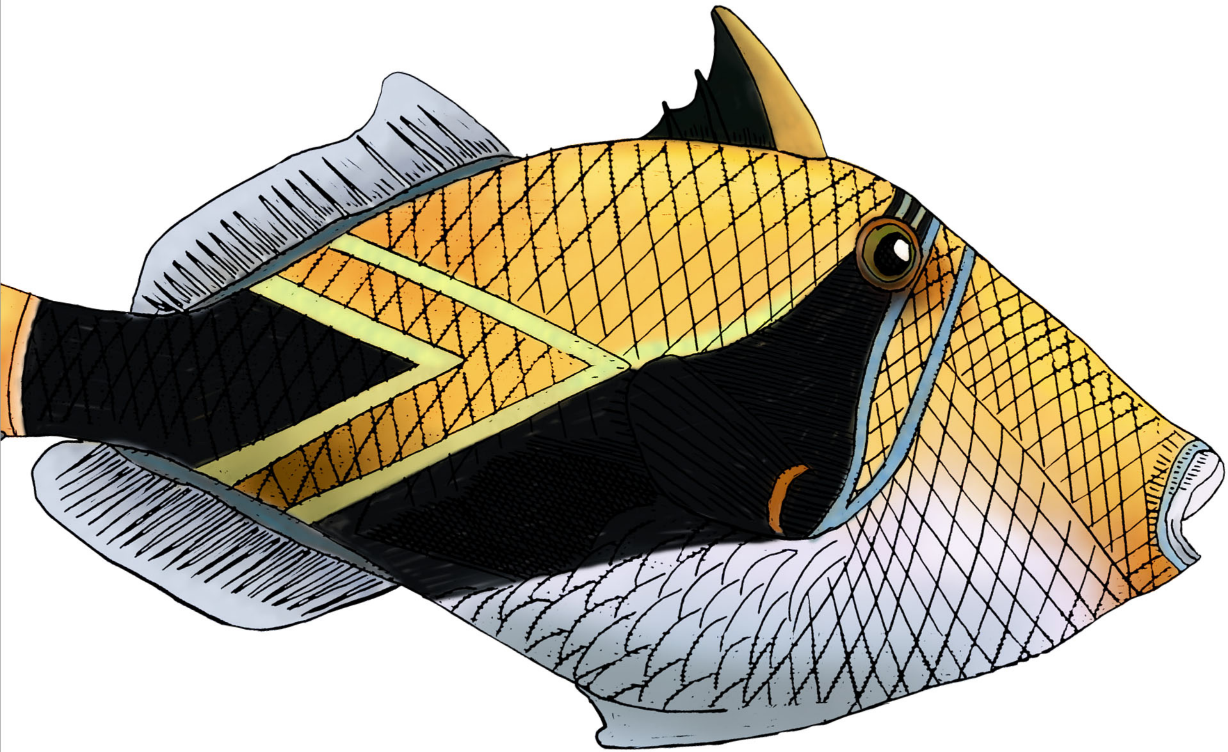
O'REILLY®

AI-Assisted Programming

Better Planning, Coding, Testing, and Deployment



Tom Taulli

O'REILLY®

AI-Assisted Programming

Get practical advice on how to leverage AI development tools for all stages of code creation, including requirements, planning, design, coding, debugging, and testing. With this book, beginners and experienced developers alike will learn how to use a wide range of tools, from general-purpose LLMs (ChatGPT, Gemini, and Claude) to code-specific systems (GitHub Copilot, Tabnine, Cursor, and Amazon CodeWhisperer).

You'll also learn about more specialized generative AI tools for tasks such as text-to-image creation.

Author Tom Taulli provides a methodology for modular programming that aligns effectively with the way prompts create AI-generated code. This guide also describes the best ways of using general-purpose LLMs to learn a programming language, explain code, or convert code from one language to another.

This book examines:

- **The core capabilities of AI-based development tools**
- **Pros, cons, and use cases of popular systems, including GitHub Copilot**
- **Ways to use ChatGPT, Gemini, Claude, and other generic LLMs for coding**
- **Using AI development tools for the software development lifecycle, including requirements, planning, coding, debugging, and testing**
- **Prompt engineering for development**
- **Using AI-assisted programming for tedious tasks like creating regular expressions**
- **How to use AI-based low-code and no-code tools**

> "When we added vector search to Cassandra in six weeks, Copilot and ChatGPT were key to meeting our deadline, but most developers have yet to take advantage of AI tools like these. Tom's book is a great way to get started and will save you hours of trial and error."
>
> **—Jonathan Ellis**
> Cofounder and CTO of DataStax

Tom Taulli is an author, advisor, and investor who's penned numerous books, including *Artificial Intelligence Basics*. He also contributes to publications such as AIBusiness.com, Inc.com, Barrons.com, eSecurity Planet, and Kiplingers.com, and has developed educational courses for O'Reilly and Pluralsight, focusing on areas such as generative AI, databases, and Python.

