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 (*https://oreil.ly/XG3mv*) 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 (*https://x.com/chipro/status/937384141791698944*), 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 (*https://huyenchip.com/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 (*https://x.com/chipro*), LinkedIn (*https://www.linkedin.com/in/chiphuyen*), 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 (*https://en.wikipedia.org/wiki/Lindy_effect*), 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 trade-offs 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*.

CHAPTER 1

Introduction to Building AI Applications with Foundation Models

If I could use only one word to describe AI post-2020, it'd be *scale*. The AI models behind applications like ChatGPT, Google's Gemini, and Midjourney are at such a scale that they're consuming a nontrivial portion (*https://oreil.ly/J0IyO*) of the world's electricity, and we're at risk of running out of publicly available internet data (*https://arxiv.org/abs/2211.04325*) to train them.

The scaling up of AI models has two major consequences. First, AI models are becoming more powerful and capable of more tasks, enabling more applications. More people and teams leverage AI to increase productivity, create economic value, and improve quality of life.

Second, training large language models (LLMs) requires data, compute resources, and specialized talent that only a few organizations can afford. This has led to the emergence of *model as a service*: models developed by these few organizations are made available for others to use as a service. Anyone who wishes to leverage AI to build applications can now use these models to do so without having to invest up front in building a model.

In short, the demand for AI applications has increased while the barrier to entry for building AI applications has decreased. This has turned *AI engineering*—the process of building applications on top of readily available models—into one of the fastest-growing engineering disciplines.

Building applications on top of machine learning (ML) models isn't new. Long before LLMs became prominent, AI was already powering many applications, including product recommendations, fraud detection, and churn prediction. While many principles of productionizing AI applications remain the same, the new generation of

large-scale, readily available models brings about new possibilities and new challenges, which are the focus of this book.

This chapter begins with an overview of foundation models, the key catalyst behind the explosion of AI engineering. I'll then discuss a range of successful AI use cases, each illustrating what AI is good and not yet good at. As AI's capabilities expand daily, predicting its future possibilities becomes increasingly challenging. However, existing application patterns can help uncover opportunities today and offer clues about how AI may continue to be used in the future.

To close out the chapter, I'll provide an overview of the new AI stack, including what has changed with foundation models, what remains the same, and how the role of an AI engineer today differs from that of a traditional ML engineer.[1]

The Rise of AI Engineering

Foundation models emerged from large language models, which, in turn, originated as just language models. While applications like ChatGPT and GitHub's Copilot may seem to have come out of nowhere, they are the culmination of decades of technology advancements, with the first language models emerging in the 1950s. This section traces the key breakthroughs that enabled the evolution from language models to AI engineering.

From Language Models to Large Language Models

While language models have been around for a while, they've only been able to grow to the scale they are today with *self-supervision*. This section gives a quick overview of what language model and self-supervision mean. If you're already familiar with those, feel free to skip this section.

Language models

A *language model* encodes statistical information about one or more languages. Intuitively, this information tells us how likely a word is to appear in a given context. For example, given the context "My favorite color is __", a language model that encodes English should predict "blue" more often than "car".

1 In this book, I use *traditional ML* to refer to all ML before foundation models.

The statistical nature of languages was discovered centuries ago. In the 1905 story "The Adventure of the Dancing Men" (*https://en.wikipedia.org/wiki/The_Adventure_of_the_Dancing_Men*), Sherlock Holmes leveraged simple statistical information of English to decode sequences of mysterious stick figures. Since the most common letter in English is *E*, Holmes deduced that the most common stick figure must stand for *E*.

Later on, Claude Shannon used more sophisticated statistics to decipher enemies' messages during the Second World War. His work on how to model English was published in his 1951 landmark paper "Prediction and Entropy of Printed English" (*https://oreil.ly/G_HBp*). Many concepts introduced in this paper, including entropy, are still used for language modeling today.

In the early days, a language model involved one language. However, today, a language model can involve multiple languages.

The basic unit of a language model is *token*. A token can be a character, a word, or a part of a word (like -tion), depending on the model.[2] For example, GPT-4, a model behind ChatGPT, breaks the phrase "I can't wait to build AI applications" into nine tokens, as shown in Figure 1-1. Note that in this example, the word "can't" is broken into two tokens, *can* and *'t*. You can see how different OpenAI models tokenize text on the OpenAI website (*https://oreil.ly/0QI91*).

```
I can't wait to build awesome AI applications
```

Figure 1-1. An example of how GPT-4 tokenizes a phrase.

The process of breaking the original text into tokens is called *tokenization*. For GPT-4, an average token is approximately ¾ the length of a word (*https://oreil.ly/EYccr*). So, 100 tokens are approximately 75 words.

The set of all tokens a model can work with is the model's *vocabulary*. You can use a small number of tokens to construct a large number of distinct words, similar to how you can use a few letters in the alphabet to construct many words. The Mixtral 8x7B (*https://oreil.ly/bxMcW*) model has a vocabulary size of 32,000. GPT-4's vocabulary size is 100,256 (*https://github.com/openai/tiktoken/blob/main/tiktoken/model.py*). The tokenization method and vocabulary size are decided by model developers.

2 For non-English languages, a single Unicode character can sometimes be represented as multiple tokens.

Why do language models use *token* as their unit instead of *word* or *character*? There are three main reasons:

1. Compared to characters, tokens allow the model to break words into meaningful components. For example, "cooking" can be broken into "cook" and "ing", with both components carrying some meaning of the original word.
2. Because there are fewer unique tokens than unique words, this reduces the model's vocabulary size, making the model more efficient (as discussed in Chapter 2).
3. Tokens also help the model process unknown words. For instance, a made-up word like "chatgpting" could be split into "chatgpt" and "ing", helping the model understand its structure. Tokens balance having fewer units than words while retaining more meaning than individual characters.

There are two main types of language models: *masked language models* and *autoregressive language models*. They differ based on what information they can use to predict a token:

Masked language model

A masked language model is trained to predict missing tokens anywhere in a sequence, *using the context from both before and after the missing tokens*. In essence, a masked language model is trained to be able to fill in the blank. For example, given the context, "My favorite __ is blue", a masked language model should predict that the blank is likely "color". A well-known example of a masked language model is bidirectional encoder representations from transformers, or BERT (Devlin et al., 2018 (*https://arxiv.org/abs/1810.04805*)).

As of writing, masked language models are commonly used for non-generative tasks such as sentiment analysis and text classification. They are also useful for tasks requiring an understanding of the overall context, such as code debugging, where a model needs to understand both the preceding and following code to identify errors.

Autoregressive language model

An autoregressive language model is trained to predict the next token in a sequence, *using only the preceding tokens*. It predicts what comes next in "My favorite color is __."[3] An autoregressive model can continually generate one token after another. Today, autoregressive language models are the models of

3 Autoregressive language models are sometimes referred to as causal language models (*https://oreil.ly/h0Y8x*).

choice for text generation, and for this reason, they are much more popular than masked language models.[4]

Figure 1-2 shows these two types of language models.

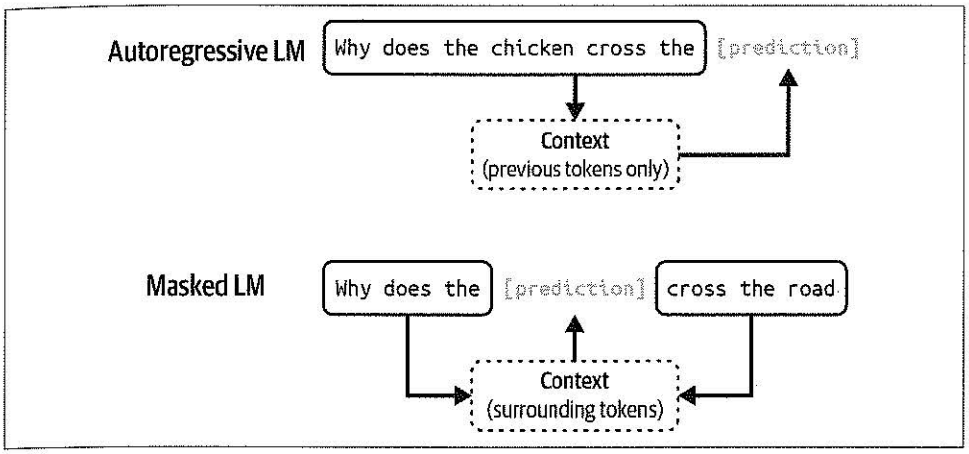


Figure 1-2. Autoregressive language model and masked language model.

In this book, unless explicitly stated, *language model* will refer to an autoregressive model.

The outputs of language models are open-ended. A language model can use its fixed, finite vocabulary to construct infinite possible outputs. A model that can generate open-ended outputs is called *generative*, hence the term *generative AI*.

You can think of a language model as a *completion machine*: given a text (prompt), it tries to complete that text. Here's an example:

Prompt (from user): "`To be or not to be`"

Completion (from language model): "`, that is the question.`"

It's important to note that completions are predictions, based on probabilities, and not guaranteed to be correct. This probabilistic nature of language models makes them both so exciting and frustrating to use. We explore this further in Chapter 2.

4 Technically, a masked language model like BERT can also be used for text generations if you try really hard.

As simple as it sounds, completion is incredibly powerful. Many tasks, including translation, summarization, coding, and solving math problems, can be framed as completion tasks. For example, given the prompt: "How are you in French is ...", a language model might be able to complete it with: "Comment ça va", effectively translating from one language to another.

As another example, given the prompt:

```
Question: Is this email likely spam? Here's the email: <email content>
Answer:
```

A language model might be able to complete it with: "Likely spam", which turns this language model into a spam classifier.

While completion is powerful, completion isn't the same as engaging in a conversation. For example, if you ask a completion machine a question, it can complete what you said by adding another question instead of answering the question. "Post-Training" on page 78 discusses how to make a model respond appropriately to a user's request.

Self-supervision

Language modeling is just one of many ML algorithms. There are also models for object detection, topic modeling, recommender systems, weather forecasting, stock price prediction, etc. What's special about language models that made them the center of the scaling approach that caused the ChatGPT moment?

The answer is that language models can be trained using *self-supervision*, while many other models require *supervision*. Supervision refers to the process of training ML algorithms using labeled data, which can be expensive and slow to obtain. Self-supervision helps overcome this data labeling bottleneck to create larger datasets for models to learn from, effectively allowing models to scale up. Here's how.

With supervision, you label examples to show the behaviors you want the model to learn, and then train the model on these examples. Once trained, the model can be applied to new data. For example, to train a fraud detection model, you use examples of transactions, each labeled with "fraud" or "not fraud". Once the model learns from these examples, you can use this model to predict whether a transaction is fraudulent.

The success of AI models in the 2010s lay in supervision. The model that started the deep learning revolution, AlexNet (Krizhevsky et al., 2012 (*https://oreil.ly/WEQFj*)), was supervised. It was trained to learn how to classify over 1 million images in the dataset ImageNet. It classified each image into one of 1,000 categories such as "car", "balloon", or "monkey".

A drawback of supervision is that data labeling is expensive and time-consuming. If it costs 5 cents for one person to label one image, it'd cost $50,000 to label a million images for ImageNet.[5] If you want two different people to label each image—so that you could cross-check label quality—it'd cost twice as much. Because the world contains vastly more than 1,000 objects, to expand models' capabilities to work with more objects, you'd need to add labels of more categories. To scale up to 1 million categories, the labeling cost alone would increase to $50 million.

Labeling everyday objects is something that most people can do without prior training. Hence, it can be done relatively cheaply. However, not all labeling tasks are that simple. Generating Latin translations for an English-to-Latin model is more expensive. Labeling whether a CT scan shows signs of cancer would be astronomical.

Self-supervision helps overcome the data labeling bottleneck. In self-supervision, instead of requiring explicit labels, the model can infer labels from the input data. Language modeling is self-supervised because each input sequence provides both the labels (tokens to be predicted) and the contexts the model can use to predict these labels. For example, the sentence "I love street food." gives six training samples, as shown in Table 1-1.

Table 1-1. Training samples from the sentence "I love street food." for language modeling.

Input (context)	Output (next token)
<BOS>	I
<BOS>, I	love
<BOS>, I, love	street
<BOS>, I, love, street	food
<BOS>, I, love, street, food	.
<BOS>, I, love, street, food, .	<EOS>

In Table 1-1, <BOS> and <EOS> mark the beginning and the end of a sequence. These markers are necessary for a language model to work with multiple sequences. Each marker is typically treated as one special token by the model. The end-of-sequence marker is especially important as it helps language models know when to end their responses.[6]

5 The actual data labeling cost varies depending on several factors, including the task's complexity, the scale (larger datasets typically result in lower per-sample costs), and the labeling service provider. For example, as of September 2024, Amazon SageMaker Ground Truth (*https://oreil.ly/EVXJl*) charges 8 cents per image for labeling fewer than 50,000 images, but only 2 cents per image for labeling more than 1 million images.

6 This is similar to how it's important for humans to know when to stop talking.

Self-supervision differs from unsupervision. In self-supervised learning, labels are inferred from the input data. In unsupervised learning, you don't need labels at all.

