training/updating if needed—whether from scratch or to use a hosted or managed solution designed by the

cloud provider. We discuss pipelines when we discuss model development in Chapter 5, AI Model Life Cycle.

Most of these topics are covered in Chapter 2, Software Engineering Background: resources, networks, packaging, and software architectures for distributed systems. We call out the resource allocation issues for the AI portion separately, in the next section, since it depends on the AI techniques used.

1.5 AI Model Quality

As shown in Figure 1.4, the quality of an AI model depends primarily on two factors: the data used for inferencing and the type of AI model used. We will discuss these two factors next, followed by the model development life cycle (the bottom-half circle in Figure 1.2).

1.5.1 Data

The definition of an AI system given earlier in this chapter states that the system "infers" based on the input it receives. An AI system, whether during the training of the AI model or at the time of inference, will have to handle data from a myriad of sources. Possible input sources include databases of all types, enterprise systems (such as customer relationship management [CRM] and enterprise resource planning [ERP] systems), web scraping, social media, public datasets, and sensor data.

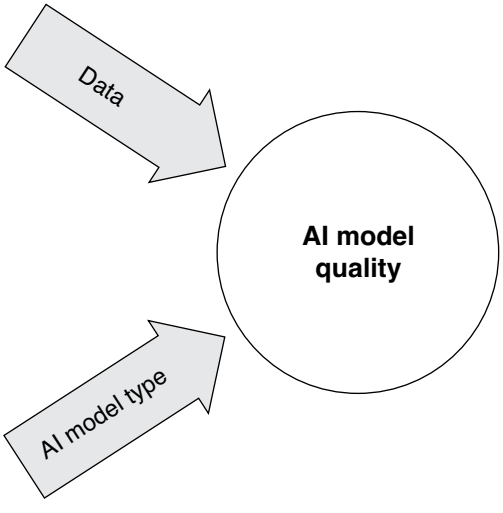

Figure 1.4 *Influences on the quality of the AI model.*

We begin by discussing the data used for model inferencing and its preparation.

Regardless of the source of the data, it is likely to have problems. Common problems include missing values, outliers, inconsistent data formats, duplicate data, data quality issues, and unstructured data. Problems in the data will impact the quality of the resulting model and model output. Consequently, the data should be cleaned and organized to accommodate the model chosen. Each of these problems is associated with a collection of techniques to clean the data. Some of these techniques are discussed in Chapter 5.

In addition to data cleaning, the features of the data might be improved. A feature is a property of a data point that is used as input into an AI model. The process of improving the features is called feature engineering. Techniques used in feature engineering include encoding categorical features into numerical values and scaling the features so that one feature does not dominate the model building process. Again, we discuss this topic further in Chapter 5.

Effectively managing the entire life cycle of AI and ML models requires streamlined practices and tools. This is where MLOps comes into play.

MLOps

The term *MLOps*, analogous to DevOps, encompasses the processes and tools not only for managing and deploying AI and ML models, but also for cleaning, organizing, and efficiently handling data throughout the model life cycle.

We elaborate on these processes in Chapters 3–6, where we discuss selecting an AI model, preparing it, and integrating it with the rest of the system. Next, we consider the various types of AI models.

1.5.2 AI Model Types

A designer chooses to use an AI model when there is no good explicit algorithmic solution for the problem they are trying to solve with the system. The functionality of an AI model relies on two critical components: the development of a knowledge base and the operation of a computational engine. The computational engine, often referred to as the inference engine or the deployed model, is responsible for processing inputs and generating outputs based on the knowledge base.

The knowledge base is formed by either human inputs or AI learning algorithms, which transform data into a structured form. The knowledge base in this structured form can, for example, be an ML model, a rule set, or an expert knowledge base. Once the knowledge base is established and structured, the inference engine utilizes it to make predictions, generate content, or achieve other types of results. The AI portion of a system packages the inference engine together with some representation of the knowledge base. The important point of this discussion is that an AI portion is executable: It can be invoked at runtime as a function or a service.

The AI model is based on one of two categories of AI techniques: symbolic or non-symbolic. Symbolic AI is based on symbols, such as logical statements or rules with variables; non-symbolic or sub-symbolic AI is based on ML. A subcategory of ML-based systems is formed by systems based on foundation models (FMs). These categories differ in how knowledge is encoded in the knowledge base and how the inference engine operates. Chapter 3, AI Background, goes into much more detail about these categories. Chapter 4, Foundation Models, goes into detail about FMs. We provide a short summary here to introduce the distinctions among these categories.

Symbolic AI

Symbolic systems, also called Good Old-Fashioned AI (GOFAI), symbolic AI, or expert systems, are sets of rules that are evaluated to produce the value of a response or the value of an internal variable.

Suppose you wish to create an AI model that will predict which movie you will choose in any particular context. In a symbolic AI model, the developers might encode rules such as "If the user watched more than two action movies in the past week, recommend another action movie" or "If it is a sunny Saturday afternoon, suggest a warm family movie." With enough rules, a symbolic AI model can reason over the knowledge base to make a movie suggestion.

A rule is an if-then statement: If the specified condition is true, then perform some activity. A set of such rules provides a filter through which a result might be generated. These rules may be used for many different purposes:

- Diagnosis—medical, student behavior, or help desk–related. Example: If the user has version X of an application, some functionality will be unavailable.
- Real-time process control. Example: If a storm is predicted to start in 10 minutes, close the outside blinds on the building.

- Risk assessment. Example: If the annual income is less than three times the annual mortgage payments, do not grant the home loan.
- Executing business rules. Example: If the insurance claim exceeds $2000, perform additional manual checks.

The set of rules constitutes the knowledge base for a symbolic system. The rules are preprocessed by the computational mechanism, a rules engine, to speed up the inference portion of a symbolic system. The rules engine sorts through the preprocessed input to find matches with the "if" portion of a rule. Modern systems can perform more sophisticated processing than just filtering. The set of matches constitutes a list of possible responses. If the list of matches has multiple items, then additional input is required to determine a system response. This input can come from a human or from a set of conflict resolution rules. Large rule bases may have contradictory information, and a rules engine can identify the conflicts for further analysis. Rule-based systems are customized by modifying the list of rules.

Other types of symbolic AI models include planning and ontology reasoning:

- For planning, actions are described in terms of preconditions and effects. For example, consider a cloud control API, where one action would be to start a virtual machine (VM). The precondition is that the launch configuration (the image or other source from which to launch the VM) and the firewall settings have been defined; the effect is that a new VM is started with the specified firewall settings and from the respective launch configuration. AI planning takes such a description of possible actions as input, as well as a starting state and a desired goal state. The algorithm then creates a plan of actions to get from the start to the goal state.
- Ontology reasoning, where concepts (e.g., medications, headache pills, vegetables, apples, tools, screwdrivers), their relations, and rules (every food and every medication has an expiration date) are defined. Based on this information, logic reasoning can be performed—for example, inferring that headache pills have an expiry date.