PROGRAMMING LANGUAGES

US $69.99 CAN $87.99
ISBN: 978-1-098-16456-0

linkedin.com/company/oreilly-media
youtube.com/oreillymedia

Praise for *AI-Assisted Programming*

When we added vector search to Cassandra in six weeks, Copilot and ChatGPT were key to meeting our deadline, but most developers have yet to take advantage of AI tools like these. Tom's book is a great way to get started and will save you hours of trial and error.

—Jonathan Ellis, cofounder and CTO of DataStax

AI-Assisted Programming is an excellent resource that showcases Tom's expertise and equips readers for the current evolution of software development, empowering everyone to code.

—Justin Dorfman, Open Source Community Manager at Sourcegraph

AI is rapidly changing how developers build software. From code editors to terminal and more, AI assistance is becoming pervasive. *AI-Assisted Programming* should be the first read for any developer trying to get the most from AI in their daily workflows.

—Zach Lloyd, CEO and cofounder of Warp

Tom Taulli's book is a well-structured journey into how AI tools like ChatGPT can change the game for developers—both early in their careers and for seasoned experts. It shares applicable insights on the ups and downs of AI-powered coding, much like my own journey from marketing and business to shipping features in production for our app. Great for programmers looking to leverage AI as part of their toolbox.

—Titus Capilnean, cofounder at Private Market Labs

AI has changed the game for development. Every programmer will need to know how to work with tools like GitHub Copilot. Tom's book shows how.

—Muddu Sudhakar, CEO and cofounder of Aisera

AI-Assisted Programming

Better Planning, Coding, Testing, and Deployment

Tom Taulli

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

AI-Assisted Programming
by Tom Taulli

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

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

Acquisitions Editor: Brian Guerin
Development Editor: Shira Evans
Production Editor: Kristen Brown
Copyeditor: Paula L. Fleming
Proofreader: Emily Wydeven
Indexer: Potomac Indexing, LLC
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

April 2024: First Edition

Revision History for the First Edition
2024-04-10: First Release

See *https://oreilly.com/catalog/errata.csp?isbn=9781098164560* for release details.

978-1-098-16456-0

[LSI]

Table of Contents

Foreword

When I graduated from UCLA in the early 1990s, I thought I would be a professor. But when IBM hired me, I quickly fell in love with the technology industry. There was no turning back. I would go on to work at other companies like VMware, Pivotal, EMC, and SGI.

Then I founded several startups. My latest is Aisera, which builds generative AI solutions for enterprises. Our platform helps with customer service, employee experience, enterprise search, IT service desk, and AIOps.

Throughout my career, I've seen many innovations, such as the emergence of the internet, cloud computing, mobile devices, and deep learning. Yet there was one area of technology that saw little progress: software development.

As a founder, this was certainly a big frustration. There would often be certain things we wanted to do but could not because of the bottlenecks with our engineering teams. But with AI-assisted programming, the game has changed in a big way. It has made Aisera more agile and nimble. It's been a force multiplier that has helped propel our growth.

Now, when I consider hiring a developer, I want to know how they leverage AI. How are they using this powerful technology to handle tedious processes and code faster?

Bottom line: I believe that understanding AI-assisted programming tools is a critical skillset. According to Microsoft CEO Satya Nadella, they are "becoming standard issue for any developer…"

But understanding AI-assisted programming requires a different approach. It's not like typical development. You need to understand how to use generative AI systems. This is where Tom's book comes in. He has written a playbook for any developer who wants to become proficient with AI-assisted programming. He's covered the key topics and has provided many useful examples. He also has done this in a fun manner, making the topic approachable.

No doubt, it's an exciting time to be a developer. There are many opportunities for career growth. And one of the best moves you can make is learning AI-assisted programming.

— Muddu Sudhakar
CEO and cofounder of Aisera

Preface

I started coding back in the early ’80s. My first rig was an Atari 400—not exactly a powerhouse with its membrane keyboard, just 8K of RAM, and programs that loaded from cassettes. I learned the BASIC language and created simple games and utility apps.

