Preface

When ChatGPT came out, like many of my colleagues, I was disoriented. What surprised me wasn't the model's size or capabilities. For over a decade, the AI community has known that scaling up a model improves it. In 2012, the AlexNet authors noted in their landmark paper that: "All of our experiments suggest that our results can be improved simply by waiting for faster GPUs and bigger datasets to become available."[1, 2]

What surprised me was the sheer number of applications this capability boost unlocked. I thought a small increase in model quality metrics might result in a modest increase in applications. Instead, it resulted in an explosion of new possibilities.

Not only have these new AI capabilities increased the demand for AI applications, but they have also lowered the entry barrier for developers. It's become so easy to get started with building AI applications. It's even possible to build an application without writing a single line of code. This shift has transformed AI from a specialized discipline into a powerful development tool everyone can use.

Even though AI adoption today seems new, it's built upon techniques that have been around for a while. Papers about language modeling came out as early as the 1950s. Retrieval-augmented generation (RAG) applications are built upon retrieval technology that has powered search and recommender systems since long before the term RAG was coined. The best practices for deploying traditional machine learning applications—systematic experimentation, rigorous evaluation, relentless optimization for faster and cheaper models—are still the best practices for working with foundation model-based applications.

[1] An author of the AlexNet paper, Ilya Sutskever, went on to cofound OpenAI, turning this lesson into reality with GPT models.

[2] Even my small project in 2017, which used a language model to evaluate translation quality, concluded that we needed "a better language model."

The familiarity and ease of use of many AI engineering techniques can mislead people into thinking there is nothing new to AI engineering. But while many principles for building AI applications remain the same, the scale and improved capabilities of AI models introduce opportunities and challenges that require new solutions.

This book covers the end-to-end process of adapting foundation models to solve real-world problems, encompassing tried-and-true techniques from other engineering fields and techniques emerging with foundation models.

I set out to write the book because I wanted to learn, and I did learn a lot. I learned from the projects I worked on, the papers I read, and the people I interviewed. During the process of writing this book, I used notes from over 100 conversations and interviews, including researchers from major AI labs (OpenAI, Google, Anthropic, ...), framework developers (NVIDIA, Meta, Hugging Face, Anyscale, LangChain, LlamaIndex, ...), executives and heads of AI/data at companies of different sizes, product managers, community researchers, and independent application developers (see "Acknowledgments" on page xx).

I especially learned from early readers who tested my assumptions, introduced me to different perspectives, and exposed me to new problems and approaches. Some sections of the book have also received thousands of comments from the community after being shared on my blog, many giving me new perspectives or confirming a hypothesis.

I hope that this learning process will continue for me now that the book is in your hands, as you have experiences and perspectives that are unique to you. Please feel free to share any feedback you might have for this book with me via X, LinkedIn, or email at *hi@huyenchip.com*.

What This Book Is About

This book provides a framework for adapting foundation models, which include both large language models (LLMs) and large multimodal models (LMMs), to specific applications.

There are many different ways to build an application. This book outlines various solutions and also raises questions you can ask to evaluate the best solution for your needs. Some of the many questions that this book can help you answer are:

- Should I build this AI application?
- How do I evaluate my application? Can I use AI to evaluate AI outputs?
- What causes hallucinations? How do I detect and mitigate hallucinations?
- What are the best practices for prompt engineering?
- Why does RAG work? What are the strategies for doing RAG?

- What's an agent? How do I build and evaluate an agent?
- When to finetune a model? When not to finetune a model?
- How much data do I need? How do I validate the quality of my data?
- How do I make my model faster, cheaper, and secure?
- How do I create a feedback loop to improve my application continually?

The book will also help you navigate the overwhelming AI landscape: types of models, evaluation benchmarks, and a seemingly infinite number of use cases and application patterns.

The content in this book is illustrated using case studies, many of which I worked on, backed by ample references and extensively reviewed by experts from a wide range of backgrounds. Although the book took two years to write, it draws from my experience working with language models and ML systems from the last decade.

