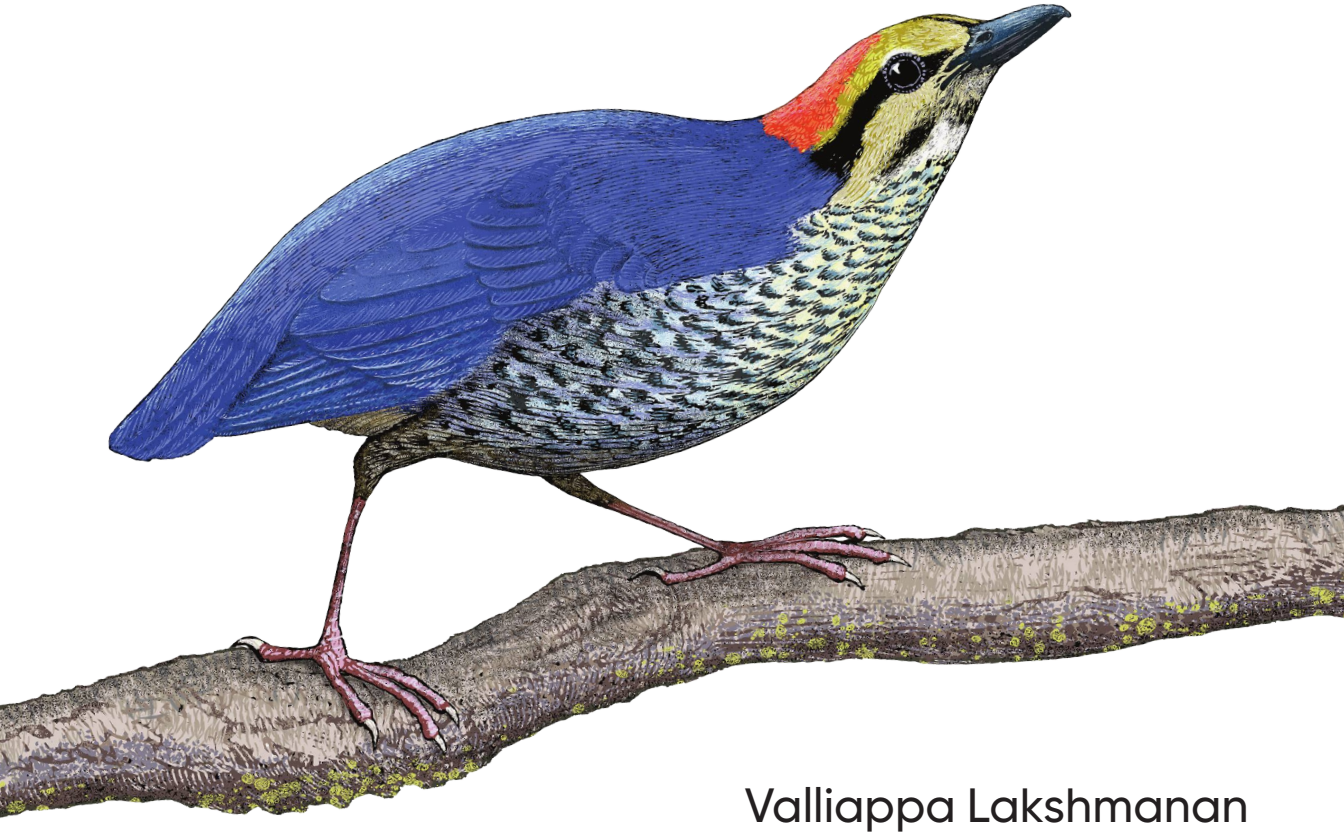
O'REILLY®

Generative AI Design Patterns

Solutions to Common Challenges When Building GenAI Agents and Applications



Valliappa Lakshmanan
& Hannes Hapke

"A clear, practical playbook for moving from GenAI prototypes to production. The design patterns capture real-world engineering experience, helping practitioners navigate hallucinations, reliability, and scaling challenges with confidence."