I was totally into it.

Naturally, over time I upgraded to beefier machines. Along the way, I got my hands dirty with languages like Pascal, C, and C++. But the IDEs didn’t change much, except for syntax highlighting and debugging features.

Everything changed dramatically with the arrival of GitHub Copilot and ChatGPT. Trying out these tools felt like the moment I first held an iPhone—absolutely game-changing.

In natural language, I asked ChatGPT to write code. Or in VS Code, I would type a fragment of a function, and GitHub Copilot would generate a code block. Often it was like hitting the bull’s-eye. Oh, and then I used ChatGPT to turn images into code.

Yet the real power was that these tools could handle many of the tedious tasks for developers. Who’s into wrestling with regex statements or piecing together bash commands or GitHub actions? Not me, for sure. But these AI programming tools? They eat that stuff for breakfast.

Turns out, these AI tools are handy for more than just coding. I began using ChatGPT for brainstorming app ideas, drafting requirements, and even knocking out unit tests.

It didn’t take long for me to be convinced that AI-assisted programming would be one of those must-have skills for programmers.

So yes, I saw a big need to write this book. I put together an outline and pitched it to O’Reilly. The folks there instantly saw the potential.

Writing the book has been lots of fun, and I've learned a lot. I've also interviewed many smart developers who have provided me with great ideas and tips.

But AI-assisted programming is quickly evolving. This is why this book has a companion GitHub repository. Here, I'll make updates to the book and include other important developments in this exciting field.

So thank you for picking up this book. I hope you find it not only informative but also a valuable guide in your journey.

What's Covered

Here's a brief look at each chapter:

- Chapter 1, "New World for Developers": This chapter kicks off with a look into how generative AI is changing the game for coders. It talks about how these AI tools are helping developers think more about the big picture and less about the nitty-gritty of coding. The chapter also takes a stroll through the history of programming languages. There are also details about the advanced AI technologies like GPT-4.
- Chapter 2, "How AI Coding Technology Works": The chapter starts off by explaining generative AI and why transformer models and large language models are big deals in the programming world. To top it off, there's a walkthrough of OpenAI's Playground, showing how you can play around with these AI models and tweak them to suit your coding needs.
- Chapter 3, "Prompt Engineering": The information here is critical to using AI-assisted programming tools. This chapter is packed with practical tips, like dealing with wordy or confusing prompts and deterring AI from making stuff up. Plus, it breaks down the key parts of a prompt and shows you how to use them effectively.
- Chapter 4, "GitHub Copilot": This chapter is a walkthrough of this powerful tool. There's a look at the core features like creating code with comments, Chat, and using an AI-powered command-line interface. There is also coverage of customizing the system for proprietary codebases.
- Chapter 5, "Other AI-Assisted Programming Tools": This chapter details the other top AI-assisted programming tools like Amazon CodeWhisperer, Google's Duet AI, and Replit to name just a few.
- Chapter 6, "ChatGPT and Other General-Purpose LLMs": This covers how to use these tools for tasks like handling regular expressions, starter code, and GitHub Actions.

- Chapter 7, "Ideas, Planning, and Requirements": The focus here is on using chatbots to kick off software projects. This involves topics like brainstorming, market research, requirements documents, and test-driven development.
- Chapter 8, "Coding": This chapter goes through common scenarios for development, whether working with APIs, using modular programming, or refactoring. There's also a look at handling functions and object-oriented programming.
- Chapter 9, "Debugging, Testing, and Deployment": This chapter is about the less glamorous parts of development. It covers topics like fixing bugs, using AI-assisted programming tools for code reviews, making unit tests, and describing pull requests.
- Chapter 10, "Takeaways": This is a wrap-up of the book, emphasizing the main points.

How This Book Is Different

Software developers thrive on certainty. When you give a program certain input, you always get the same output. For ages, this pure deterministic logic was the heart and soul of software.

But when you use AI-assisted programming tools, things get a bit topsy-turvy. Getting results is like rolling dice since everything works on probabilities. When you prompt an AI tool to whip up some code, and even use the same prompt over multiple tries, you might get different results each time. Sure, it's a bit of a head-scratcher at first, but once you get the hang of it, it's totally worth it. That's why there is a chapter on prompt engineering that will help with this new approach to programming.