Like my previous O'Reilly book, *Designing Machine Learning Systems* (DMLS), this book focuses on the fundamentals of AI engineering instead of any specific tool or API. Tools become outdated quickly, but fundamentals should last longer.[3]

Reading *AI Engineering* (AIE) with *Designing Machine Learning Systems* (DMLS)

AIE can be a companion to DMLS. DMLS focuses on building applications on top of traditional ML models, which involves more tabular data annotations, feature engineering, and model training. AIE focuses on building applications on top of foundation models, which involves more prompt engineering, context construction, and parameter-efficient finetuning. Both books are self-contained and modular, so you can read either book independently.

Since foundation models are ML models, some concepts are relevant to working with both. If a topic is relevant to AIE but has been discussed extensively in DMLS, it'll still be covered in this book, but to a lesser extent, with pointers to relevant resources.

Note that many topics are covered in DMLS but not in AIE, and vice versa. The first chapter of this book also covers the differences between traditional ML engineering and AI engineering. A real-world system often involves both traditional ML models and foundation models, so knowledge about working with both is often necessary.

[3] Teaching a course on how to use TensorFlow in 2017 taught me a painful lesson about how quickly tools and tutorials become outdated.

Determining whether something will last, however, is often challenging. I relied on three criteria. First, for a problem, I determined whether it results from the fundamental limitations of how AI works or if it'll go away with better models. If a problem is fundamental, I'll analyze its challenges and solutions to address each challenge. I'm a fan of the start-simple approach, so for many problems, I'll start from the simplest solution and then progress with more complex solutions to address rising challenges.

Second, I consulted an extensive network of researchers and engineers, who are smarter than I am, about what they think are the most important problems and solutions.

Occasionally, I also relied on Lindy's Law, which infers that the future life expectancy of a technology is proportional to its current age. So if something has been around for a while, I assume that it'll continue existing for a while longer.

In this book, however, I occasionally included a concept that I believe to be temporary because it's immediately useful for some application developers or because it illustrates an interesting problem-solving approach.

What This Book Is Not

This book isn't a tutorial. While it mentions specific tools and includes pseudocode snippets to illustrate certain concepts, it doesn't teach you how to use a tool. Instead, it offers a framework for selecting tools. It includes many discussions on the tradeoffs between different solutions and the questions you should ask when evaluating a solution. When you want to use a tool, it's usually easy to find tutorials for it online. AI chatbots are also pretty good at helping you get started with popular tools.

This book isn't an ML theory book. It doesn't explain what a neural network is or how to build and train a model from scratch. While it explains many theoretical concepts immediately relevant to the discussion, the book is a practical book that focuses on helping you build successful AI applications to solve real-world problems.

While it's possible to build foundation model-based applications without ML expertise, a basic understanding of ML and statistics can help you build better applications and save you from unnecessary suffering. You can read this book without any prior ML background. However, you will be more effective while building AI applications if you know the following concepts:

- Probabilistic concepts such as sampling, determinism, and distribution.
- ML concepts such as supervision, self-supervision, log-likelihood, gradient descent, backpropagation, loss function, and hyperparameter tuning.

- Various neural network architectures, including feedforward, recurrent, and transformer.
- Metrics such as accuracy, F1, precision, recall, cosine similarity, and cross entropy.

If you don't know them yet, don't worry—this book has either brief, high-level explanations or pointers to resources that can get you up to speed.

Who This Book Is For

This book is for anyone who wants to leverage foundation models to solve real-world problems. This is a technical book, so the language of this book is geared toward technical roles, including AI engineers, ML engineers, data scientists, engineering managers, and technical product managers. This book is for you if you can relate to one of the following scenarios:

- You're building or optimizing an AI application, whether you're starting from scratch or looking to move beyond the demo phase into a production-ready stage. You may also be facing issues like hallucinations, security, latency, or costs, and need targeted solutions.
- You want to streamline your team's AI development process, making it more systematic, faster, and reliable.
- You want to understand how your organization can leverage foundation models to improve the business's bottom line and how to build a team to do so.

You can also benefit from the book if you belong to one of the following groups:

- Tool developers who want to identify underserved areas in AI engineering to position your products in the ecosystem.
- Researchers who want to better understand AI use cases.
- Job candidates seeking clarity on the skills needed to pursue a career as an AI engineer.
- Anyone wanting to better understand AI's capabilities and limitations, and how it might affect different roles.