Dr. Marily Nika, author of *Building AI-Powered Products*

"Hannes and Lak's hard-earned wisdom makes this the comprehensive guide to bridge theory and real-world implementation."

Chip Huyen, author of *Designing Machine Learning Systems* and *AI Engineering*

Generative AI Design Patterns

Generative AI enables powerful new capabilities, but they come with some serious limitations to tackle before shipping a reliable application or agent. Luckily, experts in the field have compiled a library of 32 tried-and-true design patterns to address the challenges you're likely to encounter when building applications using LLMs, such as hallucinations, nondeterministic responses, and knowledge cutoffs.

This book codifies research and real-world experience into advice you can incorporate into your projects. Each pattern describes a problem, shows a proven way to solve it with a fully coded example, and discusses the trade-offs.

- **Design around the limitations of LLMs**
- **Ensure that generated content follows a specific style, tone, or format**
- **Maximize creativity while balancing different types of risk**
- **Build agents that plan, self-correct, take action, and collaborate with other agents**
- **Compose patterns into agentic applications for a variety of use cases**

Valliappa (Lak) Lakshmanan is cofounder and CTO of Obin.ai, an agentic AI startup. Previously, he was director of AI solutions at Google and an ML researcher at NOAA. He has authored several O'Reilly books, including *Machine Learning Design Patterns*, and was elected an American Meteorological Society Fellow for pioneering machine learning in severe weather prediction.

Hannes Hapke is principal machine learning engineer at Digits, where he built the ML systems for financial applications. He is a Google Developer Expert in machine learning and serves on Google's Developer Advisory Board. He has also coauthored multiple machine learning books, including *Building Machine Learning Pipelines* (O'Reilly) and *Natural Language Processing in Action* (Manning).

DATA

US $79.99 CAN $99.99

ISBN: 979-8-341-62266-1

O'REILLY®

Generative AI Design Patterns

Solutions to Common Challenges When Building GenAI Agents and Applications

Valliappa Lakshmanan and Hannes Hapke

O'REILLY®

Generative AI Design Patterns
by Valliappa Lakshmanan and Hannes Hapke

Printed in the United States of America.

Published by O'Reilly Media, Inc., 141 Stony Circle, Suite 195, Santa Rosa, CA 95401.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Nicole Butterfield
Development Editor: Sarah Grey
Production Editor: Christopher Faucher
Copyeditor: Doug McNair
Proofreader: Emily Wydeven
Indexer: Sue Klefstad
Cover Designer: Susan Thompson
Cover Illustrator: Susan Brown
Interior Designer: David Futato
Interior Illustrator: Kate Dullea

October 2025: First Edition

Revision History for the First Edition
2025-10-03: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9798341622661* for release details.

979-8-341-62266-1

[LSI]

Table of Contents

Preface

If you're an AI engineer building generative AI (GenAI) applications, you've likely experienced the frustrating gap between the ease of creating impressive prototypes and the complexity of deploying them reliably in production. While foundational models make it easy to build compelling demos, production systems demand solutions to fundamental challenges: hallucinations that compromise accuracy, inconsistent outputs that break downstream processes, knowledge gaps that limit enterprise applicability, and reliability issues that make systems unsuitable for critical applications.

This book bridges that gap by providing 32 battle-tested design patterns that address the recurring problems you'll encounter when building production-grade GenAI applications. These patterns aren't theoretical constructs—they codify proven solutions that are often derived from cutting-edge research and refined by practitioners who have successfully deployed GenAI systems at scale.

Supervised machine learning (ML) involves training a problem-specific model on a large training dataset of example inputs and outputs—but GenAI applications rarely include a training phase. Instead, they commonly use general-purpose foundational models. This book is focused on design patterns for AI applications that are built on top of foundational models, such as Open AI's GPT, Anthropic's Claude, Google's Gemini, or Meta's Llama.