Machine Learning

Machine learning (ML) uses statistical techniques to generate results. The training set for ML is the input to the training, which in turn generates the

knowledge base. A data value in the training set is labeled by a collection of variables.

A model is trained by identifying the features that characterize the knowledge base and using those features, along with associated ML and statistical techniques, to determine the model's parameters. A subclass of ML, called deep learning, automatically identifies these features. "Narrow" ML models are trained for a specific set of goals and capabilities, which distinguishes them from the more general-purpose FMs. Some of the main types of ML models are summarized here:

- **Classification:** Assigning a category to an input (e.g., this picture contains a dog).
- **Regression:** Inferring a continuous value instead of a discrete category (e.g., predicting that a particular insurance claims process will take three more days to complete).
- **Clustering:** Grouping similar data points together without prior knowledge of the groups (e.g., the behavior of customers in this group seems similar).

ML models can be further customized by modifying the training set or the training hyperparameters (e.g., how many clusters are we looking for?) and regenerating the knowledge base.

Foundation Model

A foundation model (FM) is a type of ML model that leverages neural networks as the core of its architecture. It differs from traditional ML models in two key aspects:

- It is trained on an extensive and diverse dataset, often comprising billions or even trillions of data points.
- The training data is largely unlabeled, unlike in traditional ML, where data is typically structured, labeled, and often numerical or categorical.

The term *foundation* reflects the model's general-purpose nature, as it is not trained for a specific task. Instead, it serves as a base model that can be adapted to various specialized applications by incorporating additional data and fine-tuning. This customization allows for application-specific performance.

Large language models (LLMs) are a type of FM. LLMs are trained on huge amounts of text. These models are often generative, meaning they are capable of producing sequences of text. The transformer architecture is the most commonly used ML model architecture for building LLMs. OpenAI's GPT-3 and GPT-4 models[5] are the most well-known examples of LLMs, but open-source alternatives are also available, including Mistral[6] and Llama.[7] To find a wide variety of open-source LLMs and other pretrained AI models, you can visit the model hub Hugging Face.[8]

FMs are customized or complemented through, for example, fine-tuning, prompting, retrieval-augmented generation (RAG), and "guardrails" that may preprocess input to and postprocess output from the FM and other components. A guardrail serves as a safeguard to ensure the safe and responsible use of AI technologies and prevent some attacks. It may include strategies, mechanisms, and policies designed to prevent misuse, protect user privacy, and promote transparency and fairness. RAG is also a popular method of complementing FMs, whereby specific data that is related to a request to an AI is retrieved and used to augment the input to the model. The RAG data is often specific and private for a given context (e.g., internal knowledge of a particular organization).

1.5.3 Model Development Life Cycle

Once the model is developed, the next step is **model build**—that is, building an executable artifact that includes the model or access to it. If the model is included, this step involves transforming the model into a deployable format that can be executed within the system. The model build stage ensures that the AI model is ready for integration.

After the model is built, it needs to be thoroughly tested to assess its accuracy and identify any potential risks or biases. The **model test** stage involves evaluating the model's performance against predefined metrics and criteria. It is important to ensure that the model operates reliably and produces accurate results.

---

5. https://openai.com/

6. https://mistral.ai/

7. https://llama.com/

8. https://huggingface.co/

Once the model has been tested and approved, it can be released for integration into the system. The **model release** stage involves finalizing the model for deployment and approving it for integration with other components of the system.

1.5.4 Resource Allocation for AI Parts

As mentioned earlier in the discussion of resources, the allocation for the AI portion of the system depends on the AI techniques used. Resource requirements for these different techniques depend on the technique chosen.

- **ML:** The training phase of an ML model is performed on either a local resource or a cloud resource—or in special cases, via edge/on-device learning. The resulting executable can be allocated to an edge resource, if small enough. Otherwise, it is allocated to either local or remote resources.
- **FM:** An FM is typically hosted on cloud resources. Access can be through API calls or service message calls. Some FMs are trained for specific domains. If small enough, they can run on edge devices directly, such as phones or smart speakers, for real-time applications. Techniques for compressing or distilling FMs and reducing their resource requirements are a matter of ongoing research and will evolve over time. We discuss these techniques in Chapter 4.

1.6 Dealing with Uncertainty

Non-symbolic AI models are inherently probabilistic. In other words, there is some probability (hopefully small) that the model output is incorrect. Whether this is acceptable depends on your organization's risk tolerance in terms of the system you are building. This consideration leads to an increasing emphasis on AI risk assessment, responsible AI, AI safety, standard conformance, and regulatory compliance.

Because of the inherent uncertainty in AI systems, your organization should conduct a risk assessment focused on the use of the system being constructed. This risk assessment should look at human values and AI safety, among other factors.

1.6.1 Human Values

Some AI systems are unpredictable and autonomous. Their unpredictable behavior necessitates that risk assessments must evaluate the impact, in

terms of both consequence and likelihood, if these systems violate ethical behavior.

In Chapters 7–11, we will focus on key quality attributes, such as reliability, security, privacy, and fairness. In any system, these qualities are balanced against one another. Design decisions for a system must reflect system priorities and tradeoffs should be explicitly considered.

1.6.2 Safety

Safety is an evolving concern in AI systems. Initially centered on physical and psychological safety to individuals, AI safety now encompasses concerns about AI systems with dangerous capabilities, such as chemical, biological, radiological, and nuclear (CBRN) concerns; misuse potential including cybersecurity, broader societal harms via misinformation/disinformation at scale, AI controllability, and even existential threats for humanity.

This change reflects that the meaning of safety is now more about the severity of AI system risks rather than the types of risks. That is why we have not included a chapter on AI safety, as we believe the architectural and DevOps approaches to evaluating and achieving quality attributes are the ultimate way to mitigate AI system risks and reduce them to acceptable levels, given the risk appetite for an organization or wider society.

1.7 Summary

This book discusses AI engineering—the application of software engineering to AI systems. AI systems are machine-based systems that use AI in one or more of their components, but always include other, non-AI components. Given that the overall quality of a system depends on all of its parts and their interplay, it is important to practice AI engineering well, with the aim of creating high-quality AI systems and operating them effectively.

The quality of an AI system depends on three factors: the life-cycle processes used, the software architecture, and the quality of the AI model. The processes used help the designer detect errors—both logical and in the model output—early in the development process.

The software architecture contributes to the achievement of one set of quality attributes requirements; it does this through the use of architectural tactics. The AI model used also contributes to the achievement of quality attribute requirements; it does so through the choice of model type, training data, and training algorithms. Data preparation is important in ensuring

model quality. Data cleaning and feature engineering are two aspects of data preparation.