Self-supervised learning means that language models can learn from text sequences without requiring any labeling. Because text sequences are everywhere—in books, blog posts, articles, and Reddit comments—it's possible to construct a massive amount of training data, allowing language models to scale up to become LLMs.

LLM, however, is hardly a scientific term. How large does a language model have to be to be considered *large*? What is large today might be considered tiny tomorrow. A model's size is typically measured by its number of parameters. A *parameter* is a variable within an ML model that is updated through the training process.[7] In general, though this is not always true, the more parameters a model has, the greater its capacity to learn desired behaviors.

When OpenAI's first generative pre-trained transformer (GPT) model came out in June 2018, it had 117 million parameters, and that was considered large. In February 2019, when OpenAI introduced GPT-2 with 1.5 billion parameters, 117 million was downgraded to be considered small. As of the writing of this book, a model with 100 billion parameters is considered large. Perhaps one day, this size will be considered small.

Before we move on to the next section, I want to touch on a question that is usually taken for granted: *Why do larger models need more data?* Larger models have more capacity to learn, and, therefore, would need more training data to maximize their performance.[8] You can train a large model on a small dataset too, but it'd be a waste of compute. You could have achieved similar or better results on this dataset with smaller models.

From Large Language Models to Foundation Models

While language models are capable of incredible tasks, they are limited to text. As humans, we perceive the world not just via language but also through vision, hearing, touch, and more. Being able to process data beyond text is essential for AI to operate in the real world.

7 In school, I was taught that model parameters include both model weights and model biases. However, today, we generally use model weights to refer to all parameters.

8 It seems counterintuitive that larger models require more training data. If a model is more powerful, shouldn't it require fewer examples to learn from? However, we're not trying to get a large model to match the performance of a small model using the same data. We're trying to maximize model performance.

For this reason, language models are being extended to incorporate more data modalities. GPT-4V and Claude 3 can understand images and texts. Some models even understand videos, 3D assets, protein structures, and so on. Incorporating more data modalities into language models makes them even more powerful. OpenAI noted in their GPT-4V system card (*https://oreil.ly/NoGX7*) in 2023 that "incorporating additional modalities (such as image inputs) into LLMs is viewed by some as a key frontier in AI research and development."

While many people still call Gemini and GPT-4V LLMs, they're better characterized as *foundation models* (*https://arxiv.org/abs/2108.07258*). The word *foundation* signifies both the importance of these models in AI applications and the fact that they can be built upon for different needs.

Foundation models mark a breakthrough from the traditional structure of AI research. For a long time, AI research was divided by data modalities. Natural language processing (NLP) deals only with text. Computer vision deals only with vision. Text-only models can be used for tasks such as translation and spam detection. Image-only models can be used for object detection and image classification. Audio-only models can handle speech recognition (speech-to-text, or STT) and speech synthesis (text-to-speech, or TTS).

A model that can work with more than one data modality is also called a *multimodal model*. A generative multimodal model is also called a large multimodal model (LMM). If a language model generates the next token conditioned on text-only tokens, a multimodal model generates the next token conditioned on both text and image tokens, or whichever modalities that the model supports, as shown in Figure 1-3.

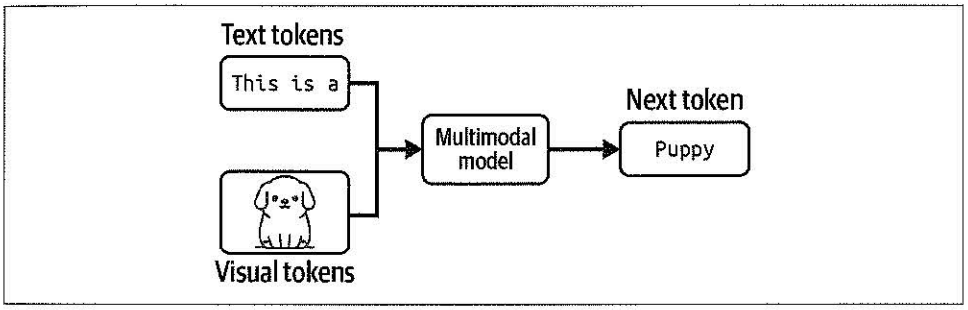


Figure 1-3. A multimodal model can generate the next token using information from both text and visual tokens.

Just like language models, multimodal models need data to scale up. Self-supervision works for multimodal models too. For example, OpenAI used a variant of self-supervision called *natural language supervision* to train their language-image model CLIP (OpenAI, 2021) (*https://oreil.ly/zcqdu*). Instead of manually generating labels for each image, they found (image, text) pairs that co-occurred on the internet. They were able to generate a dataset of 400 million (image, text) pairs, which was 400 times larger than ImageNet, without manual labeling cost. This dataset enabled CLIP to become the first model that could generalize to multiple image classification tasks without requiring additional training.

This book uses the term foundation models to refer to both large language models and large multimodal models.

Note that CLIP isn't a generative model—it wasn't trained to generate open-ended outputs. CLIP is an *embedding model*, trained to produce joint embeddings of both texts and images. "Introduction to Embedding" on page 134 discusses embeddings in detail. For now, you can think of embeddings as vectors that aim to capture the meanings of the original data. Multimodal embedding models like CLIP are the backbones of generative multimodal models, such as Flamingo, LLaVA, and Gemini (previously Bard).

Foundation models also mark the transition from task-specific models to general-purpose models. Previously, models were often developed for specific tasks, such as sentiment analysis or translation. A model trained for sentiment analysis wouldn't be able to do translation, and vice versa.

Foundation models, thanks to their scale and the way they are trained, are capable of a wide range of tasks. Out of the box, general-purpose models can work relatively well for many tasks. An LLM can do both sentiment analysis and translation. However, you can often tweak a general-purpose model to maximize its performance on a specific task.

Figure 1-4 shows the tasks used by the Super-NaturalInstructions benchmark to evaluate foundation models (Wang et al., 2022 (*https://arxiv.org/abs/2204.07705*)), providing an idea of the types of tasks a foundation model can perform.

Imagine you're working with a retailer to build an application to generate product descriptions for their website. An out-of-the-box model might be able to generate accurate descriptions but might fail to capture the brand's voice or highlight the brand's messaging. The generated descriptions might even be full of marketing speech and cliches.

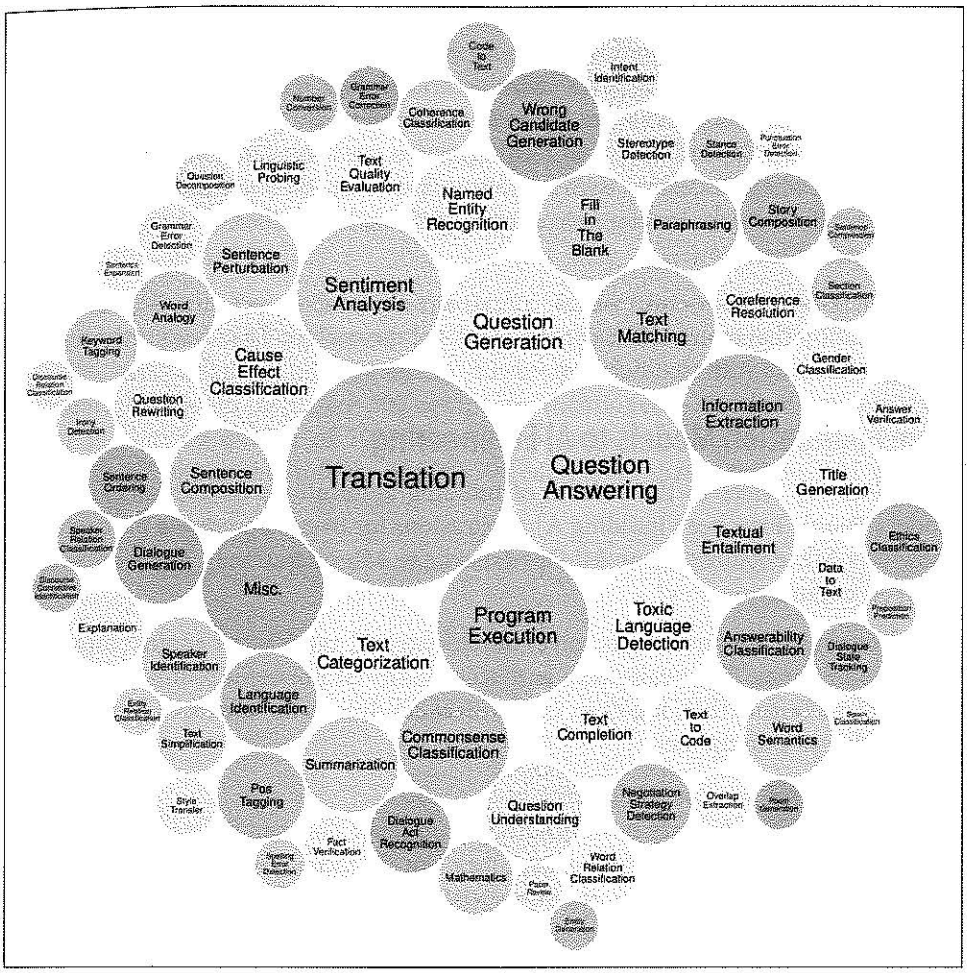


Figure 1-4. The range of tasks in the Super-NaturalInstructions benchmark (Wang et al., 2022).

There are multiple techniques you can use to get the model to generate what you want. For example, you can craft detailed instructions with examples of the desirable product descriptions. This approach is *prompt engineering*. You can connect the model to a database of customer reviews that the model can leverage to generate better descriptions. Using a database to supplement the instructions is called *retrieval-augmented generation* (RAG). You can also *finetune*—further train—the model on a dataset of high-quality product descriptions.

Prompt engineering, RAG, and finetuning are three very common AI engineering techniques that you can use to adapt a model to your needs. The rest of the book will discuss all of them in detail.

Adapting an existing powerful model to your task is generally a lot easier than building a model for your task from scratch—for example, ten examples and one weekend versus 1 million examples and six months. Foundation models make it cheaper to develop AI applications and reduce time to market. Exactly how much data is needed to adapt a model depends on what technique you use. This book will also touch on this question when discussing each technique. However, there are still many benefits to task-specific models, for example, they might be a lot smaller, making them faster and cheaper to use.

Whether to build your own model or leverage an existing one is a classic buy-or-build question that teams will have to answer for themselves. Discussions throughout the book can help with that decision.

From Foundation Models to AI Engineering

AI engineering refers to the process of building applications on top of foundation models. People have been building AI applications for over a decade—a process often known as ML engineering or MLOps (short for ML operations). Why do we talk about AI engineering now?

If traditional ML engineering involves developing ML models, AI engineering leverages existing ones. The availability and accessibility of powerful foundation models lead to three factors that, together, create ideal conditions for the rapid growth of AI engineering as a discipline:

Factor 1: General-purpose AI capabilities

Foundation models are powerful not just because they can do existing tasks better. They are also powerful because they can do more tasks. Applications previously thought impossible are now possible, and applications not thought of before are emerging. Even applications not thought possible today might be possible tomorrow. This makes AI more useful for more aspects of life, vastly increasing both the user base and the demand for AI applications.

For example, since AI can now write as well as humans, sometimes even better, AI can automate or partially automate every task that requires communication, which is pretty much everything. AI is used to write emails, respond to customer requests, and explain complex contracts. Anyone with a computer has access to tools that can instantly generate customized, high-quality images and videos to help create marketing materials, edit professional headshots, visualize art concepts, illustrate books, and so on. AI can even be used to synthesize training data, develop algorithms, and write code, all of which will help train even more powerful models in the future.

Factor 2: Increased AI investments

The success of ChatGPT prompted a sharp increase in investments in AI, both from venture capitalists and enterprises. As AI applications become cheaper to build and faster to go to market, returns on investment for AI become more attractive. Companies rush to incorporate AI into their products and processes. Matt Ross, a senior manager of applied research at Scribd, told me that the estimated AI cost for his use cases has gone down two orders of magnitude from April 2022 to April 2023.

Goldman Sachs Research (*https://oreil.ly/okMw6*) estimated that AI investment could approach $100 billion in the US and $200 billion globally by 2025.[9] AI is often mentioned as a competitive advantage. FactSet (*https://oreil.ly/tgm-a*) found that one in three S&P 500 companies mentioned AI in their earnings calls for the second quarter of 2023, three times more than did so the year earlier. Figure 1-5 shows the number of S&P 500 companies that mentioned AI in their earning calls from 2018 to 2023.