Who Should Read This Book

This book is for any developer, whether you're just starting out or you've been in the game for many years.

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 general note.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/ttaulli/AI-Assisted-Programming-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-Assisted Programming* by Tom Taulli (O'Reilly). Copyright 2024 Tom Taulli, 978-1-098-16456-0."

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

O'REILLY®

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://www.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-assisted-programming*.

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

I want to thank the folks at O'Reilly who believed in this book and worked hard to make it a reality: Nicole Butterfield, Shira Evans, and Brian Guerin. I also had the benefit of outstanding tech reviewers. They include Roja Boina, Abraham Borg, Sarah Kim, Ebubechukwu (Nnenna) Oguaju-Dike, and Gaurav Deshmukh.

CHAPTER 1

New World for Developers

While juggling dense neural network architectures and pixel-wrangling computer vision at Stanford from 2011 to 2016, Andrej Karpathy also moonlighted at Google. Over there, he tinkered around and whipped up a feature-learning system for YouTube videos. Then he decided to become a founding member of OpenAI and later the senior director of AI at Tesla, where he led a team to create the Autopilot system.

It's safe to say he's one the world's top coders. He is also a skilled wordsmith with a massive Twitter—or X—following of nearly 800,000 followers. When ChatGPT catapulted onto the scene, he tweeted:

> The hottest new programming language is English.

He wasn't kidding. This wasn't just a poetic ode to coding but a nod to a future where typing out natural language prompts could conjure up computer code in seemingly any language. It's like having a bilingual genie in your computer, ready to transcribe your English wishes into code commands.

Then there came a tweet that echoed the sentiments of many developers:

> Copilot has dramatically accelerated my coding, it's hard to imagine going back to "manual coding". Still learning to use it but it already writes ~80% of my code, ~80% accuracy. I don't even really code, I prompt. & edit.

Karpathy was tipping his hat to Microsoft's GitHub Copilot, a fresh brew of AI-assisted programming. But it wouldn't be long until many other tools sprouted up. The pace of innovation was breathtaking.

Now, for all the coders out there, the landscape might look like a dense jungle. What's this brave new world of AI tools? Where do they dazzle, and where do they fizzle? And how do you wade through all this to become a savvy AI-assisted programmer?

Well, this book will be your guide to help answer these questions—and many more. The spotlight will be on harnessing these tools to code not just faster but smarter, and with a sprinkle of fun. So, let's roll up our sleeves and jump into this AI-assisted programming journey.

Evolution and Revolution

A key theme of the evolution of programming languages is *abstraction*. This is a fancy way of describing how systems get easier for developers to use. When the tedious details are handled in the background, developers can focus on what matters most. This has been a driving force of innovation, allowing for breakthroughs like the internet, cloud computing, mobile, and AI.

Figure 1-1 highlights the evolution of abstraction over the decades.

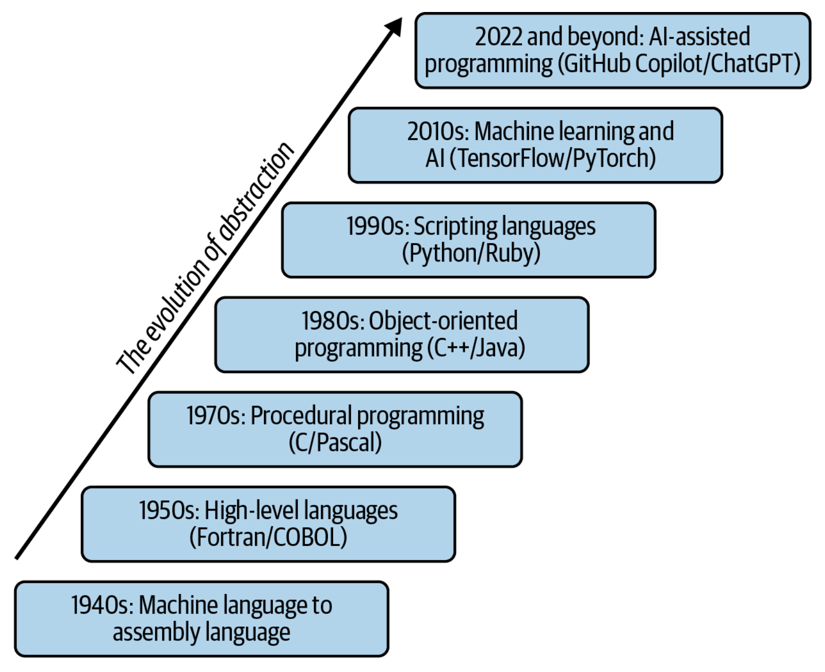


Figure 1-1. The abstraction of programming languages and tools has evolved over the decades

Let's go into more detail, starting from the 1940s:

Machine language to assembly language

At the dawn of the computer age, programmers had to wrestle with 0s and 1s to bend machines to their will. But then, assembly language came onto the scene. It offered alphanumeric instructions, which made coding easier and less error-prone.