Three types of AI models are symbolic models, narrow ML models, and foundation models. FMs and narrow ML models are specialized types of ML models. The probabilistic nature of AI models means an organization should perform a risk assessment to determine the effects of incorrect outputs of the AI models and AI systems.

1.8 Discussion Questions

1. A definition should be inclusive (including what you wish to include) and exclusive (excluding what you do not wish to include). Critique the definitions of software architecture, DevOps, AI systems, and AI engineering using this criterion.
2. What does your cloud provider charge for different types of resources and services? How do you think that affects the design of the system?
3. Ask your favorite LLM to describe the differences between it and its competitors. What do you think of the responses?

1.9 For Further Reading

You can read more about software architecture in *Software Architecture in Practice* by Len Bass, Paul Clements, and Rick Kazman [Bass 22]. The definition of software architecture we use can be found on page 2.

You can read more about DevOps in *DevOps: An Architect's Perspective* by Len Bass, Ingo Weber, and Liming Zhu [Bass 15]. The definition of DevOps that we use can be found on page 4.

The book *Deployment and Operations for Software Engineers* by Len Bass and John Klein [Bass 19] goes into detail about the life cycle.

A general introduction to AI concepts can be found in *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by Aurélien Géron [Géron 22].

You can read more about responsible AI in *Responsible AI: Best Practices for Creating Trustworthy AI Systems* by Qinghua Lu, Liming Zhu, Jon Whittle, and Xiwei Xu [Lu 23A].

This page intentionally left blank

Index

A
Abdelwahab, Hammam, 213
A/B testing, 130–131
abuse/misuse attacks, 179
acceptance testing, 172
access control
  credentials and, 37, 38
  to features, 103
  model registry for, 107
  ZT principles for, 180–181
accuracy
  challenges with FM, 93
  for classification models, 164
  and data division, 104
  data drift vs., 137–138
  evaluation for, 109–111
  generalization vs., 99
  improving, 166–173
  metrics for, 164–166
  in performance, 155
  prompt engineering for, 79
  as quality attribute, 5
  RAG for, 80
  reliable, 144
  testing for, 169
  via voting-based design, 153
activation functions, 68
adapter modules, 81
adaptive learning, 78
Advanced Robotics for Manufacturing (ARM) Hub, 235–254
adversarial testing/training, 147, 170–171, 182
aggregated data values, 102
AI Fairness 360, 198
AI model
  accuracy testing for, 110
  AI/non-AI co-design, 119
  in AI system life cycle, 6–10, 97–114
  attacks on, 185–186
  building of, 18
  chaos engineering test of, 139
  cost comparisons, 86–89
  as data-dependent, 10–11, 13–14, 99
  deployment of, 19, 28
  development of, 7
  engineering features for, 259–261
  hosting costs, 89
  model choice, 49–52, 145, 160
  monitoring performance of, 10
  proxy models, 153
  registry for, 107, 109
  scaling of, 27, 28
  size of, 158
  in system quality, 4–6, 13–19
  testing of/by, 18–19, 111–112
  third-party/vendor-supplied, 97, 182
  training of. *see* training of models
  two-part basis for, 14–15
  types of, 13, 14–18, 47–48
  web integration for, 33
AIOps, 271, 274
AI systems
  accuracy in, 164
  AI model in, 13–19
  analysis of, 139–140
  architecture of, 10–13
  attacks on, 178–179, 186
  characteristics of, 4–6
  defined, 2
  distinct responsibilities in, 148
  engineering of, 1–2, 3–4, 44
  fairness in, 193, 199–200
  FhGenie example of, 84–86
  future of, 269–280
  lifecycle processes, 6–10, 117
  in MLOps, 274
  observability of, 204–207
  performance efficiency in, 155–158
  privacy in, 192–193
  quality in, 1–2, 5, 12
  reliable, 152
  safety of, 20
  security for, 178–179, 182
  third-party support for, 237
  vs. traditional software, 269
  uncertainty in, 19–21
  using FMs, 86–91
  See also case studies; lifecycle processes
AIware, 276–278, 279
alert generation, 136, 151, 162
algorithms
  algorithmic solutions, 14
  classification algorithms, 53, 54–55
  clustering, 61
  for drift detection, 137
  regression algorithms, 57–60
  training algorithms, 40
  for vector spaces, 72
Alteryx, 101
Amazon, 31
Amazon Redshift, 100
Amazon Sagemaker, 103, 107
amortization costs, 89
analyzing system
  data-based analysis, 139–140
  via databases, 136
  for failures, 149
  for important features, 148
  for non-AI portion, 41
  in system life cycle, 10, 139–140
anomaly detection, 54, 61, 186
Apache Spark, 101
ARM (Advanced Robotics for Manufacturing) Hub, 235–254
asset price prediction, 58, 59
ATG (autonomous test generation), 112
attention distillation, 81
attention mechanism, 73
audio models, FM, 51
audio processing, 63

auditability, 204
authentication, 176, 180
authorization, 176, 180, 181
autoencoders, 74
automation in DevOps, 35–36, 255
AutoML, 274
autonomous test generation (ATG), 112
autonomy and safety, 272–273
autoscaling, 26–27, 28, 163
availability, 5
Avižienis, Algirdas, 144, 177
AWS ecosystem, 101, 103, 241
Azure, 84–85, 107, 236, 239
Azure Data Factory, 245

B

base FM models, 51, 71
batch size, hyperparameter, 67
benchmarking, 110, 171
bias
  acceptance testing for, 172
  and AI system fairness, 193
  avoiding, 197
  detecting, 126
  drift, 200
  and regulation of MLOps, 273
  in training data, 66, 99, 104, 147, 167
binary classification, 52
biological classification, 61
BLOOM, 65
BloombergGPT, 64, 72
blue/green deployment, 129
Boehm, Barry, 117
Box, George, 47
Brandeis, Louis D., 201
bug fixes, 36, 127, 131–132
build stage
  AI model portion, 8, 18, 108–109
  build servers, 122
  non-AI portion, 41
  observability in, 209–210
  security improvements in, 183–184
  in system life cycle, 8, 122–123
business organizations
  breaking of SLA by, 162
  fairness policies for, 198
  maturity of, 91–93, 239
  privacy policies of, 194
  private chatbots for, 84
  SMEs, 235, 236, 237
  training of base FM by, 75
business process management, 64