In this book, we cover the entire AI engineering workflow. After an introduction in Chapter 1, Chapter 2 provides practical patterns for controlling content style and format (including Logits Masking [Pattern 1] and Grammar [Pattern 2]). Chapter 3 and Chapter 4 cover integrating external knowledge through sophisticated retrieval-augmented generation (RAG) implementations, from Basic RAG (Pattern 6) to Deep Search (Pattern 12). Chapter 5 is about enhancing your model's reasoning capabilities with patterns like Chain of Thought (Pattern 13), Tree of Thoughts (Pattern 14), and Adapter Tuning (Pattern 15). Chapter 6 emphasizes building reliable systems with LLM-as-Judge (Pattern 17), Reflection (Pattern 18), and Prompt Opti-

mization (Pattern 20) patterns. Chapter 7 is about creating agentic systems, including Tool Calling (Pattern 21) and Multiagent Collaboration (Pattern 23). Chapter 8 covers optimizing deployment (including Small Language Model [Pattern 24] and Inference Distribution Testing [Pattern 27]), and Chapter 9 discusses implementing safety guardrails, including Self-Check (Pattern 31) and comprehensive Guardrails (Pattern 32).

Who Is This Book For?

This book is for software engineers, data scientists, and enterprise architects who are building applications powered by GenAI foundational models. It captures proven solutions you can employ to solve the common challenges that arise when building GenAI applications and agents. Read it to learn how experts in the field are handling challenges such as hallucinations, nondeterministic answers, knowledge cutoffs, and the need to customize a model for your industry or enterprise. The age-old problems of software engineering have new solutions in this realm. For example, ways to meet latency and constrain costs include distillation, speculative decoding, prompt caching, and template generation.

Understanding the different patterns in this book requires different levels of background knowledge. For example, Chain of Thought (Pattern 13) requires no more than a knowledge of basic programming, Tool Calling (Pattern 21) requires an understanding of API design, and Dependency Injection (Pattern 19) requires some experience developing large-scale software. However, Content Optimization (Pattern 5) requires familiarity with statistics and ML, and Small Language Model (Pattern 24) requires an understanding of hardware optimization. We expect that 75% of the book can be read and understood by a junior software engineer or a third-year computer science student. The remainder will require specialized knowledge or experience.

AI engineering overlaps heavily with software engineering, data engineering, and ML—but in this book, we've limited our focus to core AI engineering. We encourage you to think of this book as a companion to the literature on patterns in related areas. Specifically, the book *Machine Learning Design Patterns* (O'Reilly), also coauthored by Valliappa Lakshmanan, covers proven solutions to recurring issues you'll encounter when training a bespoke machine-learning model for a specific problem.

You'll also likely find yourself working with both bespoke ML models and general-purpose foundational models, depending on the use case. In some situations, you might start with a foundational model but then find that edge cases require you to customize (or *fine-tune*) it for your problem. This book and *Machine Learning Design Patterns* are complementary and will help you work with both models, so we recommend that you read both.

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, 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.

In the diagrams, the boxes employ a set of color conventions as depicted in Figure P-1.