High-level languages

The 1950s brought us Fortran and COBOL, languages that let programmers code using somewhat plain English like DISPLAY, READ, WRITE, and IF/THEN/ELSE. A compiler would convert these into the 0s and 1s that a computer could understand. At the same time, people without a technical background could generally read the code well enough to understand the workflow. The emergence of high-level languages would be a huge catalyst for the computer revolution.

Procedural programming

Languages like C and Pascal introduced procedural programming, essentially packing complex tasks into neat little boxes called functions. This abstraction allowed for reusability and maintainability, and it made managing colossal software projects less of a Herculean task.

Object-oriented programming (OOP)

Some of the stars of this type of computer language include C++ and Java. Object-oriented programming brought a whole new level of abstraction, allowing programmers to model real-world entities using classes and objects, encapsulating both data and behavior. This promoted modularity and allowed for more intuitive problem solving.

Scripting languages and web development

Python, Ruby, and JavaScript abstract many of the lower-level tasks associated with programming. They offer extensive libraries and built-in data structures, simplifying common programming tasks and reducing the amount of code needed to accomplish them.

Machine learning and AI

With the rise of AI and machine learning, specialized libraries and frameworks like TensorFlow and PyTorch have abstracted away many intricate mathematical details of programming. This has enabled developers to focus on model architecture and training processes.

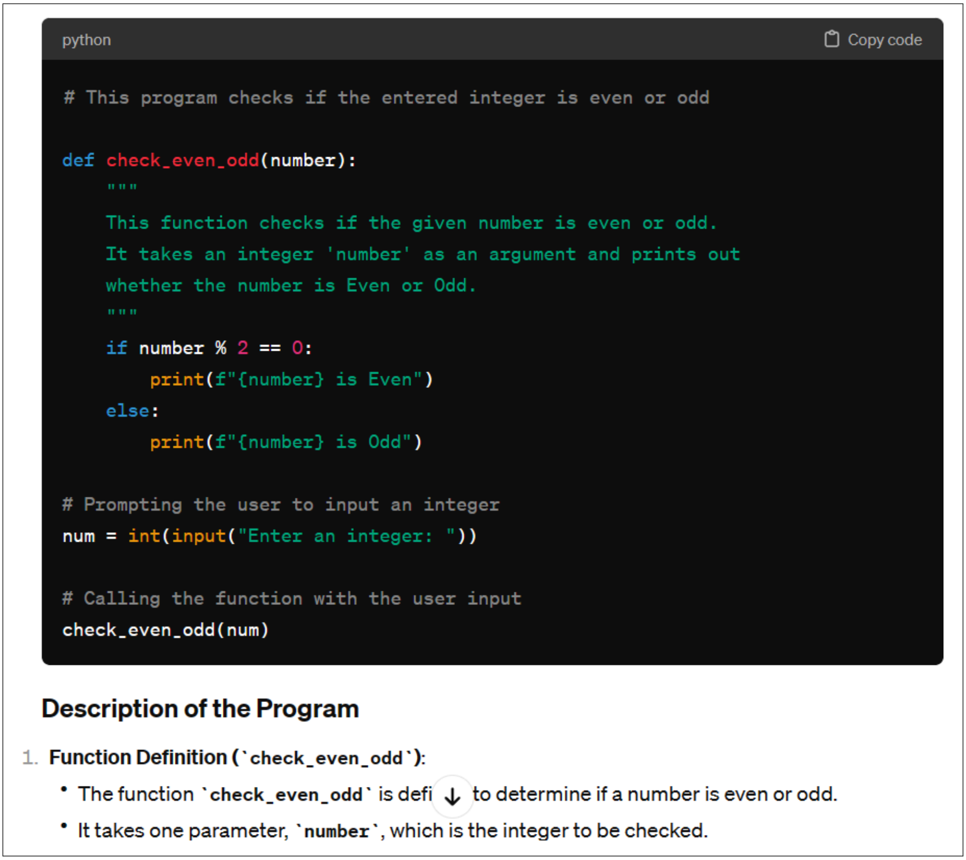
AI-assisted programming

Of course, the latest entrant to this abstraction narrative is AI-assisted programming, á la GPT-4 and other massive large language models (LLMs). These are like your backstage crew, ready to pitch in with code generation at your command.