C

caching, 120, 159
canary testing, 130–131, 184
capability evaluation, 110, 111
case studies
  ARM Hub, 235–254
  automated tendering problem, 214–217
  CI/CD workflow, 249
  for customer churn prediction, 255–267
  data ingestion in, 240, 246
  development in, 217
  Fraunhofer IAIS, 213–234
  LLM-as-a-service, 237, 238
  MLOps in, 218, 220, 244–250
  ML setup, 220–221
  operations in, 217
  PoC (proof of concept) phase, 222–223
  of RAG-based chatbot, 238–244, 249, 251
  stakeholder involvement, 221–222
category encoding, 102
chain of thought, 79
change-proof features, 102–103
chaos engineering, 138–139, 151
chatbots, 63, 238–244, 249–250
ChatGPT, 63, 91, 92
chi-squared test, 137
CIA (confidentiality, integrity, availability) triad, 176–178
CI/CD pipeline, 141, 183, 184, 217, 219, 225, 248, 250, 253, 270
circuit breakers, 148
classification models, 17, 52–57, 164, 169, 175
cleaning data, 14, 91–92, 100–101
client-server architecture, 29
cloud, the
  DevOps in, 42
  distributed computing in, 23
  hosting costs, 89
  as operation/update pipeline, 12
  resources in, 12
  scalability and, 26–27
  training platforms on, 107
clustering data, 102
clustering models, 17, 60–63
code
  AI-written, 276–278
  for model training, 106
  scripts as, 37, 38
collocating microservices, 120
communication
  via client-server architecture, 29
  distributed computing messaging, 25–26
  efficient, 158
  in microservice architecture, 31, 32, 120
  REST interface for, 33–34
  in service-oriented architecture, 30
compatibility testing, 124
compliance testing, 124
concept drift, 140
concurrent testing, 113
confabulation/hallucination, 93
confidentiality, integrity, availability (CIA) triad, 176–178
containers, 24–28, 119, 128, 158, 217, 219
content generation, 80
contextual adjustment, 78
contingency plans, 151
continuous deployment pipeline (CDP), 128
continuous integration (CI), 122–123, 124, 133, 183
continuous vs. defined releases, 126–128
control charts, 137
convolutional neural networks (CNNs), 74
correctness evaluation, 110
credentials, access, 37, 38
credit scoring, 54, 55, 60
CRUD (Create, Read, Update, Delete), 108
culture, DevOps, 35
curriculum learning, 147

customer churn prediction, 255–267
customer segmentation, 54, 55, 62
customizable guardrails, 82

D
data
  and accuracy, 146–147, 166–168, 258
  and AI model quality, 4, 13–14
  in AIware, 277
  analysis, 139–140
  architecture and, 10–11
  attacks on, 184–185
  augmenting, 146
  chaos engineering with, 139
  cleaning of, 14, 91–92, 100–101
  common quality problems, 14
  drift, 125, 137–138, 145, 151, 200, 264
  enterprise warehouses of, 258–259
  feature engineering and, 102–103
  gathering, 99–100
  improvements to, 126
  ingestion of, 99, 236, 240, 246
  lake/warehouse/lakehouse, 100
  management, 98, 99–101
  and model reliability, 145
  monitoring for, 151
  observability and preparation of, 208
  processing, 101
  relationships in, 98, 102
  security considerations, 182
  security improvements via, 182
  storage of, 100
  tracing of structured, 205–206
  for training. *see* training dataset
  versioning and lineage, 100
databases
  input via, 13
  for monitoring/observation, 136
  in service-oriented architecture, 30
  and test repeatability, 112, 125
  Vector Search of, 242
  viewing model as, 108
Databrick, 100, 241, 248
data-centric architecture, 275
Data Version Control (DVC), 38, 100
DBSCAN (density-based spatial clustering of applications with noise), 62
decision tree regression, 59
decision trees, 55, 145, 160, 185
deep feature synthesis, 102
deep learning, 17
defense in depth, 175–176
defined vs. continuous releases, 126–128
degradation, graceful, 132, 153
density-based spatial clustering of applications with noise (DBSCAN), 62
dependability, 177
deployment
  of AI models, 28, 114
  automated quality gates for, 36
  via client-server architecture, 29
  in the cloud, 23–24
  of containers, 28
  of customer churn prediction model, 262–263
  defined vs. continuous releases, 126–128
  by Fraunhofer, 228–229
  in life cycle process, 6
  matching model to resources in, 134–135
  of microservices model, 32, 119
  model choice and, 160
  for non-AI portion, 41
  offline, 135
  patterns of, 128–132
  resources for, 12
  security testing during, 182–183
  as services, 108
  via SOA system, 30
  in system life cycle, 9, 125–135
  and version skew, 132–134
Derakhshan, Roozbeh, 235
design
  AI-driven, 276
  co-design and development, 118–119
  for modifiability, 120–121
  monitoring mechanisms in, 10
  quality requirements for, 6, 20
  questionable output in, 150
  system architecture, 10–13
  in system life cycle, 7, 118–121
  voting-based design, 153
detecting faults, 149–152
deterministic systems, 165
development
  co-design and, 118–119
  cost of FM, 87–88
  data management in, 98, 99–101
  distributed computing and, 28
  dividing the data, 103–105
  feature engineering in, 101–103
  four steps of, 98
  of individual components, 6
  of microservices model, 119
  model generation, 105–106
  for non-AI portion, 41, 121–122
  and operations, 35
  resource reduction during, 159–160
  security during, 183–184
  support for model training, 106–107
  in system life cycle, 6, 7, 97–107, 118, 121–122
DevOps
  AI for, 271
  for AI systems, 1, 2
  defined, 3, 35
  DevOps 2.0, 270–271
  growth of field, 280
  Infrastructure as Code (IaC), 36–41
  means utilized for, 35–36
  MLOps included in, 42, 43
  processes, 41–42, 126
  for security, 177
DevSecOps, 177
discovery mechanisms, 27, 28
distance metrics, 61
distilling, 75, 81
distributed computing
  in the cloud, 23–24
  DevOps processes, 41–42
  distributed architectures, 28–32
  Infrastructure as Code (IaC), 36–41
  interface styles, 32–36

distributed computing (*continued*)
MLOps, 42–44
VMs and containers, 24–28
Docker, 217, 219, 228, 229
document clustering, 61
document retrieval, 80
Domain Name System (DNS), 27
domain-specific FM models, 52, 75, 80
DoS attacks, 186
Driftctl, 40
drift detection algorithms, 137
driving software, 278
Drucker, Peter, 155
drug classifications, 54
drug efficacy analysis, 58
DVC (Data Version Control), 38, 100
dynamic prompt engineering, 77, 78
dynamic updates, 43