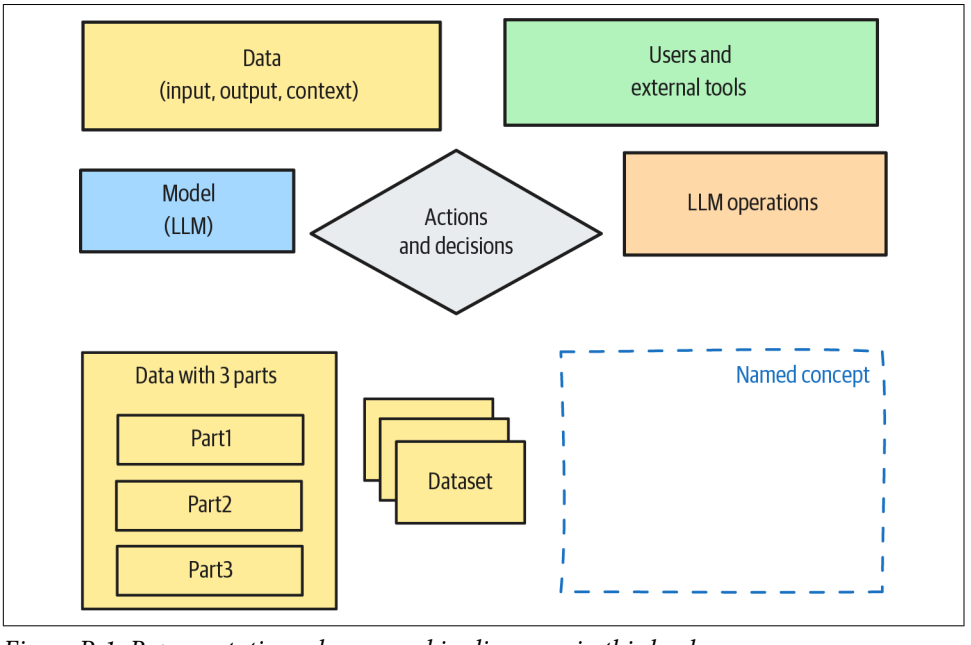


Figure P-1. Representation scheme used in diagrams in this book

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/lakshmanok/generative-ai-design-patterns*.

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: "*Generative AI Design Patterns* by Valliappa Lakshmanan and Hannes Hapke (O'Reilly). Copyright 2026 Valliappa Lakshmanan and Hannes Hapke, 979-8-341-62266-1."

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.
141 Stony Circle, Suite 195
Santa Rosa, CA 95401
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 and any additional information. You can access this page at *https://oreil.ly/genAI-design-patterns*.

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

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

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

Acknowledgments

Lak is thankful to his family for their forbearance as he (once again) vanished deep into writing and to collaborators and colleagues who gave him the opportunity to work far and wide with exciting new technology in practical ways. He's also deeply appreciative of Hannes for the partnership while writing this book.

Hannes would like to thank Lak for his insightful mentorship and guidance throughout the writing process. Lak's ability to explain complex topics in simple terms is truly exceptional, and Hannes is deeply grateful for being taken on this writing journey, from which he has learned immensely. This book would not have been possible without the unwavering support, endless patience, and love that Whitney, Hannes's partner, brought to every day of this process. Hannes is profoundly grateful for Whitney's amazing support, and he also extends his heartfelt appreciation to his family, especially his parents, who encouraged him to pursue his dreams around the world.

We are both thankful to the O'Reilly team (Nicole Butterfield, Corbin Collins, Catherine Dullea, Christopher Faucher, Sarah Grey, and Doug McNair [in alphabetical order]) for their unique blend of professionalism and flexibility. We were fortunate to have technical reviewers (David Cardozo, Mark Edmondson, Jason Fournier, Andrew Stein, and Glen Yu) who provided helpful, actionable, and speedy feedback on almost the entire book. In addition, Madhumita Baskaran, Ying-Jung Chen, Martin Gorner, Skander Hannachi, Ryan Hoium, and Danny Leybzon helped review specific chapters.

CHAPTER 1

Introduction

GenAI is so powerful and easy to use that even nontechnical users can easily prototype very compelling applications on top of GenAI. However, taking such GenAI prototypes to production is hard because GenAI models are unreliable—they can hallucinate, return different answers to the same input, and can have surprising limitations because of how they are trained. The design patterns in this book capture best practices and solutions to these and other recurring problems you're likely to encounter when building production applications on top of GenAI models.

GenAI Design Patterns

Design patterns, in software engineering, are proven solutions to common problems that occur during software design and development. They represent standardized best practices that have evolved over time through the collective experience of software developers. Design patterns are important because they establish a common vocabulary developers can use to communicate efficiently and because they help improve software quality, maintainability, and scalability.