Let's look at a simple example. For this, we'll use ChatGPT, which has a robust ability to gin up code. We will use a prompt to ask what we want the system to do. Suppose we give it the following prompt:

> *Prompt:* In Python, write a program that checks if a given integer is even or odd and print the result.

Figure 1-2 shows the response from ChatGPT.



```python
# This program checks if the entered integer is even or odd

def check_even_odd(number):
    """
    This function checks if the given number is even or odd.
    It takes an integer 'number' as an argument and prints out
    whether the number is Even or Odd.
    """
    if number % 2 == 0:
        print(f"{number} is Even")
    else:
        print(f"{number} is Odd")

# Prompting the user to input an integer
num = int(input("Enter an integer: "))

# Calling the function with the user input
check_even_odd(num)
```

Description of the Program

1. **Function Definition (`check_even_odd`):**
    - The function `check_even_odd` is defi to determine if a number is even or odd.
    - It takes one parameter, `number`, which is the integer to be checked.

Figure 1-2. When asked to create code, ChatGPT's response will include not only a listing but an explanation

We get the code listing, which even comes with helpful comments. Then there is also an explanation of how the program works. You can press the Copy code button at the top right to include the code in your IDE and run it.

Generative AI

Before we go deeper into how AI-assisted programming tools work, let's get an overview of generative AI. This is the foundation of these systems.

Generative AI is a branch of artificial intelligence (AI), which allows for the creation of new and unique content. Figure 1-3 provides a visual of how the different parts relate to each other.

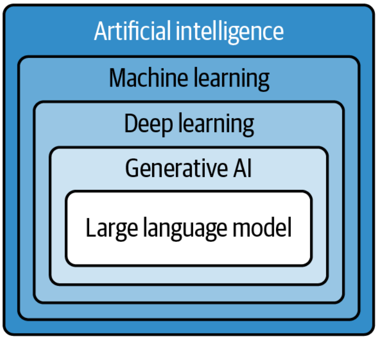


Figure 1-3. There are different types of AI, and they can be represented as nested subsets, with generative AI and finally large language models at the center

AI is the big umbrella: it includes all systems that can pull off tasks with the flair of human intelligence. Tucked within AI is machine learning (ML). Instead of marching to the beat of explicit instructions, ML systems come up with insights based on heaps of data. ML is generally based on complex algorithms, which allow for making predictions or decisions without hardcoding.

Take a step deeper, and you get deep learning (DL), a tighter slice of ML that rolls with neural networks stacked with hidden layers—hence the *deep* tag. These stacked models have shown standout results in areas like image and speech recognition.

Within the corridors of deep learning, you'll find generative AI (or GenAI). GenAI models create new data that reflects their training data.

In the innermost circle sits LLMs, such as GPT-4, Gemini, Claude, and LLaMA 2. These powerful models—often called "foundation models"—churn out human-esque text based on cutting-edge algorithms and training on huge amounts of data.

But generative AI is more than just LLMs. GenAI also has multimodal capabilities, meaning the ability to create images, audio, and video.

In the next chapter, we'll dive deeper into how generative AI works. But next, let's now take a look at the pros and cons of AI-assisted programming tools.

The Benefits

AI-assisted programming tools are crafted to enhance developers' abilities, enabling them to zero in on advanced problem solving and innovations instead of being ensnared in monotonous tasks or complex code details. This is why GitHub's use of the word *copilot* is spot on. It's about having that reliable buddy in the cockpit, navigating through the intricate and often tedious aspects of coding, allowing you to focus on what matters.

In the upcoming sections, we'll spotlight the benefits and practical applications of these powerful systems.

Minimizing Search

Developers often find themselves playing digital detectives, hunting down pesky bugs or wrapping their heads around cryptic codes. When they bump into a snag, their first instinct is to hit up Google or pay a visit to Stack Overflow. A quick search, a snippet of code, and voilá, they're back to their IDE (integrated development environment).