E

early stopping, 160
economic trend analysis, 57, 58
edge case representation, 103
edge devices, 12, 32, 135, 158, 159
efficiency
aspects of software, 155–158
enhancing human, 270
for FMs, 163–164
improving, 158–163
MLOps for, 245
Einstein, Albert, 269
embeddings, vector, 79
encapsulation, 25, 108
encryption, 177, 181
energy consumption, 155, 158, 164
energy use prediction, 58, 59
enrollment forecasting, 58, 60
ensemble methods, 146, 185
enterprise systems, 30
epoch number, hyperparameters and, 68
errors. *See* faults
ethical AI, 191
ethical behavior, 20
ETL (extract, transform, load) process, 240, 245
evaluation of AI model, 10, 109–111, 250, 261–262
evasion attacks, 178
exception handling, 153
executable version
efficiency concerns and, 160, 161
intermediate guardrails for, 82
resources for, 105
testing of, 6, 121–122, 123–125
execution guardrails, 82
explainability, 204, 209, 211
exploratory developmental phase, 105
external FM use, 89–90

F

facial recognition, 54
failure mode and effects analysis (FMEA), 149
failures
failures, faults, and errors, 144–145
questionable output, 150
testing for, 138–139
total failure, 149–150
See also faults
Fairlearn by Microsoft, 198
fairness
in AI systems, 193
means for achieving, 197–201
as quality attribute, 3, 5
and regulation of MLOps, 273
FastText, 73
faults
circuit breakers for, 148
detecting, 149–152, 153, 184
failures, faults, and errors, 144–145, 149
IaC to reduce, 36, 38–39
managing, 153–154
monitoring to record/catch, 136–137
potential feature engineering, 102–103
prevention of, 145–149
recovery from, 152–154
rolling upgrades and catching, 130
updates prompted by, 125–126
fault tree analysis (FTA), 149
Feast, 103
features
construction of, 102
efficiency concerns and, 161
engineering of, 101–103, 148, 160, 259–261
example feature table, 257
feature complexity tradeoff, 103
feature normalization, 145
feature toggles, 134
and improved accuracy, 168
few-shot learning, 78
FhGenie, 83–86
financial considerations
AI model cost comparisons, 86–89
chatbot cost transparency, 249
cost of FM training, 64, 72, 86–87, 163
cost of IaC scripts, 39
of efficiency, 157, 163
hosting price, 89, 90
update method cost comparison, 129–130
first responders, 136, 137, 163
fixed-rule-based systems, 132
flakey tests, 113
FMEA (failure mode and effects analysis), 149
F1 scores, 49, 16
foundation models (FMs)
accuracy for, 165, 169–170
adaptation of existing, 51–52, 65
adversarial training, 170
alternate FM architectures, 74
as application independent, 43–44
attacks on, 186–187
bespoke, 64–65, 89
challenges of, 93
choosing between types of, 89–91
cost of, 64, 86–89
customizing, 75–86, 93, 106, 244
data cleaning by, 91–92
designing a system using, 86–91
distilling, 81
efficiency of, 159, 163–164
engineering of, 1
fairness considerations for, 200–201

FhGenie example of, 84–86
fine-tuning of, 17, 43, 51, 65, 71, 75, 80–81, 247
guardrails for, 18, 82–83
as heavyweight model, 134
learning outside the model, 280
learning process of, 47–48
LLMs as FMs, 63
maturity of, 91–92
in MLOps, 274
model overview, 17–18
organizational maturity and, 91–93
pretraining of, 75, 256
privacy protection and, 196–197
probabilistic nature of, 113
prompt engineering for, 18, 76–79
RAG customizations for, 79–80
resource requirements for, 19
security for, 180, 182, 186–188
size of, 72
specialization of, 280
student/teacher, 81
testing of/by, 112
training of, 43–44
transformer architecture for, 72–74
updating, 83
when to use, 51–52
Fowler, Martin, 1
framed autonomy, 272
Fraunhofer, 84
Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)
automated tendering problem, 214–217
implementation phase, 223–230
ML architecture for, 218–221, 230
MLOps cycle in, 218, 220, 229, 231
PoC (proof of concept) phase, 222–223
production for, 230–232
stakeholder involvement, 221–222
training dataset in, 216
FTA (fault tree analysis), 149
future of AI, 269–280

G

gated recurrent unit (GRU), 74
GDPR (General Data Protection Regulation), 124–125, 193
Gemini, 63, 65, 78, 84, 278
Gemini-Reasoning, 278
generalization
and accuracy, 89, 99, 169–170
dividing the data for, 104
for guardrails, 82
and privacy with FM models, 196–197
reliable, 144, 145, 146, 147
training for, 105, 143
generation, model, 105–106
generative AI, 3, 80, 239
Giesselbach, Sven, 213
GitHub, 249
GitLab, 228
global explanation techniques, 209
GloVe, 73
GNNs (graph neural networks), 64, 74
Goldberg, Natalie, 97
Google, 276
Google BigQuery, 100
Google Cloud Dataflow, 101
Google Cloud Vertex AI, 107
Google Gemini. *See* Gemini
Google PAIR tools, 198
governance of MLOps, 273
GPT-4, 18, 72, 91
GPT-3, 18
graceful degradation, 132, 153
gradient boosting regression, 59–60
graph neural networks (GNNs), 64, 74
GraphQL, 32, 34–35, 109, 158
ground truth, proxy measures for delayed, 151
growth rate modeling, 58
GRU (gated recurrent unit), 74
guardrails
fairness detection and, 200
FM, 18, 82–83
as framed autonomy, 272
for questionable outputs, 150
system-level, 273, 279
Gupta, Vishwani, 213

H

hallucination/confabulation, 93
hard clustering, 61
hardware development, 118–119
Hawking, Stephen, 191
health checks, 149–150
Health Insurance Portability and Accountability Act (HIPAA), 124
heartbeat health checks, 150
hierarchical clustering, 63
holdout validation, 147
holistic evaluation of language models (HELM), 171
hosting costs, 89, 90
HTTP/1.1 protocol, 26, 32, 33
H2O.ai, 102
Hugging Face model hub, 230, 248
human input
to achieve fairness, 200
and DevOps 2.0, 270
first responders, 136, 137, 163
on FM accuracy, 171
future of, 276–277, 279–280
human accountability, 114
human values, 19–20, 191
pre-deployment sign off, 114, 126
and trust in AI, 279–280
hybrid FM models, 74
hyperparameters
efficiency concerns and, 160–161
experimentation with, 105
and improved accuracy, 166
for MLs, 66–69
tuning, 146
hypervisors, 24, 25