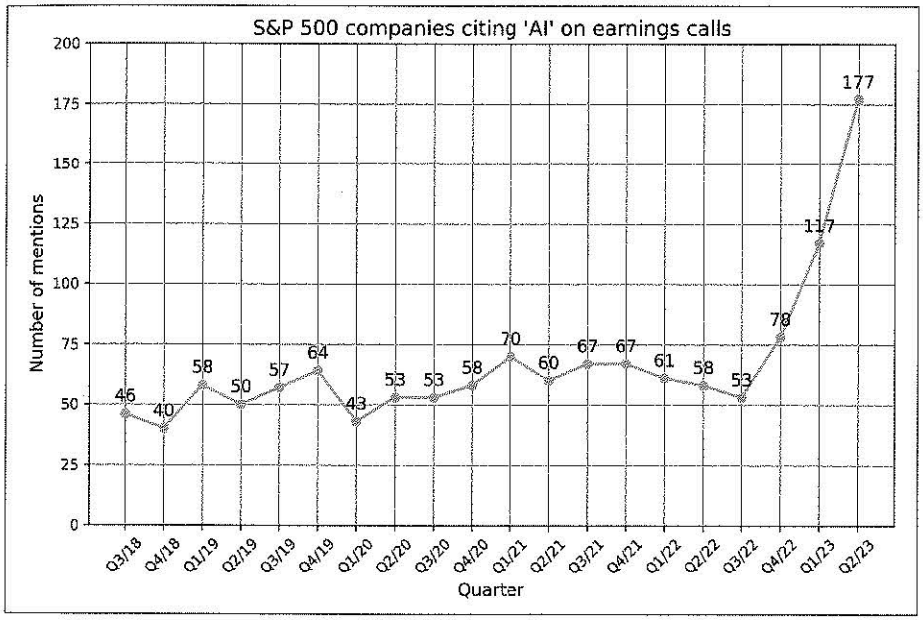


Figure 1-5. The number of S&P 500 companies that mention AI in their earnings calls reached a record high in 2023. Data from FactSet.

9 For comparison, the entire US expenditures for public elementary and secondary schools are around $900 billion, only nine times the investments in AI in the US.

According to WallStreetZen, companies that mentioned AI in their earning calls saw their stock price increase more than those that didn't: an average of a 4.6% increase compared to 2.4% (*https://oreil.ly/fK5uh*). It's unclear whether it's causation (AI makes these companies more successful) or correlation (companies are successful because they are quick to adapt to new technologies).

Factor 3: Low entrance barrier to building AI applications

The model as a service approach popularized by OpenAI and other model providers makes it easier to leverage AI to build applications. In this approach, models are exposed via APIs that receive user queries and return model outputs. Without these APIs, using an AI model requires the infrastructure to host and serve this model. These APIs give you access to powerful models via single API calls.

Not only that, AI also makes it possible to build applications with minimal coding. First, AI can write code for you, allowing people without a software engineering background to quickly turn their ideas into code and put them in front of their users. Second, you can work with these models in plain English instead of having to use a programming language. *Anyone, and I mean anyone, can now develop AI applications.*

Because of the resources it takes to develop foundation models, this process is possible only for big corporations (Google, Meta, Microsoft, Baidu, Tencent), governments (Japan (*https://oreil.ly/r86Qz*), the UAE (*https://oreil.ly/IUcVg*)), and ambitious, well-funded startups (OpenAI, Anthropic, Mistral). In a September 2022 interview, Sam Altman, CEO of OpenAI (*https://oreil.ly/D9QBM*), said that the biggest opportunity for the vast majority of people will be to adapt these models for specific applications.

The world is quick to embrace this opportunity. AI engineering has rapidly emerged as one of the fastest, and quite possibly the fastest-growing, engineering discipline. Tools for AI engineering are gaining traction faster than any previous software engineering tools. Within just two years, four open source AI engineering tools (AutoGPT, Stable Diffusion eb UI, LangChain, Ollama) have already garnered more stars on GitHub than Bitcoin. They are on track to surpass even the most popular web development frameworks, including React and Vue, in star count. Figure 1-6 shows the GitHub star growth of AI engineering tools compared to Bitcoin, Vue, and React.

A LinkedIn survey from August 2023 shows that the number of professionals adding terms like "Generative AI," "ChatGPT," "Prompt Engineering," and "Prompt Crafting" to their profile increased on average 75% each month (*https://oreil.ly/m8SvB*). *ComputerWorld* (*https://oreil.ly/47sGE*) declared that "teaching AI to behave is the fastest-growing career skill".

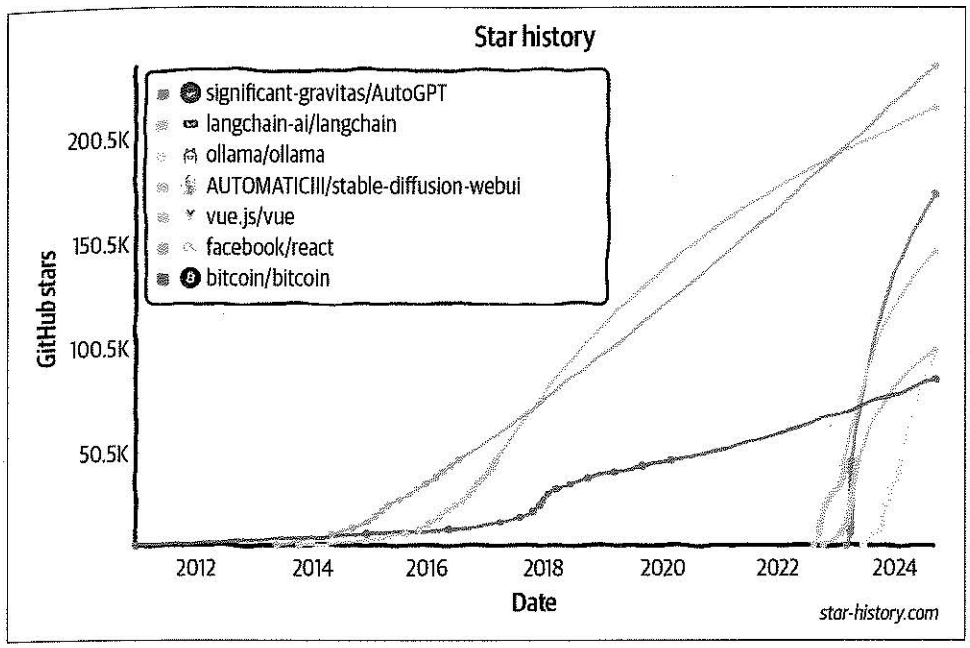


Figure 1-6. Open source AI engineering tools are growing faster than any other software engineering tools, according to their GitHub star counts.

Why the Term "AI Engineering?"

Many terms are being used to describe the process of building applications on top of foundation models, including ML engineering, MLOps, AIOps, LLMOps, etc. Why did I choose to go with AI engineering for this book?

I didn't go with the term ML engineering because, as discussed in "AI Engineering Versus ML Engineering" on page 39, working with foundation models differs from working with traditional ML models in several important aspects. The term ML engineering won't be sufficient to capture this differentiation. However, ML engineering is a great term to encompass both processes.

I didn't go with all the terms that end with "Ops" because, while there are operational components of the process, the focus is more on tweaking (engineering) foundation models to do what you want.

Finally, I surveyed 20 people who were developing applications on top of foundation models about what term they would use to describe what they were doing. Most people preferred *AI engineering*. I decided to go with the people.

The rapidly expanding community of AI engineers has demonstrated remarkable creativity with an incredible range of exciting applications. The next section will explore some of the most common application patterns.

Foundation Model Use Cases

If you're not already building AI applications, I hope the previous section has convinced you that now is a great time to do so. If you have an application in mind, you might want to jump to "Planning AI Applications" on page 28. If you're looking for inspiration, this section covers a wide range of industry-proven and promising use cases.

The number of potential applications that you could build with foundation models seems endless. Whatever use case you think of, there's probably an AI for that.[10] It's impossible to list all potential use cases for AI.

Even attempting to categorize these use cases is challenging, as different surveys use different categorizations. For example, Amazon Web Services (AWS) (*https://oreil.ly/-k_QX*) has categorized enterprise generative AI use cases into three buckets: customer experience, employee productivity, and process optimization. A 2024 O'Reilly survey categorized the use cases into eight categories: programming, data analysis, customer support, marketing copy, other copy, research, web design, and art.

Some organizations, like Deloitte (*https://oreil.ly/T272_*), have categorized use cases by value capture, such as cost reduction, process efficiency, growth, and accelerating innovation. For value capture, Gartner (*https://oreil.ly/OyIUP*) has a category for *business continuity*, meaning an organization might go out of business if it doesn't adopt generative AI. Of the 2,500 executives Gartner surveyed in 2023, 7% cited business continuity as the motivation for embracing generative AI.

10 Fun fact: as of September 16, 2024, the website *theresanaiforthat.com* lists 16,814 AIs for 14,688 tasks and 4,803 jobs.

Eloundou et al. (2023) (*https://arxiv.org/abs/2303.10130*) has excellent research on how exposed different occupations are to AI. They defined a task as exposed if AI and AI-powered software can reduce the time needed to complete this task by at least 50%. An occupation with 80% exposure means that 80% of the occupation's tasks are exposed. According to the study, occupations with 100% or close to 100% exposure include interpreters and translators, tax preparers, web designers, and writers. Some of them are shown in Table 1-2. Not unsurprisingly, occupations with no exposure to AI include cooks, stonemasons, and athletes. This study gives a good idea of what use cases AI is good for.

Table 1-2. Occupations with the highest exposure to AI as annotated by humans. α refers to exposure to AI models directly, whereas β and ζ refer to exposures to AI-powered software. Table from Eloundou et al. (2023).

Group	Occupations with highest exposure	% Exposure
Human α	Interpreters and translators	76.5
	Survey researchers	75.0
	Poets, lyricists, and creative writers	68.8
	Animal scientists	66.7
	Public relations specialists	66.7
Human β	Survey researchers	84.4
	Writers and authors	82.5
	Interpreters and translators	82.4
	Public relations specialists	80.6
	Animal scientists	77.8
Human ζ	Mathematicians	100.0
	Tax preparers	100.0
	Financial quantitative analysts	100.0
	Writers and authors	100.0
	Web and digital interface designers	100.0
	Humans labeled 15 occupations as "fully exposed".	

When analyzing the use cases, I looked at both enterprise and consumer applications. To understand enterprise use cases, I interviewed 50 companies on their AI strategies and read over 100 case studies. To understand consumer applications, I examined 205 open source AI applications with at least 500 stars on GitHub.[11] I categorized applications into eight groups, as shown in Table 1-3. The limited list here serves best as a reference. As you learn more about how to build foundation models in Chapter 2 and how to evaluate them in Chapter 3, you'll also be able to form a better picture of what use cases foundation models can and should be used for.

Table 1-3. Common generative AI use cases across consumer and enterprise applications.

Category	Examples of consumer use cases	Examples of enterprise use cases
Coding	Coding	Coding
Image and video production	Photo and video editing Design	Presentation Ad generation
Writing	Email Social media and blog posts	Copywriting, search engine optimization (SEO) Reports, memos, design docs
Education	Tutoring Essay grading	Employee onboarding Employee upskill training
Conversational bots	General chatbot AI companion	Customer support Product copilots
Information aggregation	Summarization Talk-to-your-docs	Summarization Market research
Data organization	Image search Memex	Knowledge management Document processing
Workflow automation	Travel planning Event planning	Data extraction, entry, and annotation Lead generation

Because foundation models are general, applications built on top of them can solve many problems. This means that an application can belong to more than one category. For example, a bot can provide companionship and aggregate information. An application can help you extract structured data from a PDF and answer questions about that PDF.

Figure 1-7 shows the distribution of these use cases among the 205 open source applications. Note that the small percentage of education, data organization, and writing use cases doesn't mean that these use cases aren't popular. It just means that these applications aren't open source. Builders of these applications might find them more suitable for enterprise use cases.

11 Exploring different AI applications is perhaps one of my favorite things about writing this book. It's a lot of fun seeing what people are building. You can find the list of open source AI applications (*https://huyenchip.com/llama-police*) that I track. The list is updated every 12 hours.

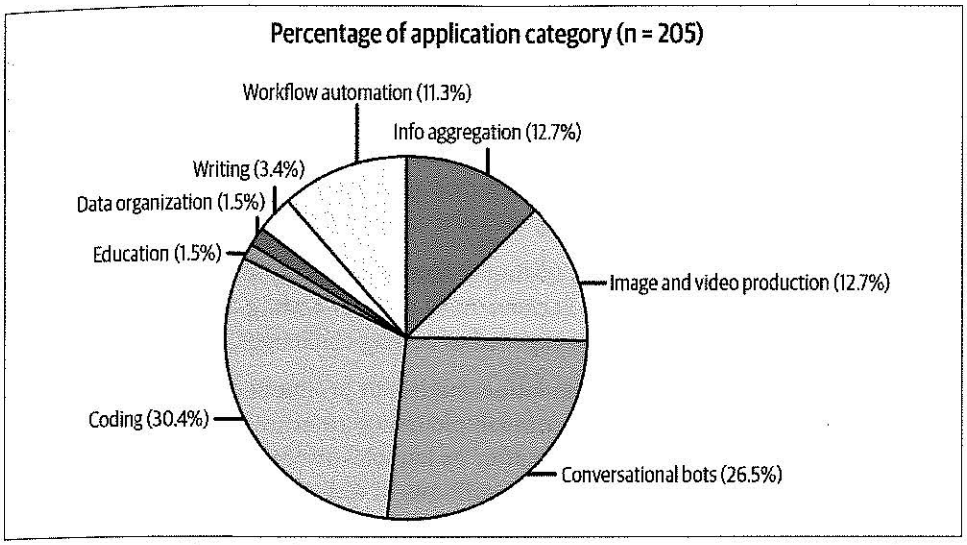


Figure 1-7. Distribution of use cases in the 205 open source repositories on GitHub.