But sometimes this can turn into an ordeal. The discussion on Stack Overflow may wind up being a dead end. You search some more—but nothing seems to be on point. However, there's one discussion that somewhat helps, and you do further research on some related topics. You even search YouTube for a video. After chewing on the problem for more than 30 minutes, you finally solve it.

Yes, all developers have experienced this. Interestingly enough, the 2022 Developer Survey from Stack Overflow, which included responses from more than 70,000 developers, highlights this frustration. It found that 62% of the respondents spent more than 30 minutes a day searching for answers, and 25% spent over an hour a day. According to the survey, "For a team of 50 developers, the amount of time spent searching for answers/solutions adds up to between 333–651 hours of *time lost per week* across the entire team."

Now, what if there was a way to slice through this thicket of time-consuming searches and get to the solution pronto? Enter AI-assisted programming, our knight in shining algorithm. Research from Microsoft supports this: it shows that more than 90% of developers who used GitHub Copilot managed to race through their tasks at a faster clip.

Microsoft even put this to the test in a coder showdown. The company recruited 95 professional developers and split them into two groups. The task was to write an HTTP server in JavaScript. Those who used GitHub Copilot completed the job 55% faster than those who did not.

And it's not just Microsoft singing praises. McKinsey & Company also conducted a research study. More than 40 developers from across the United States and Asia participated, with varying degrees of experience and backgrounds. Over several weeks, they completed three common software tasks: code generation, refactoring, and documentation.

The results? When it came to documentation for keeping the code neat and tidy, AI-assisted tools were the standouts, cutting the time spent by half, and AI tools performed nearly the same on drafting new code and refactoring.

However, for complex tasks, the AI tools didn't hit the high notes. The time trimmed was shy of 10%.

Interestingly, the research also showed that reducing the time spent did not negatively impact the overall quality of the code, as reflected in, for example, bugs, readability, and maintainability. In fact, the AI-assisted programming tools provided marginal improvements. But this often was due to the fact that developers iterated with the tools.

The McKinsey study provides the following takeaways:

Easing routine chores
: The tools are great at tackling mundane tasks like autofilling code functions, aiding in real-time code completion, and autodocumenting code. By handling these tasks, they free up developers to dive into complex business issues and speedily deploy software features.

Producing smoother code drafts
: Staring at a blank canvas can be daunting, but with generative AI tools, developers can nudge the creative process along by fetching code suggestions with a simple prompt, right within their IDE or separately. Many developers found these AI-based suggestions invaluable, as they helped the humans overcome the "blank screen problem" and get into the coding "zone" with a quicker pace.

Accelerating tweaks to existing code
: With effective prompts, developers can adapt and improve existing code more swiftly. For instance, they can snag code from online libraries, pop it into a prompt, and then make iterative requests for AI-finessed adjustments based on specified criteria.

Enhancing developers' prep for new challenges

The technology acts like a fast-track introductory course and helps developers get acquainted with unfamiliar coding environments or languages. When tackling something new, these tools step in like a seasoned buddy, shedding light on fresh concepts, dissecting various code bases, and dishing out comprehensive guides on framework usage.

Harnessing multiple tools

The research indicates that bringing multiple tools into play is more effective. Picture this: a developer swings one tool for prompts or chats, and another tool jumps in as part of the codebase, dishing out autocomplete options and suggestions. Developers found the first tool to be a whiz at fielding queries during code refactoring, thanks to its conversational finesse. On the flip side, the second tool showed effectiveness in conjuring up new code that was integrated smoothly with the development environment. When these AI tools teamed up for a task, developers saw a time efficiency surge of 1.5 to 2.5 times.