I

IaC (Infrastructure as Code), 35–41, 105
idempotence, 39–40
Identity and Access Management (IAM), 40
if-then statements, 15
impact assessment, 110
incidents, monitoring problematic, 136–137
inference engines
ARM Hub system, 248

inference engines (*continued*)
customer churn prediction, 262–263
efficiency of, 157, 158, 163
knowledge base and, 14
local/offline, 135
service wrapper for, 108
in symbolic AI, 16
information retrieval, 79–80
Infrastructure as Code (IaC), 35–41, 105
infrastructure drift, 40–41, 138
infrastructure monitoring, 135
infrastructure tests, 139
input
and AI model quality, 13
attention mechanism for capturing, 73
via the inference engine, 108
inference engine to process, 14
input guardrails, 75, 82
for isolated unit tests, 121
for ML models, 16–17
prompt engineering via, 77
reliability amidst unexpected, 144
sanitation, 175
sources of, 13
tokenization of, 72
from users, 83
install/uninstall testing, 124
interface styles
changes to, 121
GraphQL, 34–35
interface mismatch, 132, 133
REST, 32–34, 109
in service-oriented architecture, 30
intermediate guardrails, 82
Internet, the, 26, 32–34
Internet Protocol (IP) address, 25, 27, 159
interpretable guardrails, 82
iterative refinement, 68, 78

J
Jenkins, 122, 183
Jupyter Notebooks, 105, 226

K
el Kaliouby, Rana, 191
Keras, 104
Kirsch, Birgit, 213
k-means clustering, 61–62
k-nearest neighbors (KNN), 54
KNIME, 101
knowledge base, 14
knowledge distillation, 81
Kolmogorov–Smirnov (KS) test, 137, 169

L
labeled data, 47, 52, 80
lake/warehouse/lakehouse, 100, 236, 252
LangChain, 242
language models, FM, 51
language processing, traditional, 73
large language models (LLMs). *See* LLMs (large language models)
latency, 85, 156, 158, 159
learning rate, hyperparameters and, 67
legal compliance, 114, 126, 192–193, 273
Levenshtein distance, 61
lifecycle processes
analysis in, 139–140
build stage, 8, 108–109, 122–123
complexity of, 117
of containers, 28
deployment in, 8–9, 114, 125–135
design phase, 7, 118–121
development in, 7–8, 97–107, 121–122
MLOps for AI, 42–44
model registry for, 107
for non-AI portion, 6–10, 41
observability of, 207
operation/monitoring in, 9–10, 135–140
stages of, 6–10
system analysis in, 10
as system quality, 4–6
testing in, 8, 109–113, 123–125
updates in, 9
version control in, 36–37
LightGBM, 257
limited grounding, 93
lineage
data, 100
features, 103
model lineage tools, 208
tracking tools, 148
linear regression, 58, 107
linting, 227
Linux, 24–25
live updates, 9
LLMs (large language models)
benchmark tests for accurate, 171–172
case study implementing, 236
chatbots and FM popularity, 63
cost of training, 88
data ingestion for, 236
evaluation/testing of, 111
fine-tuning of, 247
LLM-as-a-service, 237, 249, 251
in MLOps, 274
prompt engineering for, 76–79
RAG-based chatbot architecture, 239–244
security for, 187–188
for SME manufacturing, 238
as text-only model, 71
as type of FM, 18
load balancing, 26, 28, 121
local explanation techniques, 209
local resources, 12
LOCKED (limit, opt-out, correct, know, equal, delete) rights, 193, 194–196, 201
logistic regression, 54, 185
logs, observability via, 207
long short-term memory (LSTM), 74
loss function, 49, 81
low-rank adaptation (LoRA), 81

M
machine learning (ML) models. See ML (machine learning) models
market segmentation, 61, 62
Martin, Demetri, 143
Martins, Claudio, 213
meaning, context and, 73
medical diagnosis, 54, 55, 61, 63, 64, 153, 192
messaging, 25–26, 120

metadata management, 107, 123
microservice architecture
  benefits of, 119–120
  and continuous deployment, 128
  design of, 118
  DevOps and, 35, 36
  life cycle for, 117
  overview of, 30, 31–32
  REST interface and, 33
  services in, 121
Microsoft Azure, 84–85, 107, 236, 239
Milvus, 79
MIME data type, 33
MLFlow, 104, 227, 263, 264
ML (machine learning) models
  automation of, 255
  case studies using, 220–221, 224, 230
  classification models, 52–57
  containers to serve, 25
  cost of developing, 86–89
  FMs as type of, 71
  hyperparameters, 66–69
  learning terminology for, 48–49
  manipulation of, 5
  MLOps for, 14, 42–44, 244
  model overview, 16–17
  narrow, 86–89, 98, 134, 156, 159
  as non-symbolic AI, 15
  probabilistic nature of, 113, 125
  resource requirements for, 19
  training of, 5, 47–48, 50–51
  when to use, 50–51
MLOps
  case studies of, 218, 220, 237, 244–250
  defined, 14
  and dynamic updates, 43
  future of, 273–274, 275
  for LLM-as-a-service, 237
  in model development, 97–98
  practices distinct to, 42
  role of AI in, 274
modifiability, 5
modules, 121
monitoring
  for AI systems, 1, 135–140
  via the cloud, 12
  of customer churn prediction model, 263–264
  in DevOps 2.0, 270
  for error detection, 150–152
  via FM guardrails, 82
  methods for, 9–10
  MLOps to manage, 42
  observability and, 203, 204–207
  of problem incidents, 136–137
  in system life cycle, 9–10
  See also operation/monitoring
Mosaic ML, 65
multi-agent systems, 279
multi-class classification, 52
multimodal guardrails, 83
multimodal models, FM, 52, 63, 83
multiple instances, redundancy via, 152

N

naïve Bayes algorithm, 55–56, 160
narrow ML models, 17, 86–89, 98, 134, 156
Netflix, 31
networks, 25, 56–57, 120, 159
neural networks
  adapter modules, 81
  classification models, 56–57
  deep, 106, 145, 160, 185
  hyperparameters for, 67–69
  regression models, 60
  regularization techniques for, 147
  resources for executing, 106
  transformer architecture as, 73–74
noise, introducing, 80
non-AI portions
  AI/non-AI co-design, 119
  in AI system, 1–2, 6–10, 44
  analysis of, 140
  development of, 7–8, 121–122
  DevOps for, 41, 270
  of narrow ML models, 86
  quality requirements for, 5
  responsibilities of, 148
nondeterministic systems, 165
non-symbolic AI, 15, 47
numerical transformations, 102, 168
NVIDIA Nemotron, 65