The enterprise world generally prefers applications with lower risks. For example, a 2024 a16z Growth report (*https://oreil.ly/XWeDt*) showed that companies are faster to deploy internal-facing applications (internal knowledge management) than external-facing applications (customer support chatbots), as shown in Figure 1-8. Internal applications help companies develop their AI engineering expertise while minimizing the risks associated with data privacy, compliance, and potential catastrophic failures. Similarly, while foundation models are open-ended and can be used for any task, many applications built on top of them are still close-ended, such as classification. Classification tasks are easier to evaluate, which makes their risks easier to estimate.

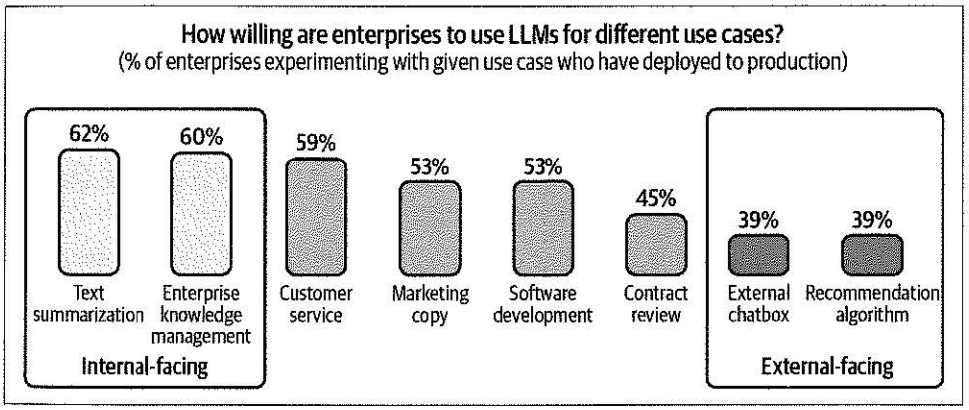


Figure 1-8. Companies are more willing to deploy internal-facing applications

Even after seeing hundreds of AI applications, I still find new applications that surprise me every week. In the early days of the internet, few people foresaw that the dominating use case on the internet one day would be social media. As we learn to make the most out of AI, the use case that will eventually dominate might surprise us. With luck, the surprise will be a good one.

Coding

In multiple generative AI surveys, coding is hands down the most popular use case. AI coding tools are popular both because AI is good at coding and because early AI engineers are coders who are more exposed to coding challenges.

One of the earliest successes of foundation models in production is the code completion tool GitHub Copilot, whose annual recurring revenue crossed $100 million (*https://oreil.ly/Xamik*) only two years after its launch. As of this writing, AI-powered coding startups have raised hundreds of millions of dollars, with Magic raising $320 million (*https://oreil.ly/t0xDf*) and Anysphere raising $60 million (*https://oreil.ly/BW5Hk*), both in August 2024. Open source coding tools like gpt-engineer (*https://github.com/gpt-engineer-org/gpt-engineer*) and screenshot-to-code (*https://github.com/abi/screenshot-to-code*) both got 50,000 stars on GitHub within a year, and many more are being rapidly introduced.

Other than tools that help with general coding, many tools specialize in certain coding tasks. Here are examples of these tasks:

- Extracting structured data from web pages and PDFs (AgentGPT (*https://github.com/reworkd/AgentGPT*))
- Converting English to code (DB-GPT (*https://github.com/eosphoros-ai/DB-GPT*), SQL Chat (*https://github.com/sqlchat/sqlchat*), PandasAI (*https://github.com/Sinaptik-AI/pandas-ai*))
- Given a design or a screenshot, generating code that will render into a website that looks like the given image (screenshot-to-code, draw-a-ui (*https://github.com/sawyerhood/draw-a-ui*))
- Translating from one programming language or framework to another (GPT-Migrate (*https://github.com/joshpxyne/gpt-migrate*), AI Code Translator (*https://github.com/mckaywrigley/ai-code-translator*))
- Writing documentation (Autodoc (*https://github.com/context-labs/autodoc*))
- Creating tests (PentestGPT (*https://github.com/GreyDGL/PentestGPT*))
- Generating commit messages (AI Commits (*https://github.com/Nutlope/aicommits*))

It's clear that AI can do many software engineering tasks. The question is whether AI can automate software engineering altogether. At one end of the spectrum, Jensen Huang, CEO of NVIDIA (*https://oreil.ly/zUpGu*), predicts that AI will replace human software engineers and that we should stop saying kids should learn to code. In a leaked recording, AWS CEO Matt Garman (*https://oreil.ly/Hz_3i*) shared that in the near future, most developers will stop coding. He doesn't mean it as the end of software developers; it's just that their jobs will change.

At the other end are many software engineers who are convinced that they will never be replaced by AI, both for technical and emotional reasons (people don't like admitting that they can be replaced).

Software engineering consists of many tasks. AI is better at some than others. McKinsey (*https://oreil.ly/aqUmX*) researchers found that AI can help developers be twice as productive for documentation, and 25–50% more productive for code generation and code refactoring. Minimal productivity improvement was observed for highly complex tasks, as shown in Figure 1-9. In my conversations with developers of AI coding tools, many told me that they've noticed that AI is much better at frontend development than backend development.

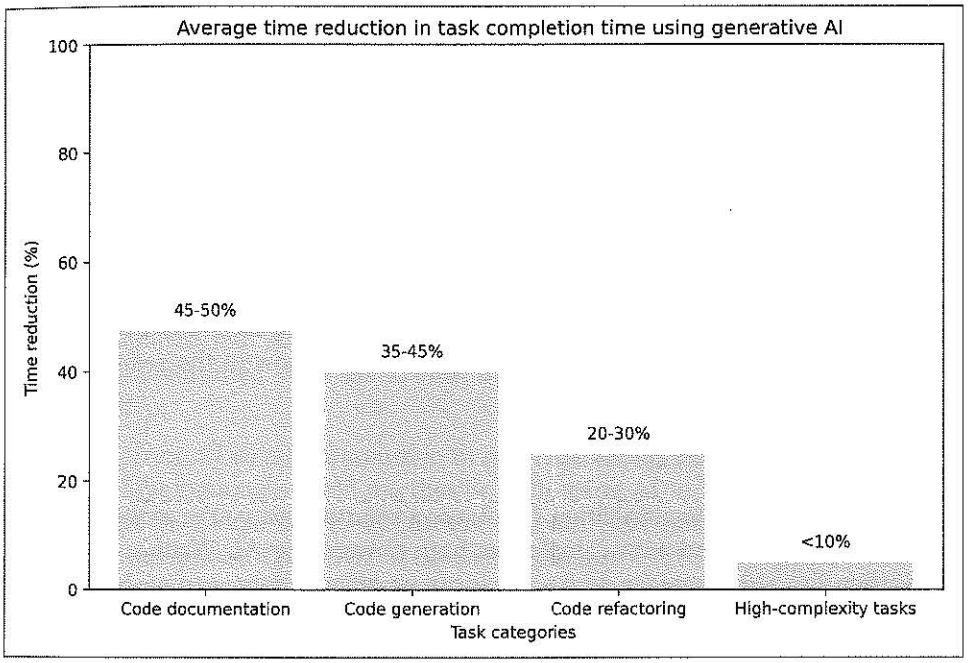


Figure 1-9. AI can help developers be significantly more productive, especially for simple tasks, but this applies less for highly complex tasks. Data by McKinsey.

Regardless of whether AI will replace software engineers, AI can certainly make them more productive. This means that companies can now accomplish more with fewer engineers. AI can also disrupt the outsourcing industry, as outsourced tasks tend to be simpler ones outside of a company's core business.

Image and Video Production

Thanks to its probabilistic nature, AI is great for creative tasks. Some of the most successful AI startups are creative applications, such as Midjourney for image generation, Adobe Firefly for photo editing, and Runway, Pika Labs, and Sora for video generation. In late 2023, at one and a half years old, Midjourney (*https://oreil.ly/EAzCI*) had already generated $200 million in annual recurring revenue. As of December 2023, among the top 10 free apps for Graphics & Design on the Apple App Store, half have AI in their names. I suspect that soon, graphics and design apps will incorporate AI by default, and they'll no longer need the word "AI" in their names. Chapter 2 discusses the probabilistic nature of AI in more detail.

It's now common to use AI to generate profile pictures for social media, from LinkedIn to TikTok. Many candidates believe that AI-generated headshots can help them put their best foot forward and increase their chances of landing a job (*https://oreil.ly/fZLVg*). The perception of AI-generated profile pictures has changed significantly. In 2019, Facebook (*https://oreil.ly/WNqUw*) banned accounts using AI-generated profile photos for safety reasons. In 2023, many social media apps provide tools that let users use AI to generate profile photos.

For enterprises, ads and marketing have been quick to incorporate AI.[12] AI can be used to generate promotional images and videos directly. It can help brainstorm ideas or generate first drafts for human experts to iterate upon. You can use AI to generate multiple ads and test to see which one works the best for the audience. AI can generate variations of your ads according to seasons and locations. For example, you can use AI to change leaf colors during fall or add snow to the ground during winter.

Writing

AI has long been used to aid writing. If you use a smartphone, you're probably familiar with autocorrect and auto-completion, both powered by AI. Writing is an ideal application for AI because we do it a lot, it can be quite tedious, and we have a high tolerance for mistakes. If a model suggests something that you don't like, you can just ignore it.

12 Because enterprises usually spend a lot of money on ads and marketing, automation there can lead to huge savings. On average, 11% of a company's budget is spent on marketing. See "Marketing Budgets Vary by Industry" (*https://oreil.ly/D0-yA*) (Christine Moorman, *WSJ*, 2017).

It's not a surprise that LLMs are good at writing, given that they are trained for text completion. To study the impact of ChatGPT on writing, an MIT study (Noy and Zhang, 2023 (*https://oreil.ly/IzQ6F*)) assigned occupation-specific writing tasks to 453 college-educated professionals and randomly exposed half of them to ChatGPT. Their results show that among those exposed to ChatGPT, the average time taken decreased by 40% and output quality rose by 18%. ChatGPT helps close the gap in output quality between workers, which means that it's more helpful to those with less inclination for writing. Workers exposed to ChatGPT during the experiment were 2 times as likely to report using it in their real job two weeks after the experiment and 1.6 times as likely two months after that.

For consumers, the use cases are obvious. Many use AI to help them communicate better. You can be angry in an email and ask AI to make it pleasant. You can give it bullet points and get back complete paragraphs. Several people claimed they no longer send an important email without asking AI to improve it first.

Students are using AI to write essays. Writers are using AI to write books.[13] Many startups already use AI to generate children's, fan fiction, romance, and fantasy books. Unlike traditional books, AI-generated books can be interactive, as a book's plot can change depending on a reader's preference. This means that readers can actively participate in creating the story they are reading. A children's reading app identifies the words that a child has trouble with and generates stories centered around these words.

Note-taking and email apps like Google Docs, Notion, and Gmail all use AI to help users improve their writing. Grammarly (*https://arxiv.org/abs/2305.09857*), a writing assistant app, finetunes a model to make users' writing more fluent, coherent, and clear.

AI's ability to write can also be abused. In 2023, the New York Times (*https://oreil.ly/LB72P*) reported that Amazon was flooded with shoddy AI-generated travel guidebooks, each outfitted with an author bio, a website, and rave reviews, all AI-generated.

For enterprises, AI writing is common in sales, marketing, and general team communication. Many managers told me they've been using AI to help them write performance reports. AI can help craft effective cold outreach emails, ad copywriting, and product descriptions. Customer relationship management (CRM) apps like HubSpot and Salesforce also have tools for enterprise users to generate web content and outreach emails.

13 I have found AI very helpful in the process of writing this book, and I can see that AI will be able to automate many parts of the writing process. When writing fiction, I often ask AI to brainstorm ideas on what it thinks will happen next or how a character might react to a situation. I'm still evaluating what kind of writing can be automated and what kind of writing can't be.

AI seems particularly good with SEO, perhaps because many AI models are trained with data from the internet, which is populated with SEO-optimized text. AI is so good at SEO that it has enabled a new generation of content farms. These farms set up junk websites and fill them with AI-generated content to get them to rank high on Google to drive traffic to them. Then they sell advertising spots through ad exchanges. In June 2023, NewsGuard (*https://oreil.ly/mZKjr*) identified almost 400 ads from 141 popular brands on junk AI-generated websites. One of those junk websites produced 1,200 articles a day. Unless something is done to curtail this, the future of internet content will be AI-generated, and it'll be pretty bleak.[14]

Education

Whenever ChatGPT is down, OpenAI's Discord server is flooded with students complaining about being unable to complete their homework. Several education boards, including the New York City Public Schools and the Los Angeles Unified School District, were quick to ban ChatGPT (*https://oreil.ly/pqI5z*) for fear of students using it for cheating, but reversed their decisions (*https://oreil.ly/nxtzw*) just a few months later.