The concept of design patterns was heavily influenced by the work of architect Christopher Alexander, who introduced patterns in architecture in his book *A Pattern Language* (Oxford University Press, 1977). Design patterns gained significant prominence in software engineering with the publication of the book *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (Addison-Wesley), which is often called "the Gang of Four book." Since then, design patterns have been cataloged for other software engineering domains, such as for Java Enterprise applications and ML.

When building AI products today, developers increasingly turn to *foundational* GenAI models (such as GPT-4, Gemini, Claude, Llama, DeepSeek, Qwen, and

Mistral) that are trained on large, application-agnostic datasets, rather than building custom ML models that need to be trained from scratch on application-specific data. In this book, we'll follow Chip Huyen's *AI Engineering* (O'Reilly) in referring to this approach of building on top of foundational models as *AI engineering* and to practitioners of this approach as *AI engineers*.

AI engineering has a wide range of applications—including natural-language processing (NLP), text generation, code explanation, image understanding, and video synthesis—to power use cases such as content generation, AI assistants, workflow automation, and robotics.

As an AI engineer, you can ask a foundational model to directly generate the content your application needs by sending the model an appropriate text input, which is known as a *prompt*. However, you will face certain common problems—the generated content may not match the style you want, may be missing enterprise knowledge that the model doesn't know about, or may lack certain capabilities. In this book, we catalog a variety of proven solutions to such problems that arise in the context of building applications on top of GenAI foundational models.

In this book, you will also find detailed explanations of 32 patterns that codify research advances and the experience of experts into advice that you can readily incorporate into your projects. Each chapter offers a set of patterns as potential solutions to a particular problem that commonly arises in AI engineering. For example, Chapter 3 is about solving the problem that foundational models can't generate content that is informed by confidential enterprise data, because they are trained by model providers who don't have access to that data. The patterns presented in that chapter all address this problem. Each section that presents a pattern includes a description of the problem, a proven solution, an end-to-end working example of the pattern, and a discussion of alternatives and other considerations for implementing it.

AI engineers often encounter tasks that are too complex for a foundational model to perform all at once, so a common tactic is to break the complex task into smaller components that can be accomplished by foundational models. Such small software components that provide capabilities with the help of foundational models are called *agents*. Agents become increasingly autonomous as they use GenAI models to plan out a sequence of operations, identify the backend tools that they can invoke for each operation, determine how to recover from errors, and/or evaluate whether the task is complete. Applications that are built by orchestrating agents are called *agentic*. By showing you how to handle the inevitable challenges that arise when building applications on foundational models, the patterns in this book will help you build better agents and agentic applications.

Building on Foundational Models

In this section, we'll quickly cover the basics of AI engineering so that we don't have to repeat this introductory material in the sections on the patterns that follow in later chapters. For deeper coverage of building GenAI applications, we refer you to books such as Omar Sanseviero et al.'s *Hands-On Generative AI with Transformers and Diffusion Models* (O'Reilly), which covers the underlying technology; Chris Fregly et al.'s *Generative AI on AWS* (O'Reilly), which covers hyperscaler offerings; and Leonid Kuligin et al.'s *Generative AI on Google Cloud with LangChain* (Packt), which covers an open source GenAI framework.

A Note on Models and Frameworks

In *Machine Learning Design Patterns*, we used just two frameworks (scikit-learn and TensorFlow) and a single hyperscaler (Google Cloud Platform [GCP]) for consistency, but many readers felt that the resulting examples were too TensorFlow- and GCP-heavy. Therefore, in this book, we endeavor to be agnostic to model, framework, and hyperscaler.

Our code examples employ a wide range of technologies from a number of different vendors: large language models (LLMs) from OpenAI, Anthropic, Google, Alibaba, and Meta; and GenAI frameworks like LangChain, Pydantic AI, Hugging Face, and DSPy. Our examples are also agnostic to hyperscalers such as Amazon Web Services (AWS), Azure, GCP, and Oracle Cloud Infrastructure. Since you're likely to be using a different model in a different framework to address a different scenario, the code examples are meant only to serve as starting points for your implementation—we fully intend that you will have to adapt the code examples to your preferred LLM, framework, and hyperscaler.