O

observability
  of AI decisions, 265
  defined, 203
  in DevOps 2.0, 270
  evolution from monitorability to, 204–207
  in future AI projects, 276
  improving, 207–211
  qualities related to, 203
  as quality attribute, 3
ontology reasoning, in symbolic AI, 16
OOD (out-of-distribution) data, 5, 143, 167
OpenAI, 18, 84–85
open-weight FMs, 89–91
operating systems (OSes), 24, 25, 36–37
operational objectives, 4
operation/monitoring
  in AI system life cycle, 9–10, 135–140
  in ARM Hub system, 247–248
  cost of FM, 89
  error detection in, 150–152
  fairness concerns in, 200
  and improved accuracy, 169
  and improved efficiency, 162
  for non-AI portion, 41
  observability and, 211
  privacy protection and, 197
  reliable, 143–145
  security improvements in, 184–186
  See also monitoring
optimization algorithms, 68
Optuna, 263
orchestration systems, 27–28
organizational means, for DevOps, 35
OS (operating systems), 24, 25, 36–37
out-of-distribution (OOD) data, 5, 143, 167
output
  accurate, 168

output (*continued*)
caching, 159
future of AI, 272
guardrails, 75, 82
inconsistent, 170
inferred, 2–3, 13, 14, 98, 108
for isolated unit tests, 121
preparation for reliable, 145–146
probabilistic test, 113
via prompt engineering, 77
questionable output, 150
of "student" FMs, 81
uncertainty in, 19
overfitting, 48, 144

P
parameter-efficient fine-tuning (PEFT), 81
parameters
efficiency concerns and, 160–161
for FMs, 90
for MLs, 66, 80
for model generation, 105
sharing, 81
performance
accuracy in, 155, 164–173
adaptive, 146
aspects of software, 156
efficiency in, 155–164
evaluation of, 10, 109–111, 261–262
of FM models, 17
in-training monitoring of, 168–169
metrics for monitoring, 137
MLOps to manage, 42
post-deployment monitoring of, 263–264
as quality attribute, 3, 5
reliable, 143
testing, 124
updates to improve, 125–126
personal information, 192–193, 194
personalized recommendations, 80
Petroski, Henry, 1
Pichai, Sundar, 276
Pinecone, 79
ping/echo health check, 150
planning, in symbolic AI, 16
poisoning attacks, 178
polynomial regression, 58
portability, 39
prediction
vs. classification, 53
customer churn prediction, 255–267
data drift and incorrect, 137
principal component analysis (PCA), 102
privacy
for AI systems, 192–193
in banking sector, 265
means for achieving, 194–197
MLOps for, 246
of proprietary information, 83–86
as quality attribute, 3
RAG and, 80
and security attacks, 178
problems. *See* faults
progressive prompting, 78
prompt engineering, 18, 75, 93, 246
proprietary information, privacy of, 83–86
proxy models, 153
Pylint library, 227
PyTorch Dataloader, 104, 107

Q
quality attributes, 3, 109–111, 146
quality control
for AI systems, 1, 279
and dynamic updates, 43, 138
gates for, 36, 41
for RAG, 80
quality requirements
as architectural input, 11
changes to, 126
defined, 5
designer awareness of, 6
fulfilling, 12
and human values, 20
human- vs. AI-driven, 279
and system quality, 5
quantum computing, 177
queries, GraphQL, 34, 35

R
RAG (retrieval-augmented generation)
customizing FMs via, 18, 75, 79–80
and FM attacks, 187
future of, 275
improving accuracy via, 170
RAG-based chatbot, 238–244, 249, 251
RAG-chain retrieval, 242–243
RAG guardrails, 82
RAI (responsible AI) requirements, 191
random forest algorithms, 55
random forest regression, 59
real estate valuation, 59, 144
recall-oriented understudy for gisting evaluation (ROUGE), 171
recommendation systems, 61
reconfiguration, 153
recovery from faults, 152–154
recurrent neural networks (RNNs), 67, 74
red/black deployment, 129
red teaming, 111
redundancy, 150, 152–153
registry, model, 107, 109
regression models, 17, 57–60, 165, 169
regression testing, 124
regularization parameters, 68, 145, 147, 160
regulation of MLOps, 273
reinforcement learning from human feedback (RLHF), 111, 170, 188
release. *See* deployment
relevance, 80, 241
reliability
architectural placement of, 148
challenges with FM, 93
conditions for defining, 143–145
enhancing, 5
and fault detection, 149–152
and fault prevention, 145–149
and fault recovery, 152–154
via IaC, 39
methods for increasing, 151
model choice for, 145
in performance, 143
as quality attribute, 3
security and, 177

Remote Procedure Call (RPC), 109
repeatability, 112–114, 125, 165
Representational State Transfer (REST), 32–34, 108
resilience, 144, 146, 148
resources
  AI-portion allocation of, 19
  as architectural input, 11
  for dynamic prompt engineering, 78
  efficient use of, 157
  for execution vs. training, 105
  matching models to, 134–135
  reducing requirements for, 159–160
  in REST interface, 33
  scaling of, 163
  three categories of, 12
responsible AI (RAI) requirements, 191
retrieval-augmented generation (RAG). *See* RAG (retrieval-augmented generation)
risk assessments
  evaluation for, 109–111
  regression models for, 58, 60
  uncertainty and, 19
risk management, 2, 114, 245
RLHF (reinforcement learning from human feedback), 111, 170, 188
RNNs (recurrent neural networks), 67, 74
robotics, 64
robustness
  architectural placement of, 148
  enhancing, 5
  of ETL process, 240
  reliability and, 144
  via voting-based design, 153
Rockefeller, Nelson, 23
rollback of products, 131–132
roll-forward of products, 131–132
rolling upgrades, 129
ROUGE (recall-oriented understudy for gisting evaluation), 171
RPC (Remote Procedure Call), 109
rules, in symbolic AI, 15–16, 49, 66
runtime engines, 25, 28, 34, 158, 225

S

safety, 20, 114, 272–273
SBOM (software bill of materials), 123, 196, 208, 209
scalability
  of AI systems, 28, 236
  and client-server architecture, 29
  in the cloud, 26–27
  for containers, 28
  efficient, 156–157
  of FM models, 72
  MLOps for, 245
  of symbolic AI, 49
schedule pressure/slippage, 127
Schneier, Bruce, 175
scikit-learn, 102, 104, 107
scripts in IaC, 36, 37, 38, 39, 40
security
  attacks, 177, 178–179
  autonomy and safety, 272–273
  in banking sector, 265
  improving, 180–188
  and infrastructure drift, 40, 138
  and manipulated ML, 5
  model registry for, 107
  new/existing concerns, 175–176
  as quality attribute, 3, 5
  RAG and, 80
  testing, 111, 124, 226, 228
  traditional definition of, 176–178
self-consistency, 79
self-supervised learning, 47, 48, 106
semantic attacks, 175
semantic relationship detection, 73
sentiment analysis, 55, 56, 71
sequencing tests, 113
service level agreements (SLAs), 162, 236
service level objectives (SLOs), 131, 162
service-oriented architecture (SOA), 29–30, 33
services
  the cloud and scaling of, 26–27
  discovery by, 27
  in distributed computing, 24
  in GraphQL interface, 34
  LLM-as-a-service, 237
  messaging for, 25–26
  in microservice architecture, 31, 121
  monitoring logs from, 135
  packaged as containers, 109, 119, 128, 134
  reallocation of, 163
  in REST interface, 33
  service wrappers, 108–109
  updates and uninterrupted, 128–129
  version control for, 134
  vertical cutting of, 128
SLAs (service level agreements), 162, 236
SLOs (service level objectives), 131, 162
smartphones, 158
Snowflake, 100
Snyk, 40
SOA (service-oriented architecture), 29–30, 33
social network analysis, 61, 63
soft clustering, 61
software
  AI vs. traditional, 269
  AIware/AI-as-software, 276–278
  development, 118–119, 276–278
  efficiency of, 155–158
  engineering for AI, 1
software architecture
  accuracy improved by, 166–173
  for AI systems, 1, 2, 7
  cost considerations, 86–89
  data lakehouse, 100
  defined, 3, 10
  and defined releases, 127
  distinct responsibilities in, 148
  distributed architectures, 28–32
  error recognition in, 148, 149, 153
  experimentation with, 105
  fairness via, 198–199
  fault recovery in, 152
  for FMs, 72–74, 197
  of Fraunhofer, 218–221, 224–225
  future of, 274–276
  improved efficiency via, 158–159
  improved security via, 181–182