I love getting to the bottom of things, so some sections dive a bit deeper into the technical side. While many early readers like the detail, it might not be for everyone. I'll give you a heads-up before things get too technical. Feel free to skip ahead if it feels a little too in the weeds!

Navigating This Book

This book is structured to follow the typical process for developing an AI application. Here's what this typical process looks like and how each chapter fits into the process. Because this book is modular, you're welcome to skip any section that you're already familiar with or that is less relevant to you.

Before deciding to build an AI application, it's necessary to understand what this process involves and answer questions such as: Is this application necessary? Is AI needed? Do I have to build this application myself? The first chapter of the book helps you answer these questions. It also covers a range of successful use cases to give a sense of what foundation models can do.

While an ML background is not necessary to build AI applications, understanding how a foundation model works under the hood is useful to make the most out of it. Chapter 2 analyzes the making of a foundation model and the design decisions with significant impacts on downstream applications, including its training data recipe, model architectures and scales, and how the model is trained to align to human preference. It then discusses how a model generates a response, which helps explain the model's seemingly baffling behaviors, like inconsistency and hallucinations. Changing the generation setting of a model is also often a cheap and easy way to significantly boost the model's performance.

Once you've committed to building an application with foundation models, evaluation will be an integral part of every step along the way. Evaluation is one of the hardest, if not the hardest, challenges of AI engineering. This book dedicates two chapters, Chapters 3 and 4, to explore different evaluation methods and how to use them to create a reliable and systematic evaluation pipeline for your application.

Given a query, the quality of a model's response depends on the following aspects (outside of the model's generation setting):

- The instructions for how the model should behave
- The context the model can use to respond to the query
- The model itself

The next three chapters of the book focus on how to optimize each of these aspects to improve a model's performance for an application. Chapter 5 covers prompt engineering, starting with what a prompt is, why prompt engineering works, and prompt engineering best practices. It then discusses how bad actors can exploit your application with prompt attacks and how to defend your application against them.

Chapter 6 explores why context is important for a model to generate accurate responses. It zooms into two major application patterns for context construction: RAG and agentic. The RAG pattern is better understood and has proven to work well in production. On the other hand, while the agentic pattern promises to be much more powerful, it's also more complex and is still being explored.

Chapter 7 is about how to adapt a model to an application by changing the model itself with finetuning. Due to the scale of foundation models, native model finetuning is memory-intensive, and many techniques are developed to allow finetuning better models with less memory. The chapter covers different finetuning approaches, supplemented by a more experimental approach: model merging. This chapter contains a more technical section that shows how to calculate the memory footprint of a model.

Due to the availability of many finetuning frameworks, the finetuning process itself is often straightforward. However, getting data for finetuning is hard. The next chapter is all about data, including data acquisition, data annotations, data synthesis, and data processing. Many of the topics discussed in Chapter 8 are relevant beyond finetuning, including the question of what data quality means and how to evaluate the quality of your data.

If Chapters 5 to 8 are about improving a model's quality, Chapter 9 is about making its inference cheaper and faster. It discusses optimization both at the model level and inference service level. If you're using a model API—i.e., someone else hosts your model for you—this API will likely take care of inference optimization for you. However, if you host the model yourself—either an open source model or a model developed in-house—you'll need to implement many of the techniques discussed in this chapter.

The last chapter in the book brings together the different concepts from this book to build an application end-to-end. The second part of the chapter is more product-focused, with discussions on how to design a user feedback system that helps you collect useful feedback while maintaining a good user experience.

I often use "we" in this book to mean you (the reader) and I. It's a habit I got from my teaching days, as I saw writing as a shared learning experience for both the writer and the readers.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, input prompts into models, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/chiphuyen/aie-book*. The repository contains additional resources about AI engineering, including important papers and helpful tools. It also covers topics that are too deep to go into in this book. For those interested in the process of writing this book, the GitHub repository also contains behind-the-scenes information and statistics about the book.

If you have a technical question or a problem using the code examples, please send email to *support@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*AI Engineering* by Chip Huyen (O'Reilly). Copyright 2025 Developer Experience Advisory LLC, 978-1-098-16630-4."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)
support@oreilly.com
https://oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/ai-engineering*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*

Watch us on YouTube: *https://youtube.com/oreillymedia*

Acknowledgments

This book would've taken a lot longer to write and missed many important topics if it wasn't for so many wonderful people who helped me through the process.