Instead of banning AI, schools could incorporate it to help students learn faster. AI can summarize textbooks and generate personalized lecture plans for each student. I find it strange that ads are personalized because we know everyone is different, but education is not. AI can help adapt the materials to the format best suited for each student. Auditory learners can ask AI to read the materials out loud. Students who love animals can use AI to adapt visualizations to feature more animals. Those who find it easier to read code than math equations can ask AI to translate math equations into code.

AI is especially helpful for language learning, as you can ask AI to roleplay different practice scenarios. Pajak and Bicknell (Duolingo, 2022) (*https://oreil.ly/C8kmI*) found that out of four stages of course creation, lesson personalization is the stage that can benefit the most from AI, as shown in Figure 1-10.

14 My hypothesis is that we'll become so distrustful of content on the internet that we'll only read content generated by people or brands we trust.

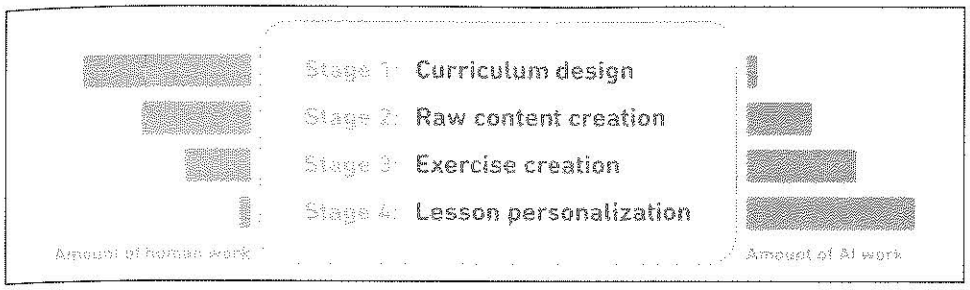


Figure 1-10. AI can be used throughout all four stages of course creation at Duolingo, but it's the most helpful in the personalization stage. Image from Pajak and Bicknell (Duolingo, 2022).

AI can generate quizzes, both multiple-choice and open-ended, and evaluate the answers. AI can become a debate partner as it's much better at presenting different views on the same topic than the average human. For example, Khan Academy (*https://oreil.ly/tC7-g*) offers AI-powered (*https://oreil.ly/_N1JR*) teaching assistants to students and course assistants to teachers. An innovative teaching method I've seen is that teachers assign AI-generated essays for students to find and correct mistakes.

While many education companies embrace AI to build better products, many find their lunches taken by AI. For example, Chegg, a company that helps students with their homework, saw its share price plummet from $28 when ChatGPT launched in November 2022 to $2 in September 2024, as students have been turning to AI for help (*https://oreil.ly/Y-hBW*).

If the risk is that AI can replace many skills, the opportunity is that AI can be used as a tutor to learn any skill. For many skills, AI can help someone get up to speed quickly and then continue learning on their own to become better than AI.

Conversational Bots

Conversational bots are versatile. They can help us find information, explain concepts, and brainstorm ideas. AI can be your companion and therapist. It can emulate personalities, letting you talk to a digital copy of anyone you like. Digital girlfriends and boyfriends have become weirdly popular in an incredibly short amount of time. Many are already spending more time talking to bots than to humans (see the discussions here (*https://oreil.ly/dZbym*) and here (*https://oreil.ly/svWj8*)). Some are worried that AI will ruin (*https://oreil.ly/SNme7*) dating (*https://oreil.ly/Jbt4R*).

In research, people have also found that they can use a group of conversational bots to simulate a society, enabling them to conduct studies on social dynamics (Park et al., 2023 (*https://arxiv.org/abs/2304.03442*)).

For enterprises, the most popular bots are customer support bots. They can help companies save costs while improving customer experience because they can respond to users sooner than human agents. AI can also be product copilots that guide customers through painful and confusing tasks such as filing insurance claims, doing taxes, or looking up corporate policies.

The success of ChatGPT prompted a wave of text-based conversational bots. However, text isn't the only interface for conversational agents. Voice assistants such as Google Assistant, Siri, and Alexa have been around for years.[15] 3D conversational bots are already common in games and gaining traction in retail and marketing.

One use case of AI-powered 3D characters is smart NPCs, non-player characters (see NVIDIA's demos of Inworld (*https://oreil.ly/yn-DN*) and Convai (*https://oreil.ly/zAHwz*)).[16] NPCs are essential for advancing the storyline of many games. Without AI, NPCs are typically scripted to do simple actions with a limited range of dialogues. AI can make these NPCs much smarter. Intelligent bots can change the dynamics of existing games like *The Sims* and *Skyrim* as well as enable new games never possible before.

Information Aggregation

Many people believe that our success depends on our ability to filter and digest useful information. However, keeping up with emails, Slack messages, and news can sometimes be overwhelming. Luckily, AI came to the rescue. AI has proven to be capable of aggregating information and summarizing it. According to Salesforce's 2023

15 It surprises me how long it takes Apple and Amazon to incorporate generative AI advances into Siri and Alexa. A friend thinks it's because these companies might have higher bars for quality and compliance, and it takes longer to develop voice interfaces than chat interfaces.

16 Disclaimer: I'm an advisor of Convai.

Generative AI Snapshot Research (*https://oreil.ly/74soT*), 74% of generative AI users use it to distill complex ideas and summarize information.

For consumers, many applications can process your documents—contracts, disclosures, papers—and let you retrieve information in a conversational manner. This use case is also called *talk-to-your-docs*. AI can help you summarize websites, research, and create reports on the topics of your choice. During the process of writing this book, I found AI helpful for summarizing and comparing papers.

Information aggregation and distillation are essential for enterprise operations. More efficient information aggregation and dissimilation can help an organization become leaner, as it reduces the burden on middle management. When Instacart (*https://oreil.ly/Qq5-g*) launched an internal prompt marketplace, it discovered that one of the most popular prompt templates is "Fast Breakdown". This template asks AI to summarize meeting notes, emails, and Slack conversations with facts, open questions, and action items. These action items can then be automatically inserted into a project tracking tool and assigned to the right owners.

AI can help you surface the critical information about your potential customers and run analyses on your competitors.

The more information you gather, the more important it is to organize it. Information aggregation goes hand in hand with data organization.

Data Organization

One thing certain about the future is that we'll continue producing more and more data. Smartphone users will continue taking photos and videos. Companies will continue to log everything about their products, employees, and customers. Billions of contracts are being created each year. Photos, videos, logs, and PDFs are all unstructured or semistructured data. It's essential to organize all this data in a way that can be searched later.

AI can help with exactly that. AI can automatically generate text descriptions about images and videos, or help match text queries with visuals that match those queries. Services like Google Photos are already using AI to surface images that match search queries.[17] Google Image Search goes a step further: if there's no existing image matching users' needs, it can generate some.

17 I currently have over 40,000 photos and videos in my Google Photos. Without AI, it'd be near impossible for me to search for the photos I want, when I want them.

AI is very good with data analysis. It can write programs to generate data visualization, identify outliers, and make predictions like revenue forecasts.[18]

Enterprises can use AI to extract structured information from unstructured data, which can be used to organize data and help search it. Simple use cases include automatically extracting information from credit cards, driver's licenses, receipts, tickets, contact information from email footers, and so on. More complex use cases include extracting data from contracts, reports, charts, and more. It's estimated that the IDP, intelligent data processing, industry will reach $12.81 billion by 2030 (*https://oreil.ly/vnDNK*), growing 32.9% each year.

Workflow Automation

Ultimately, AI should automate as much as possible. For end users, automation can help with boring daily tasks like booking restaurants, requesting refunds, planning trips, and filling out forms.

For enterprises, AI can automate repetitive tasks such as lead management, invoicing, reimbursements, managing customer requests, data entry, and so on. One especially exciting use case is using AI models to synthesize data, which can then be used to improve the models themselves. You can use AI to create labels for your data, looping in humans to improve the labels. We discuss data synthesis in Chapter 8.

Access to external tools is required to accomplish many tasks. To book a restaurant, an application might need permission to open a search engine to look up the restaurant's number, use your phone to make calls, and add appointments to your calendar. AIs that can plan and use tools are called *agents*. The level of interest around agents borders on obsession, but it's not entirely unwarranted. AI agents have the potential to make every person vastly more productive and generate vastly more economic value. Agents are a central topic in Chapter 6.

It's been a lot of fun looking into different AI applications. One of my favorite things to daydream about is the different applications I can build. However, not all applications should be built. The next section discusses what we should consider before building an AI application.

Planning AI Applications

Given the seemingly limitless potential of AI, it's tempting to jump into building applications. If you just want to learn and have fun, jump right in. Building is one of the best ways to learn. In the early days of foundation models, several heads of AI

18 Personally, I also find AI good at explaining data and graphs. When encountering a confusing graph with too much information, I ask ChatGPT to break it down for me.

told me that they encouraged their teams to experiment with AI applications to upskill themselves.

However, if you're doing this for a living, it might be worthwhile to take a step back and consider why you're building this and how you should go about it. It's easy to build a cool demo with foundation models. It's hard to create a profitable product.

Use Case Evaluation

The first question to ask is why you want to build this application. Like many business decisions, building an AI application is often a response to risks and opportunities. Here are a few examples of different levels of risks, ordered from high to low:

1. *If you don't do this, competitors with AI can make you obsolete.* If AI poses a major existential threat to your business, incorporating AI must have the highest priority. In the 2023 Gartner study (*https://oreil.ly/gqi3d*), 7% cited business continuity as their reason for embracing AI. This is more common for businesses involving document processing and information aggregation, such as financial analysis, insurance, and data processing. This is also common for creative work such as advertising, web design, and image production. You can refer to the 2023 OpenAI study, "GPTs are GPTs" (Eloundou et al., 2023 (*https://arxiv.org/abs/2303.10130*)), to see how industries rank in their exposure to AI.
2. *If you don't do this, you'll miss opportunities to boost profits and productivity.* Most companies embrace AI for the opportunities it brings. AI can help in most, if not all, business operations. AI can make user acquisition cheaper by crafting more effective copywrites, product descriptions, and promotional visual content. AI can increase user retention by improving customer support and customizing user experience. AI can also help with sales lead generation, internal communication, market research, and competitor tracking.
3. *You're unsure where AI will fit into your business yet, but you don't want to be left behind.* While a company shouldn't chase every hype train, many have failed by waiting too long to take the leap (cue Kodak, Blockbuster, and BlackBerry). Investing resources into understanding how a new, transformational technology can impact your business isn't a bad idea if you can afford it. At bigger companies, this can be part of the R&D department.[19]

Once you've found a good reason to develop this use case, you might consider whether you have to build it yourself. If AI poses an existential threat to your business, you might want to do AI in-house instead of outsourcing it to a competitor.

19 Smaller startups, however, might have to prioritize product focus and can't afford to have even one person to "look around."

However, if you're using AI to boost profits and productivity, you might have plenty of buy options that can save you time and money while giving you better performance.

The role of AI and humans in the application

What role AI plays in the AI product influences the application's development and its requirements. Apple (*https://oreil.ly/Dz1HE*) has a great document explaining different ways AI can be used in a product. Here are three key points relevant to the current discussion:

Critical or complementary
: If an app can still work without AI, AI is complementary to the app. For example, Face ID wouldn't work without AI-powered facial recognition, whereas Gmail would still work without Smart Compose.

  The more critical AI is to the application, the more accurate and reliable the AI part has to be. People are more accepting of mistakes when AI isn't core to the application.

Reactive or proactive
: A reactive feature shows its responses in reaction to users' requests or specific actions, whereas a proactive feature shows its responses when there's an opportunity for it. For example, a chatbot is reactive, whereas traffic alerts on Google Maps are proactive.

  Because reactive features are generated in response to events, they usually, but not always, need to happen fast. On the other hand, proactive features can be precomputed and shown opportunistically, so latency is less important.

  Because users don't ask for proactive features, they can view them as intrusive or annoying if the quality is low. Therefore, proactive predictions and generations typically have a higher quality bar.

Dynamic or static
: Dynamic features are updated continually with user feedback, whereas static features are updated periodically. For example, Face ID needs to be updated as people's faces change over time. However, object detection in Google Photos is likely updated only when Google Photos is upgraded.

  In the case of AI, dynamic features might mean that each user has their own model, continually finetuned on their data, or other mechanisms for personalization such as ChatGPT's memory feature, which allows ChatGPT to remember each user's preferences. However, static features might have one model for a group of users. If that's the case, these features are updated only when the shared model is updated.

It's also important to clarify the role of humans in the application. Will AI provide background support to humans, make decisions directly, or both? For example, for a customer support chatbot, AI responses can be used in different ways:

- AI shows several responses that human agents can reference to write faster responses.
- AI responds only to simple requests and routes more complex requests to humans.
- AI responds to all requests directly, without human involvement.

Involving humans in AI's decision-making processes is called *human-in-the-loop*.

Microsoft (2023) proposed a framework for gradually increasing AI automation in products that they call Crawl-Walk-Run (*https://oreil.ly/JW4_A*):

1. Crawl means human involvement is mandatory.
2. Walk means AI can directly interact with internal employees.
3. Run means increased automation, potentially including direct AI interactions with external users.