Prompt and Context

When you build AI applications, you typically invoke hosted foundational models through an API. This might be the API provided by the vendor of the foundational model, or it might be a framework that allows you to easily switch between providers.

You invoke a foundational model by sending it a *prompt* and getting back a *response*. You are, doubtless, familiar with doing this by using the web user interface of a foundational model. For example, on ChatGPT, you might type a prompt like this one into the text box:

> Create a pencil sketch in the style of Degas depicting a family of four playing a board game

The simplest prompt typically consists of an *instruction* to the model that asks it to perform some content-generation task. In this case, the model follows the instruction and sends back a response that contains an image of the type requested (see Figure 1-1).[1] Both prompts and responses can be *multimodal*—they could be text, but they could also be images, video, or audio.

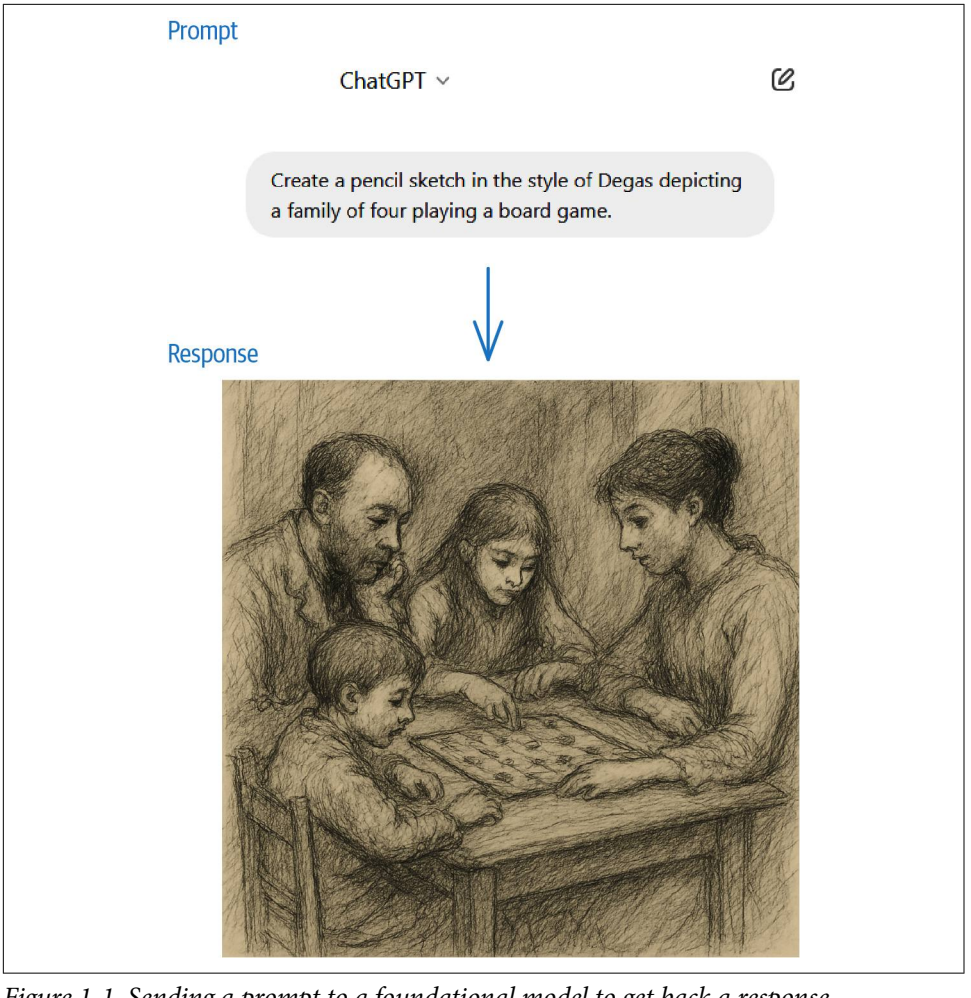


Figure 1-1. Sending a prompt to a foundational model to get back a response

1 Look carefully at the sketch. Do you see any issues? Do all the characters have the appropriate number of fingers? Is the girl's face symmetric? Does the board game correspond to anything recognizable? At the time of writing, image models struggle with symmetry, count, and exact recall.