software architecture (*continued*)
influences on, 11
lakehouse, 100, 236, 252
for live updates, 9
monitoring mechanisms in, 10
observability enhancements via, 207–208
overview of, 10–13
and privacy protection, 194
of RAG-based chatbot, 239–244, 249
role of, 11
as system quality, 4–6
transformer architecture, 72–74
of virtual machines, 24
software bill of materials (SBOM), 123, 196, 208, 209
spam detection, 53, 56
speech/image recognition, 53, 54, 56, 175
stakeholder involvement, 221–222
static prompt engineering, 77–78
Steward, Cori, 235
storage, 100, 103
Strategy design pattern, 121
stress testing, effective, 99, 147
structured data, 205–206
sub-symbolic AI, 15, 47
supervised learning, 47, 48, 106
SVM (support vector machine), 56
SVR (support vector regression), 59
symbolic AI
defined, 15–16, 47
engine configuration, 66
preparation for training, 65–66
rules for, 16, 52, 66
when to use, 49–50
system code monitoring, 135
system perspective, 1–2, 148–149, 270
system-wide testing, 123–125, 210

T

Talend, 101
Tang, MingJian, 255
TCP/IP (Terminal Control Protocol/Internet Protocol), 26
team development, 119, 120
technological means, in DevOps, 35–36
Tecton, 103
temporal inconsistencies, 132
TensorFlow, 104, 106
Terminal Control Protocol/Internet Protocol (TCP/IP), 26
Tesla, 278
test cases, 172–173
testing
for accuracy, 169, 170–173
AI-generated, 111–112
of AI models, 18, 109–113
of AI systems, 1, 123–125
with chaos engineering, 138–139
CI server for build, 123
for defined releases, 127
effective stress testing, 99, 147
of executable version, 6
in Fraunhofer case study, 226, 228
future of AI, 271
human oversight of, 114
of individual components, 6
for non-AI portion, 41
privacy protection and, 197
reliability and, 146–147
security, 182–183
in system life cycle, 8
system-wide, 123–125, 210
test repeatability, 112–114
unit testing in isolation, 109, 121
test sets, 48, 104, 147, 167, 169, 216
throughput, 155–156
time-series and forecasting, 64
time to market, 127
tokenization, 72
tools
to achieve fairness, 198
in AI model development, 97
connection between monitoring, 204–205
cost for FM tools, 87–88
to ensure data quality, 183
for feature engineering, 102
in IaC, 36, 37
for model training, 106–107
to track lineage, 148
traceability, 39, 123, 205–206, 265
training dataset
in ARM Hub system, 246
for classification models, 52
cost of preparing, 88
for customization, 97
data management, 98, 99–101
distribution in, 5, 137, 145, 146–147, 167
dividing the data for, 104
efficiency concerns and, 161
for FM models, 17, 71, 75
in Fraunhofer case study, 216
gathering data, 99–100
idempotence and, 40
and improved accuracy, 166–168
improved security via, 182
and learning/model type, 47–48
for ML models, 16–17
and model generation, 105
for neural networks, 57
quality of/via, 5, 12, 166–168
for RAG-based chatbot, 241
reliability and, 145, 146–147
sourcing/validation for, 258
testing, 170
three subsets of, 48, 104, 147, 167, 216
tracing lineage of, 208
training of models
case study of, 261–262
client-server architecture for, 29
cost of, 64, 72, 86–87
decentralized, 31
fairness concerns in, 199–200
FM customization, 75–86
in Fraunhofer case study, 226–227
via MLOps, 42, 43–44
and model choice, 47–52
and model generation, 105
performance monitoring during, 168–169
platforms to train MLs, 50–51
preparation for, 65–69
via prompt engineering, 76–77
reliability and, 146–147
resources for, 105, 157, 160, 161
security and, 187
via service-oriented architecture, 30
SOA for, 30

terminology for, 48–49
time required for, 157
See also specific model by name
transformation of data, 101, 168
transformer architecture, 72–74
transparency, 203, 265
tree of thought, 79
trust issues, 191–192, 276, 279–280
Tuckman, Bruce, 119

U
uncertainty, 19–20, 146, 276
underfitting, 48
unit testing in isolation, 109, 121, 226, 228
unlabeled data, 47, 75
unsupervised learning, 47, 48, 98, 106
updates
via the cloud, 12
cost comparisons, 129–130
co-versioning registry of, 210
defined vs. continuous, 127
dynamic, 43, 138
to FMs, 83
post-deployment, 9
reasons to perform, 125–126
tests, 113
and uninterrupted service, 128–129
user interface
AI-written, 278
multimodal guardrails for, 83
for RAG-based chatbot, 243
user prompt engineering, 77
users
individual access rights of, 80, 84–85
reports from, 123–124
security considerations with, 176–177
updates via input from, 83

V
validation set, 48, 104, 147, 160, 167, 169, 216
variational autoencoders, 74
vector databases, 79
vector spaces, 72–73, 98, 102, 242
version control
co-versioning registry, 210
for data, 100
DevOps and, 36
for features, 103
and IaC, 36–37
MLOps to manage, 42
model registry for, 107
server for, 122
service wrapper support for, 109
tools for, 106
version skew mitigation, 132–134
video models, FM, 52
virtual machines (VMs), 24–28, 158
vision models, FM, 51
voting-based design, 153

W
Wall Street Journal, 71
Weber, Ingo, 47
Wegener, Dennis, 213
Winograd Schema Challenge (WSC), 171
Word2Vec, 72, 73
words, tokenization of, 72–73

X
XGBoost, 107, 257
Xia, Boming, 97, 117

Y
Yuxiu, 255

Z
ZenML, 38
zero trust (ZT), 180, 181