The role of humans can change over time as the quality of the AI system improves. For example, in the beginning, when you're still evaluating AI capabilities, you might use it to generate suggestions for human agents. If the acceptance rate by human agents is high, for example, 95% of AI-suggested responses to simple requests are used by human agents verbatim, you can let customers interact with AI directly for those simple requests.

AI product defensibility

If you're selling AI applications as standalone products, it's important to consider their defensibility. The low entry barrier is both a blessing and a curse. If something is easy for you to build, it's also easy for your competitors. What moats do you have to defend your product?

In a way, building applications on top of foundation models means providing a layer on top of these models.[20] This also means that if the underlying models expand in capabilities, the layer you provide might be subsumed by the models, rendering your application obsolete. Imagine building a PDF-parsing application on top of ChatGPT based on the assumption that ChatGPT can't parse PDFs well or can't do so at scale. Your ability to compete will weaken if this assumption is no longer true. However, even in this case, a PDF-parsing application might still make sense if it's built on top

20 A running joke in the early days of generative AI is that AI startups are OpenAI or Claude wrappers.

of open source models, gearing your solution toward users who want to host models in-house.

One general partner at a major VC firm told me that she's seen many startups whose entire products could be a feature for Google Docs or Microsoft Office. If their products take off, what would stop Google or Microsoft from allocating three engineers to replicate these products in two weeks?

In AI, there are generally three types of competitive advantages: technology, data, and distribution—the ability to bring your product in front of users. With foundation models, the core technologies of most companies will be similar. The distribution advantage likely belongs to big companies.

The data advantage is more nuanced. Big companies likely have more existing data. However, if a startup can get to market first and gather sufficient usage data to continually improve their products, data will be their moat. Even for the scenarios where user data can't be used to train models directly, usage information can give invaluable insights into user behaviors and product shortcomings, which can be used to guide the data collection and training process.[21]

There have been many successful companies whose original products could've been features of larger products. Calendly could've been a feature of Google Calendar. Mailchimp could've been a feature of Gmail. Photoroom could've been a feature of Google Photos.[22] Many startups eventually overtake bigger competitors, starting by building a feature that these bigger competitors overlooked. Perhaps yours can be the next one.

Setting Expectations

Once you've decided that you need to build this amazing AI application by yourself, the next step is to figure out what success looks like: how will you measure success? The most important metric is how this will impact your business. For example, if it's a customer support chatbot, the business metrics can include the following:

- What percentage of customer messages do you want the chatbot to automate?
- How many more messages should the chatbot allow you to process?
- How much quicker can you respond using the chatbot?
- How much human labor can the chatbot save you?

21 During the process of writing this book, I could hardly talk to any AI startup without hearing the phrase "data flywheel."

22 Disclaimer: I'm an investor in Photoroom.

A chatbot can answer more messages, but that doesn't mean it'll make users happy, so it's important to track customer satisfaction and customer feedback in general. "User Feedback" on page 474 discusses how to design a feedback system.

To ensure a product isn't put in front of customers before it's ready, have clear expectations on its usefulness threshold: how good it has to be for it to be useful. Usefulness thresholds might include the following metrics groups:

- Quality metrics to measure the quality of the chatbot's responses.
- Latency metrics including TTFT (time to first token), TPOT (time per output token), and total latency. What is considered acceptable latency depends on your use case. If all of your customer requests are currently being processed by humans with a median response time of an hour, anything faster than this might be good enough.
- Cost metrics: how much it costs per inference request.
- Other metrics such as interpretability and fairness.

If you're not yet sure what metrics you want to use, don't worry. The rest of the book will cover many of these metrics.

Milestone Planning

Once you've set measurable goals, you need a plan to achieve these goals. How to get to the goals depends on where you start. Evaluate existing models to understand their capabilities. The stronger the off-the-shelf models, the less work you'll have to do. For example, if your goal is to automate 60% of customer support tickets and the off-the-shelf model you want to use can already automate 30% of the tickets, the effort you need to put in might be less than if it can automate no tickets at all.

It's likely that your goals will change after evaluation. For example, after evaluation, you may realize that the resources needed to get the app to the usefulness threshold will be more than its potential return, and, therefore, you no longer want to pursue it.

Planning an AI product needs to account for its last mile challenge. Initial success with foundation models can be misleading. As the base capabilities of foundation models are already quite impressive, it might not take much time to build a fun demo. However, a good initial demo doesn't promise a good end product. It might take a weekend to build a demo but months, and even years, to build a product.

In the paper UltraChat, Ding et al. (2023) (*https://arxiv.org/abs/2305.14233*) shared that "the journey from 0 to 60 is easy, whereas progressing from 60 to 100 becomes exceedingly challenging." LinkedIn (2024) (*https://www.linkedin.com/blog/engineering/generative-ai/musings-on-building-a-generative-ai-product*) shared the same sentiment. It took them one month to achieve 80% of the experience they wanted. This

initial success made them grossly underestimate how much time it'd take them to improve the product. They found it took them four more months to finally surpass 95%. A lot of time was spent working on the product kinks and dealing with hallucinations. The slow speed of achieving each subsequent 1% gain was discouraging.

Maintenance

Product planning doesn't stop at achieving its goals. You need to think about how this product might change over time and how it should be maintained. Maintenance of an AI product has the added challenge of AI's fast pace of change. The AI space has been moving incredibly fast in the last decade. It'll probably continue moving fast for the next decade. Building on top of foundation models today means committing to riding this bullet train.

Many changes are good. For example, the limitations of many models are being addressed. Context lengths are getting longer. Model outputs are getting better. Model *inference*, the process of computing an output given an input, is getting faster and cheaper. Figure 1-11 shows the evolution of inference cost and model performance on Massive Multitask Language Understanding (MMLU) (Hendrycks et al., 2020 (*https://arxiv.org/abs/2009.03300*)), a popular foundation model benchmark, between 2022 and 2024.

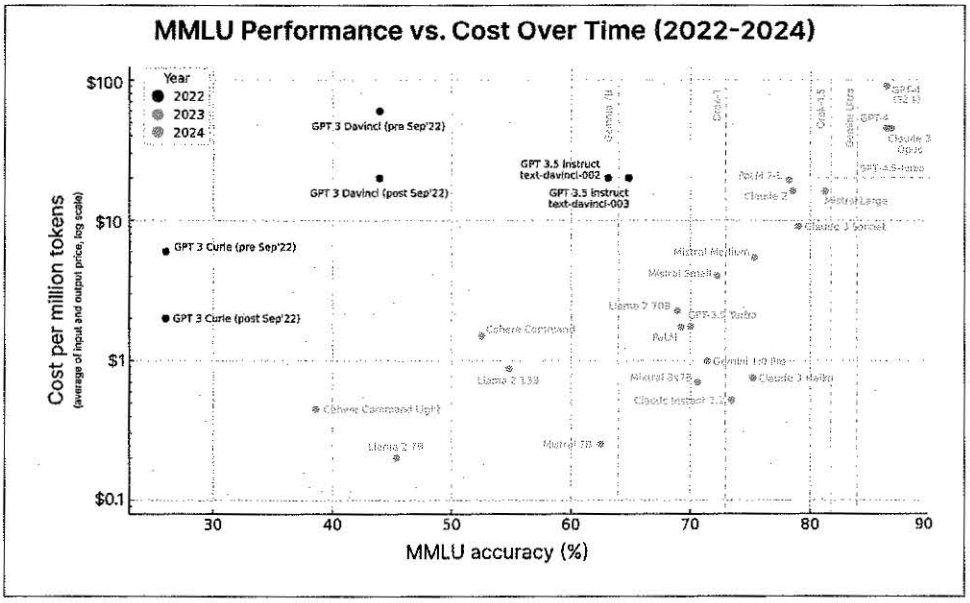
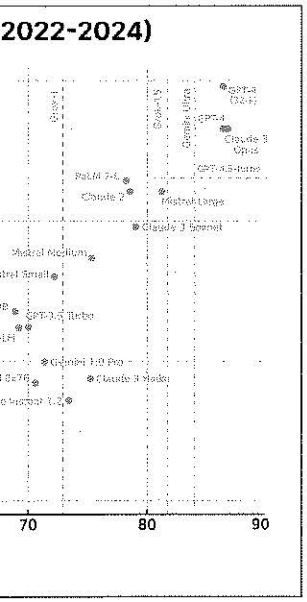


Figure 1-11. The cost of AI reasoning rapidly drops over time. Image from Katrina Nguyen (https://oreil.ly/UyL8r) (2024).

uch time it'd take them to
e months to finally surpass
ts and dealing with halluci-
gain was discouraging.

need to think about how
e maintained. Maintenance
ce of change. The AI space
bably continue moving fast
s today means committing

of many models are being
utputs are getting better.
n an input, is getting faster
ce cost and model perfor-
MMLU) (Hendrycks et al.,
dation model benchmark,



e. Image from Katrina

However, even these good changes can cause friction in your workflows. You'll have to constantly be on your guard and run a cost-benefit analysis of each technology investment. The best option today might turn into the worst option tomorrow. You may decide to build a model in-house because it seems cheaper than paying for model providers, only to find out after three months that model providers have dropped their prices in half, making in-house the expensive option. You might invest in a third-party solution and tailor your infrastructure around it, only for the provider to go out of business after failing to secure funding.

Some changes are easier to adapt to. For example, as model providers converge to the same API, it's becoming easier to swap one model API for another. However, as each model has its quirks, strengths, and weaknesses, developers working with the new model will need to adjust their workflows, prompts, and data to this new model. Without proper infrastructure for versioning and evaluation in place, the process can cause a lot of headaches.

Some changes are harder to adapt to, especially those around regulations. Technologies surrounding AI are considered national security issues for many countries, meaning resources for AI, including compute, talent, and data, are heavily regulated. The introduction of Europe's General Data Protection Regulation (GDPR), for example, was estimated to cost businesses $9 billion (*https://oreil.ly/eDfB8*) to become compliant. Compute availability can change overnight as new laws put more restrictions on who can buy and sell compute resources (see the US October 2023 Executive Order (*https://oreil.ly/eYTmr*)). If your GPU vendor is suddenly banned from selling GPUs to your country, you're in trouble.

Some changes can even be fatal. For example, regulations around intellectual property (IP) and AI usage are still evolving. If you build your product on top of a model trained using other people's data, can you be certain that your product's IP will always belong to you? Many IP-heavy companies I've talked to, such as game studios, hesitate to use AI for fear of losing their IPs later on.

Once you've committed to building an AI product, let's look into the engineering stack needed to build these applications.

The AI Engineering Stack

AI engineering's rapid growth also induced an incredible amount of hype and FOMO (fear of missing out). The number of new tools, techniques, models, and applications introduced every day can be overwhelming. Instead of trying to keep up with the constantly shifting sand, let's look into the fundamental building blocks of AI engineering.

To understand AI engineering, it's important to recognize that AI engineering evolved out of ML engineering. When a company starts experimenting with foundation models, it's natural that its existing ML team should lead the effort. Some companies treat AI engineering the same as ML engineering, as shown in Figure 1-12.

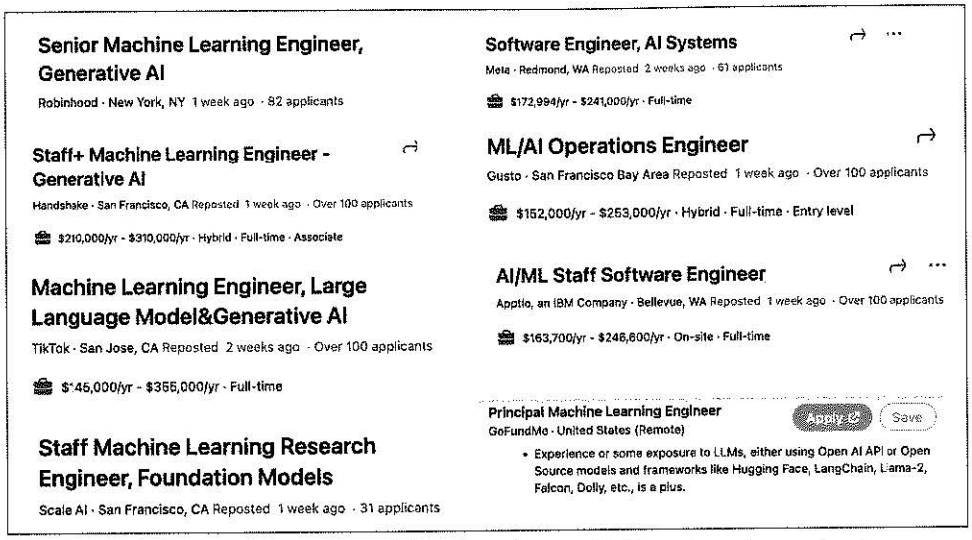


Figure 1-12. Many companies put AI engineering and ML engineering under the same umbrella, as shown in the job headlines on LinkedIn from December 17, 2023.

Some companies have separate job descriptions for AI engineering, as shown in Figure 1-13.