A more complex prompt might consist not just of an instruction but also *context*. This might be information that the model should use in the generation, or it might involve asking the model to take on a specific role. For example, we could tell Claude (on Claude.ai) to take on a personality and use some information when generating content:

> You are an expert marketer who is very familiar with the book market in university towns in Germany.
>
> *Covenant of Water* is a novel that tells the story of three generations of an Orthodox Saint Thomas Christian family in Kerala.
>
> Write a one-paragraph blurb introducing the book to readers at a bookstore in Göttingen, drawing local connections.

The response from Claude Sonnet takes into account the role, context, and instruction:

> **Covenant of Water - A Literary Journey to Kerala**
>
> Nestled between the historic walls of Göttingen, where academic tradition meets literary curiosity, we present Abraham Verghese's masterful epic, "The Covenant of Water." This sweeping saga follows three generations of an Orthodox Saint Thomas Christian family in Kerala, India—a region whose complex religious heritage might resonate with students of our university's renowned theological studies program. As the story unfolds across decades amid Kerala's lush backwaters, readers familiar with Göttingen's own relationship with water—from the sacred Gänseliesel fountain to the meandering Leine River—will find surprising parallels in how geography shapes destiny. Verghese weaves a tale of family secrets, medical mysteries, and generational bonds that transcends cultural boundaries, offering our community of international scholars and lifelong learners a profound exploration of how water—whether in Kerala's monsoons or Lower Saxony's gentle rains—becomes both metaphor and covenant, connecting us across time and space.

Using the Model Provider's API

APIs that are designed to invoke foundational models are available in a wide variety of languages—Python, Go, Java, and TypeScript are all commonly employed. For simplicity, we focus on Python APIs in this book.

To invoke Claude Sonnet 3 through the Anthropic API in Python, you could use the Anthropic Python package as follows (the full code is on GitHub):

```
import anthropic
client = anthropic.Anthropic(
    # defaults to os.environ.get("ANTHROPIC_API_KEY")
    api_key="YOUR_ANTHROPIC_API_KEY",
)

completion = client.messages.create(
    model="claude-3-7-sonnet-latest",
```

```
    system="You are an expert Python programmer.",
    messages=[
        {
            "role": "user",
            "content": [
               {
                  "type": "text",
                  "text": """
Write code to find the median value of a list of integers.
"""
               }
            ]
        }
    ]
)

print(completion.content[0].text)
```

In this API call, the prompt has been broken into two parts—a *system prompt* and a *user prompt*. The system prompt is set by the developer and guides the model's overall behavior, while the user prompt is more dynamic and provides specific instructions for a specific task you want the model to perform. Here, the AI assistant's role has been set in the system prompt while the user prompt and context are sent as messages.

Using an LLM-Agnostic Framework

To perform the same task using the PydanticAI framework, you'd use code such as the following (assuming the needed API key is set in an environment variable):

```
from pydantic_ai import Agent
agent = Agent('anthropic:claude-3-7-sonnet-latest',
              system_prompt="You are an expert Python programmer.")

result = agent.run_sync(
           "Write code to find the median value of a list of integers.")
print(result.data)
```

The advantage here is that you can easily switch between foundational model providers by switching the model string to `openai:gpt-4o-mini`, `google-vertex:gemini-2.0-flash`, `groq:llama3-70b-8192`, and so on (see Pydantic's documentation for the full list of models supported).

The class in the Pydantic API that invokes the Claude model is called `Agent`. We'll discuss what agents are in the next section, but before that, let's conclude our discussion of ways to invoke foundational models.