Because the timeline for the project was tight—two years for a 150,000-word book that covers so much ground—I'm grateful to the technical reviewers who put aside their precious time to review this book so quickly.

Luke Metz is an amazing soundboard who checked my assumptions and prevented me from going down the wrong path. Han-chung Lee, always up to date with the latest AI news and community development, pointed me toward resources that I had missed. Luke and Han were the first to review my drafts before I sent them to the next round of technical reviewers, and I'm forever indebted to them for tolerating my follies and mistakes.

Having led AI innovation at Fortune 500 companies, Vittorio Cretella and Andrei Lopatenko provided invaluable feedback that combined deep technical expertise with executive insights. Vicki Reyzelman helped me ground my content and keep it relevant for readers with a software engineering background.

Eugene Yan, a dear friend and amazing applied scientist, provided me with technical and emotional support. Shawn Wang (swyx) provided an important vibe check that helped me feel more confident about the book. Sanyam Bhutani, one of the best learners and most humble souls I know, not only gave thoughtful written feedback but also recorded videos to explain his feedback.

Kyle Kranen is a star deep learning lead who interviewed his colleagues and shared with me an amazing writeup about their finetuning process, which guided the finetuning chapter. Mark Saroufim, an inquisitive mind who always has his finger on the pulse of the most interesting problems, introduced me to great resources on efficiency. Both Kyle and Mark's feedback was critical in writing Chapters 7 and 9.

Kittipat "Bot" Kampa, in addition to answering my many questions, shared with me a detailed visualization of how he thinks about AI platforms. I appreciate Denys Linkov's systematic approach to evaluation and platform development. Chetan Tekur gave great examples that helped me structure AI application patterns. I'd also like to thank Shengzhi (Alex) Li and Hien Luu for their thoughtful feedback on my draft on AI architecture.

Aileen Bui is a treasure who shared unique feedback and examples from a product manager's perspective. Thanks to Todor Markov for the actionable advice on the RAG and Agents chapter. Thanks to Tal Kachman for jumping in at the last minute to push the Finetuning chapter over the finish line.

There are so many wonderful people whose company and conversations gave me ideas that guided the content of this book. I tried my best to include the names of everyone who has helped me here, but due to the inherent faultiness of human memory, I undoubtedly neglected to mention many. If I forgot to include your name, please know that it wasn't because I don't appreciate your contribution, and please kindly remind me so that I can rectify this as soon as possible!

Andrew Francis, Anish Nag, Anthony Galczak, Anton Bacaj, Balázs Galambosi, Charles Frye, Charles Packer, Chris Brousseau, Eric Hartford, Goku Mohandas, Hamel Husain, Harpreet Sahota, Hassan El Mghari, Huu Nguyen, Jeremy Howard, Jesse Silver, John Cook, Juan Pablo Bottaro, Kyle Gallatin, Lance Martin, Lucio Dery, Matt Ross, Maxime Labonne, Miles Brundage, Nathan Lambert, Omar Khattab, Phong Nguyen, Purnendu Mukherjee, Sam Reiswig, Sebastian Raschka, Shahul ES, Sharif Shameem, Soumith Chintala, Teknium, Tim Dettmers, Undi95, Val Andrei Fajardo, Vern Liang, Victor Sanh, Wing Lian, Xiquan Cui, Ying Sheng, and Kristofer.

I'd like to thank all early readers who have also reached out with feedback. Douglas Bailley is a super reader who shared so much thoughtful feedback. Thanks to Nutan Sahoo for suggesting an elegant way to explain perplexity.

I learned so much from the online discussions with so many. Thanks to everyone who's ever answered my questions, commented on my posts, or sent me an email with your thoughts.

Of course, the book wouldn't have been possible without the team at O'Reilly, especially my development editors (Melissa Potter, Corbin Collins, Jill Leonard) and my production editor (Elizabeth Kelly). Liz Wheeler is the most discerning copyeditor I've ever worked with. Nicole Butterfield is a force who oversaw this book from an idea to a final product.

This book, after all, is an accumulation of invaluable lessons I learned throughout my career. I owe these lessons to my extremely competent and patient coworkers and former coworkers. Every person I've worked with has taught me something new about bringing ML into the world.