Regardless of where organizations position AI engineers and ML engineers, their roles have significant overlap. Existing ML engineers can add AI engineering to their lists of skills to expand their job prospects. However, there are also AI engineers with no previous ML experience.

To best understand AI engineering and how it differs from traditional ML engineering, the following section breaks down different layers of the AI application building process and looks at the role each layer plays in AI engineering and ML engineering.

nize that AI engineering perimenting with founda- ead the effort. Some com- shown in Figure 1-12.

gineering under the same ecember 17, 2023.

engineering, as shown in

and ML engineers, their dd AI engineering to their are also AI engineers with

traditional ML engineer- he AI application building ing and ML engineering.

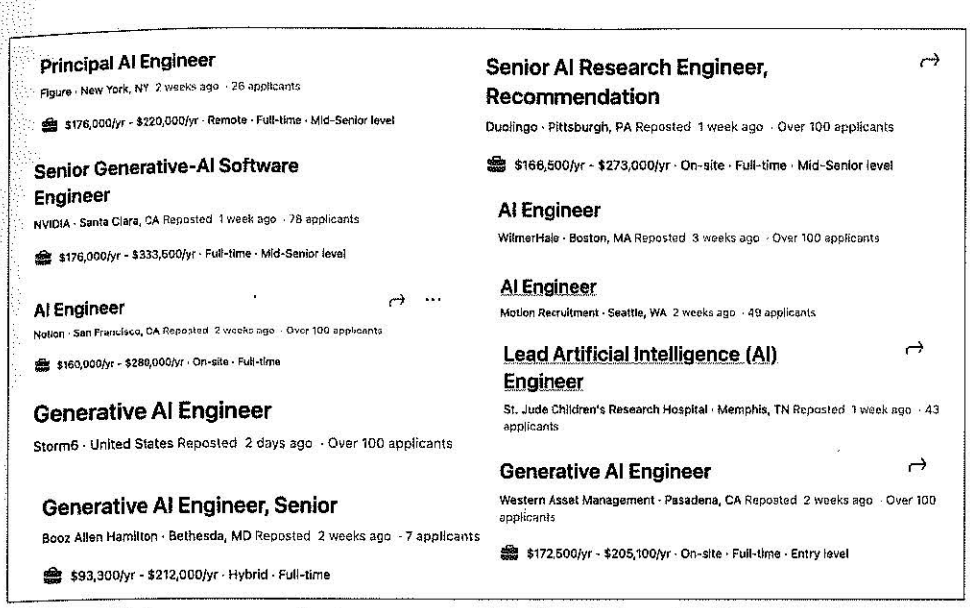


Figure 1-13. Some companies have separate job descriptions for AI engineering, as shown in the job headlines on LinkedIn from December 17, 2023.

Three Layers of the AI Stack

There are three layers to any AI application stack: application development, model development, and infrastructure. When developing an AI application, you'll likely start from the top layer and move down as needed:

Application development
: With models readily available, anyone can use them to develop applications. This is the layer that has seen the most action in the last two years, and it is still rapidly evolving. Application development involves providing a model with good prompts and necessary context. This layer requires rigorous evaluation. Good applications also demand good interfaces.

Model development
: This layer provides tooling for developing models, including frameworks for modeling, training, finetuning, and inference optimization. Because data is central to model development, this layer also contains dataset engineering. Model development also requires rigorous evaluation.

Infrastructure
: At the bottom is the stack is infrastructure, which includes tooling for model serving, managing data and compute, and monitoring.

These three layers and examples of responsibilities for each layer are shown in Figure 1-14.

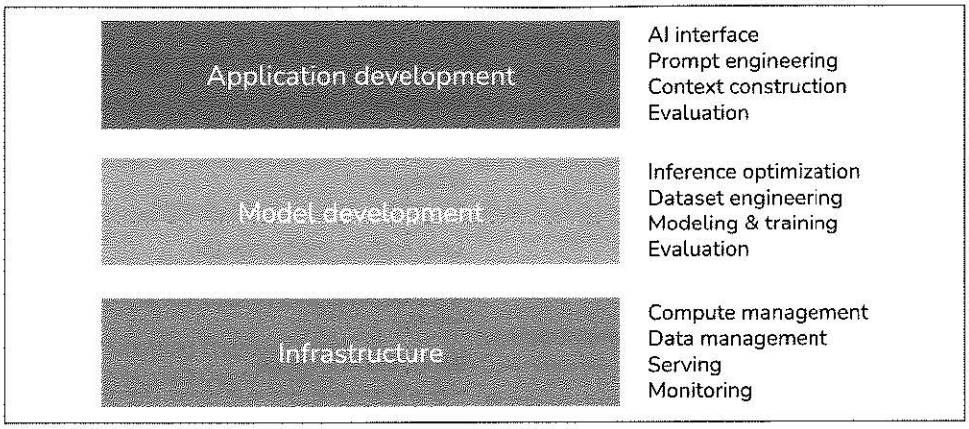


Figure 1-14. Three layers of the AI engineering stack.

To get a sense of how the landscape has evolved with foundation models, in March 2024, I searched GitHub for all AI-related repositories with at least 500 stars. Given the prevalence of GitHub, I believe this data is a good proxy for understanding the ecosystem. In my analysis, I also included repositories for applications and models, which are the products of the application development and model development layers, respectively. I found a total of 920 repositories. Figure 1-15 shows the cumulative number of repositories in each category month-over-month.

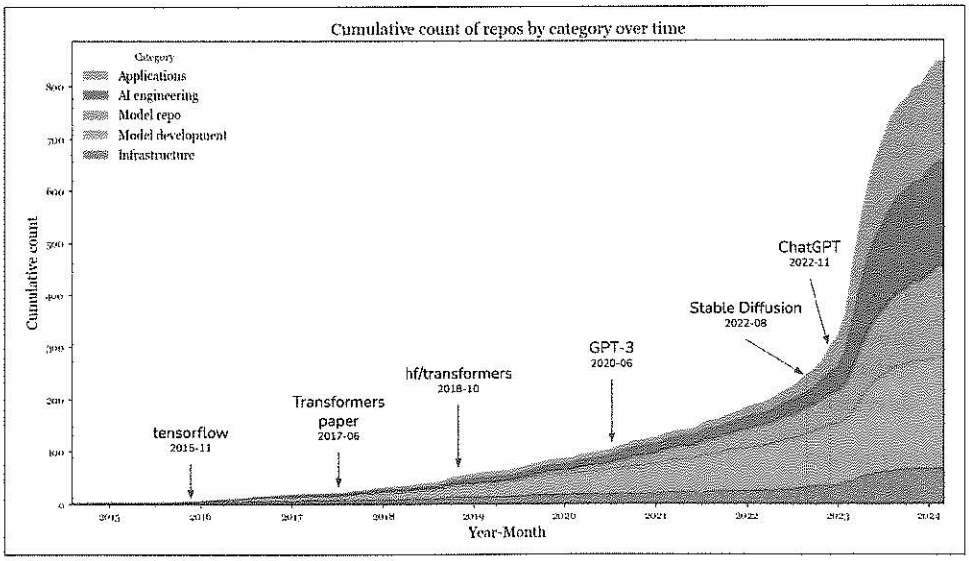
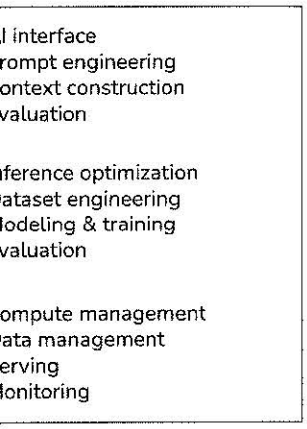
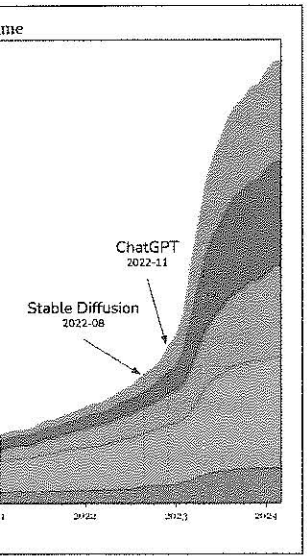


Figure 1-15. Cumulative count of repositories by category over time.

each layer are shown in



ndation models, in March
th at least 500 stars. Given
oxy for understanding the
r applications and models,
d model development lay-
1-15 shows the cumulative
h.



er time.

The data shows a big jump in the number of AI toolings in 2023, after the introduction of Stable Diffusion and ChatGPT. In 2023, the categories that saw the highest increases were applications and application development. The infrastructure layer saw some growth, but it was much less than the growth seen in other layers. This is expected. Even though models and applications have changed, the core infrastructural needs—resource management, serving, monitoring, etc.—remain the same.

This brings us to the next point. While the level of excitement and creativity around foundation models is unprecedented, many principles of building AI applications remain the same. For enterprise use cases, AI applications still need to solve business problems, and, therefore, it's still essential to map from business metrics to ML metrics and vice versa. You still need to do systematic experimentation. With classical ML engineering, you experiment with different hyperparameters. With foundation models, you experiment with different models, prompts, retrieval algorithms, sampling variables, and more. (Sampling variables are discussed in Chapter 2.) We still want to make models run faster and cheaper. It's still important to set up a feedback loop so that we can iteratively improve our applications with production data.

This means that much of what ML engineers have learned and shared over the last decade is still applicable. This collective experience makes it easier for everyone to begin building AI applications. However, built on top of these enduring principles are many innovations unique to AI engineering, which we'll explore in this book.

AI Engineering Versus ML Engineering

While the unchanging principles of deploying AI applications are reassuring, it's also important to understand how things have changed. This is helpful for teams that want to adapt their existing platforms for new AI use cases and developers who are interested in which skills to learn to stay competitive in a new market.

At a high level, building applications using foundation models today differs from traditional ML engineering in three major ways:

1. Without foundation models, you have to train your own models for your applications. With AI engineering, you use a model someone else has trained for you. This means that AI engineering focuses less on modeling and training, and more on model adaptation.
2. AI engineering works with models that are bigger, consume more compute resources, and incur higher latency than traditional ML engineering. This means that there's more pressure for efficient training and inference optimization. A corollary of compute-intensive models is that many companies now need more GPUs and work with bigger compute clusters than they previously did, which

means there's more need for engineers who know how to work with GPUs and big clusters.[23]

3. AI engineering works with models that can produce open-ended outputs. Open-ended outputs give models the flexibility to be used for more tasks, but they are also harder to evaluate. This makes evaluation a much bigger problem in AI engineering.

In short, AI engineering differs from ML engineering in that it's less about model development and more about adapting and evaluating models. I've mentioned model adaptation several times in this chapter, so before we move on, I want to make sure that we're on the same page about what model adaptation means. In general, model adaptation techniques can be divided into two categories, depending on whether they require updating model weights.

Prompt-based techniques, which include prompt engineering, adapt a model without updating the model weights. You adapt a model by giving it instructions and context instead of changing the model itself. Prompt engineering is easier to get started and requires less data. Many successful applications have been built with just prompt engineering. Its ease of use allows you to experiment with more models, which increases your chance of finding a model that is unexpectedly good for your applications. However, prompt engineering might not be enough for complex tasks or applications with strict performance requirements.

Finetuning, on the other hand, requires updating model weights. You adapt a model by making changes to the model itself. In general, finetuning techniques are more complicated and require more data, but they can improve your model's quality, latency, and cost significantly. Many things aren't possible without changing model weights, such as adapting the model to a new task it wasn't exposed to during training.

Now, let's zoom into the application development and model development layers to see how each has changed with AI engineering, starting with what existing ML engineers are more familiar with. This section gives an overview of different processes involved in developing an AI application. How these processes work will be discussed throughout this book.

Model development

Model development is the layer most commonly associated with traditional ML engineering. It has three main responsibilities: modeling and training, dataset engineering, and inference optimization. Evaluation is also required, but because most people

23 As the head of AI at a Fortune 500 company told me: his team knows how to work with 10 GPUs, but they don't know how to work with 1,000 GPUs.

will come across it first in the application development layer, I'll discuss evaluation in the next section.

Modeling and training. *Modeling and training* refers to the process of coming up with a model architecture, training it, and finetuning it. Examples of tools in this category are Google's TensorFlow, Hugging Face's Transformers, and Meta's PyTorch.

Developing ML models requires specialized ML knowledge. It requires knowing different types of ML algorithms (such as clustering, logistic regression, decision trees, and collaborative filtering) and neural network architectures (such as feedforward, recurrent, convolutional, and transformer). It also requires understanding how a model learns, including concepts such as gradient descent, loss function, regularization, etc.

With the availability of foundation models, ML knowledge is no longer a must-have for building AI applications. I've met many wonderful and successful AI application builders who aren't at all interested in learning about gradient descent. However, ML knowledge is still extremely valuable, as it expands the set of tools that you can use and helps troubleshooting when a model doesn't work as expected.

On the Differences Among Training, Pre-Training, Finetuning, and Post-Training

Training always involves changing model weights, but not all changes to model weights constitute training. For example, quantization, the process of reducing the precision of model weights, technically changes the model's weight values but isn't considered training.

The term training can often be used in place of pre-training, finetuning, and post-training, which refer to different training phases:

Pre-training

Pre-training refers to training a model from scratch—the model weights are randomly initialized. For LLMs, pre-training often involves training a model for text completion. Out of all training steps, pre-training is often the most resource-intensive by a long shot. For the InstructGPT model, pre-training takes up to 98% of the overall compute and data resources (*https://oreil.ly/G3LUh*). Pre-training also takes a long time to do. A small mistake during pre-training can incur a significant financial loss and set back the project significantly. Due to the resource-intensive nature of pre-training, this has become an art that only a few practice. Those with expertise in pre-training large models, however, are heavily sought after.[24]

24 And they are offered incredible compensation packages (*https://oreil.ly/AhANP*).

Finetuning

Finetuning means continuing to train a previously trained model—the model weights are obtained from the previous training process. Because the model already has certain knowledge from pre-training, finetuning typically requires fewer resources (e.g., data and compute) than pre-training.

Post-training

Many people use *post-training* to refer to the process of training a model after the pre-training phase. Conceptually, post-training and finetuning are the same and can be used interchangeably. However, sometimes, people might use them differently to signify the different goals. It's usually post-training when it's done by model developers. For example, OpenAI might post-train a model to make it better at following instructions before releasing it. It's finetuning when it's done by application developers. For example, you might finetune an OpenAI model (which might have been post-trained itself) to adapt it to your needs.

Pre-training and post-training make up a spectrum.[25] Their processes and toolings are very similar. Their differences are explored further in Chapters 2 and 7.

Some people use the term training to refer to prompt engineering, which isn't correct. I read a *Business Insider* article (*https://oreil.ly/0VqmX*) where the author said she trained ChatGPT to mimic her younger self. She did so by feeding her childhood journal entries into ChatGPT. Colloquially, the author's usage of the word *training* is correct, as she's teaching the model to do something. But technically, if you teach a model what to do via the context input into the model, you're doing prompt engineering. Similarly, I've seen people using the term *finetuning* when what they do is prompt engineering.

Dataset engineering. *Dataset engineering* refers to curating, generating, and annotating the data needed for training and adapting AI models.

In traditional ML engineering, most use cases are close-ended—a model's output can only be among predefined values. For example, spam classification with only two possible outputs, "spam" and "not spam", is close-ended. Foundation models, however, are open-ended. Annotating open-ended queries is much harder than annotating close-ended queries—it's easier to determine whether an email is spam than to write an essay. So data annotation is a much bigger challenge for AI engineering.

25 If you find the terms "pre-training" and "post-training" lacking in imagination, you're not alone. The AI research community is great at many things, but naming isn't one of them. We already talked about how "large language models" is hardly a scientific term because of the ambiguity of the word "large". And I really wish people would stop publishing papers with the title "X is all you need."

Another difference is that traditional ML engineering works more with tabular data, whereas foundation models work with unstructured data. In AI engineering, data manipulation is more about deduplication, tokenization, context retrieval, and quality control, including removing sensitive information and toxic data. Dataset engineering is the focus of Chapter 8.

Many people argue that because models are now commodities, data will be the main differentiator, making dataset engineering more important than ever. How much data you need depends on the adapter technique you use. Training a model from scratch generally requires more data than finetuning, which, in turn, requires more data than prompt engineering.

Regardless of how much data you need, expertise in data is useful when examining a model, as its training data gives important clues about that model's strengths and weaknesses.

Inference optimization. *Inference optimization* means making models faster and cheaper. Inference optimization has always been important for ML engineering. Users never say no to faster models, and companies can always benefit from cheaper inference. However, as foundation models scale up to incur even higher inference cost and latency, inference optimization has become even more important.

One challenge with foundation models is that they are often *autoregressive*—tokens are generated sequentially. If it takes 10 ms for a model to generate a token, it'll take a second to generate an output of 100 tokens, and even more for longer outputs. As users are getting notoriously impatient, getting AI applications' latency down to the 100 ms latency (*https://oreil.ly/gGXZ-*) expected for a typical internet application is a huge challenge. Inference optimization has become an active subfield in both industry and academia.

A summary of how the importance of different categories of model development change with AI engineering is shown in Table 1-4.

Table 1-4. How different responsibilities of model development have changed with foundation models.

Category	Building with traditional ML	Building with foundation models
Modeling and training	ML knowledge is required for training a model from scratch	ML knowledge is a nice-to-have, not a must-have[a]
Dataset engineering	More about feature engineering, especially with tabular data	Less about feature engineering and more about data deduplication, tokenization, context retrieval, and quality control
Inference optimization	Important	Even more important

[a] Many people would dispute this claim, saying that ML knowledge is a must-have.

Inference optimization techniques, including quantization, distillation, and parallelism, are discussed in Chapters 7 through 9.

Application development

With traditional ML engineering, where teams build applications using their proprietary models, the model quality is a differentiation. With foundation models, where many teams use the same model, differentiation must be gained through the application development process.

The application development layer consists of these responsibilities: evaluation, prompt engineering, and AI interface.

Evaluation. *Evaluation* is about mitigating risks and uncovering opportunities. Evaluation is necessary throughout the whole model adaptation process. Evaluation is needed to select models, to benchmark progress, to determine whether an application is ready for deployment, and to detect issues and opportunities for improvement in production.

While evaluation has always been important in ML engineering, it's even more important with foundation models, for many reasons. The challenges of evaluating foundation models are discussed in Chapter 3. To summarize, these challenges chiefly arise from foundation models' open-ended nature and expanded capabilities. For example, in close-ended ML tasks like fraud detection, there are usually expected ground truths that you can compare your model's outputs against. If a model's output differs from the expected output, you know the model is wrong. For a task like chatbots, however, there are so many possible responses to each prompt that it is impossible to curate an exhaustive list of ground truths to compare a model's response to.

The existence of so many adaptation techniques also makes evaluation harder. A system that performs poorly with one technique might perform much better with another. When Google launched Gemini in December 2023, they claimed that Gemini is better than ChatGPT in the MMLU benchmark (Hendrycks et al., 2020 (*https://arxiv.org/abs/2009.03300*)). Google had evaluated Gemini using a prompt engineering technique called CoT@32 (*https://oreil.ly/VDwaR*). In this technique, Gemini was shown 32 examples, while ChatGPT was shown only 5 examples. When both were shown five examples, ChatGPT performed better, as shown in Table 1-5.

Table 1-5. Different prompts can cause models to perform very differently, as seen in Gemini's technical report (December 2023).

	Gemini Ultra	Gemini Pro	GPT-4	GPT-3.5	PaLM 2-L	Claude 2	Inflection-2	Grok 1	Llama-2
MMLU performance	90.04% CoT@32	79.13% CoT@8	87.29% CoT@32 (via API)	70% 5-shot	78.4% 5-shot	78.5% 5-shot CoT	79.6% 5-shot	73.0% 5-shot	68.0%
	83.7% 5-shot	71.8% 5-shot	86.4% 5-shot (reported)						

Prompt engineering and context construction. *Prompt engineering* is about getting AI models to express the desirable behaviors from the input alone, without changing the model weights. The Gemini evaluation story highlights the impact of prompt engineering on model performance. By using a different prompt engineering technique, Gemini Ultra's performance on MMLU went from 83.7% to 90.04%.

It's possible to get a model to do amazing things with just prompts. The right instructions can get a model to perform the task you want, in the format of your choice. Prompt engineering is not just about telling a model what to do. It's also about giving the model the necessary context and tools to do a given task. For complex tasks with long context, you might also need to provide the model with a memory management system so that the model can keep track of its history. Chapter 5 discusses prompt engineering, and Chapter 6 discusses context construction.

AI interface. *AI interface* means creating an interface for end users to interact with your AI applications. Before foundation models, only organizations with sufficient resources to develop AI models could develop AI applications. These applications were often embedded into the organizations' existing products. For example, fraud detection was embedded into Stripe, Venmo, and PayPal. Recommender systems were part of social networks and media apps like Netflix, TikTok, and Spotify.

With foundation models, anyone can build AI applications. You can serve your AI applications as standalone products or embed them into other products, including products developed by other people. For example, ChatGPT and Perplexity are standalone products, whereas GitHub's Copilot is commonly used as a plug-in in VSCode, and Grammarly is commonly used as a browser extension for Google Docs. Midjourney can either be used via its standalone web app or via its integration in Discord.

There need to be tools that provide interfaces for standalone AI applications or make it easy to integrate AI into existing products. Here are just some of the interfaces that are gaining popularity for AI applications:

- Standalone web, desktop, and mobile apps.[26]
- Browser extensions that let users quickly query AI models while browsing.
- Chatbots integrated into chat apps like Slack, Discord, WeChat, and WhatsApp.
- Many products, including VSCode, Shopify, and Microsoft 365, provide APIs that let developers integrate AI into their products as plug-ins and add-ons. These APIs can also be used by AI agents to interact with the world, as discussed in Chapter 6.

While the chat interface is the most commonly used, AI interfaces can also be voice-based (such as with voice assistants) or embodied (such as in augmented and virtual reality).

These new AI interfaces also mean new ways to collect and extract user feedback. The conversation interface makes it so much easier for users to give feedback in natural language, but this feedback is harder to extract. User feedback design is discussed in Chapter 10.

A summary of how the importance of different categories of app development changes with AI engineering is shown in Table 1-6.

Table 1-6. The importance of different categories in app development for AI engineering and ML engineering.

Category	Building with traditional ML	Building with foundation models
AI interface	Less important	Important
Prompt engineering	Not applicable	Important
Evaluation	Important	More important

AI Engineering Versus Full-Stack Engineering

The increased emphasis on application development, especially on interfaces, brings AI engineering closer to full-stack development.[27] The rising importance of interfaces leads to a shift in the design of AI toolings to attract more frontend engineers. Traditionally, ML engineering is Python-centric. Before foundation models, the most popular ML frameworks supported mostly Python APIs. Today, Python is still popu-

26 Streamlit, Gradio, and Plotly Dash are common tools for building AI web apps.

27 Anton Bacaj told me that "AI engineering is just software engineering with AI models thrown in the stack."

lar, but there is also increasing support for JavaScript APIs, with LangChain.js (*https://github.com/langchain-ai/langchainjs*), Transformers.js (*https://github.com/huggingface/transformers.js*), OpenAI's Node library (*https://github.com/openai/openai-node*), and Vercel's AI SDK (*https://github.com/vercel/ai*).

While many AI engineers come from traditional ML backgrounds, more are increasingly coming from web development or full-stack backgrounds. An advantage that full-stack engineers have over traditional ML engineers is their ability to quickly turn ideas into demos, get feedback, and iterate.

With traditional ML engineering, you usually start with gathering data and training a model. Building the product comes last. However, with AI models readily available today, it's possible to start with building the product first, and only invest in data and models once the product shows promise, as visualized in Figure 1-16.

ML Engineering:	Data	→	Model	→ Product
AI Engineering:	Product	→	Data	→ Model

Figure 1-16. The new AI engineering workflow rewards those who can iterate fast. Image recreated from "The Rise of the AI Engineer" (Shawn Wang, 2023 (https://oreil.ly/OOZK-)).

In traditional ML engineering, model development and product development are often disjointed processes, with ML engineers rarely involved in product decisions at many organizations. However, with foundation models, AI engineers tend to be much more involved in building the product.

Summary

I meant this chapter to serve two purposes. One is to explain the emergence of AI engineering as a discipline, thanks to the availability of foundation models. Two is to give an overview of the process needed to build applications on top of these models. I hope that this chapter achieved this goal. As an overview chapter, it only lightly touched on many concepts. These concepts will be explored further in the rest of the book.

The chapter discussed the rapid evolution of AI in recent years. It walked through some of the most notable transformations, starting with the transition from language models to large language models, thanks to a training approach called self-supervision. It then traced how language models incorporated other data modalities to become foundation models, and how foundation models gave rise to AI engineering.

The rapid growth of AI engineering is motivated by the many applications enabled by the emerging capabilities of foundation models. This chapter discussed some of the most successful application patterns, both for consumers and enterprises. Despite the incredible number of AI applications already in production, we're still in the early stages of AI engineering, with countless more innovations yet to be built.

Before building an application, an important yet often overlooked question is whether you should build it. This chapter discussed this question together with major considerations for building AI applications.

While AI engineering is a new term, it evolved out of ML engineering, which is the overarching discipline involved with building applications with all ML models. Many principles from ML engineering are still applicable to AI engineering. However, AI engineering also brings with it new challenges and solutions. The last section of the chapter discusses the AI engineering stack, including how it has changed from ML engineering.

One aspect of AI engineering that is especially challenging to capture in writing is the incredible amount of collective energy, creativity, and engineering talent that the community brings. This collective enthusiasm can often be overwhelming, as it's impossible to keep up-to-date with new techniques, discoveries, and engineering feats that seem to happen constantly.

One consolation is that since AI is great at information aggregation, it can help us aggregate and summarize all these new updates. But tools can help only to a certain extent. The more overwhelming a space is, the more important it is to have a framework to help us navigate it. This book aims to provide such a framework.

The rest of the book will explore this framework step-by-step, starting with the fundamental building block of AI engineering: the foundation models that make so many amazing applications possible.