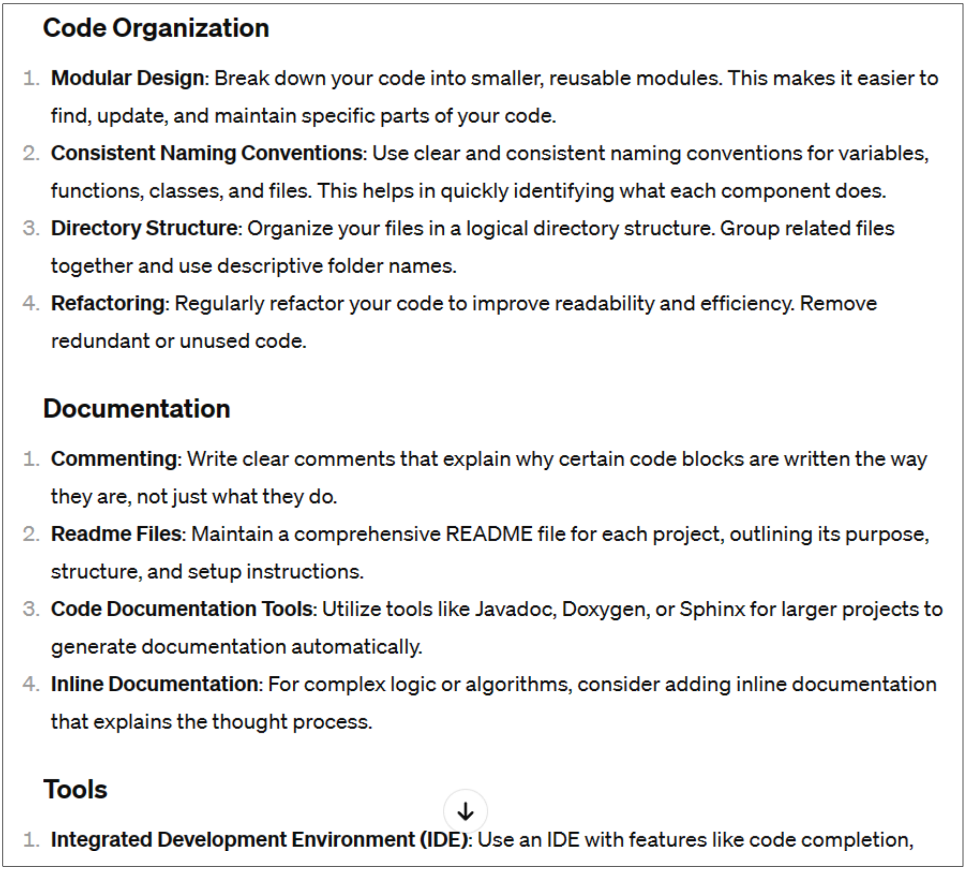
Your Advisor

With ChatGPT, you can ask for advice on many types of development activities. Here's a prompt:

> *Prompt:* Please provide detailed tips and best practices for minimizing search time and enhancing productivity when programming. Include strategies related to code organization, documentation, tools, and mindset.

Figure 1-4 shows the response.

ChatGPT provides three main areas to consider. It recommends using a modular design, maintaining consistent naming, and organizing files logically. It also advises prioritizing clear documentation with comments, docstrings, and READMEs. ChatGPT then goes on to mention using the search functions of an IDE, using tools like Git, and bookmarking key resources.



Code Organization

1. **Modular Design**: Break down your code into smaller, reusable modules. This makes it easier to find, update, and maintain specific parts of your code.
2. **Consistent Naming Conventions**: Use clear and consistent naming conventions for variables, functions, classes, and files. This helps in quickly identifying what each component does.
3. **Directory Structure**: Organize your files in a logical directory structure. Group related files together and use descriptive folder names.
4. **Refactoring**: Regularly refactor your code to improve readability and efficiency. Remove redundant or unused code.

Documentation

1. **Commenting**: Write clear comments that explain why certain code blocks are written the way they are, not just what they do.
2. **Readme Files**: Maintain a comprehensive README file for each project, outlining its purpose, structure, and setup instructions.
3. **Code Documentation Tools**: Utilize tools like Javadoc, Doxygen, or Sphinx for larger projects to generate documentation automatically.
4. **Inline Documentation**: For complex logic or algorithms, consider adding inline documentation that explains the thought process.

Tools

1. **Integrated Development Environment (IDE)**: Use an IDE with features like code completion,

Figure 1-4. You can get useful advice on programming tasks from ChatGPT

IDE Integration

Seamless integration with the IDE is crucial for AI-assisted programming. It keeps the momentum of the development process going strong, without the heavy lifting of mastering a new platform. This means less time scrambling up the learning curve and more time coding and—let's not forget—less switching between different platforms or tools means less friction and makes for a smoother coding journey.

Then there is the advantage of real-time feedback. As developers knit together or tweak code, integrated tools are right there to spotlight errors, offer up corrections, or suggest a better way to get things done. This instantaneous back-and-forth of writing, feedback, and tweaking is like having a friendly coach by your side. You'll be guided toward cleaner, more efficient code without the hassle of manual reviews or external checks.

AI-assisted systems can also amp up an IDE by tuning into the broader coding narrative. The AI gets the gist of variable types, method signatures, and even the project's structural blueprint to churn out relevant code suggestions. It's not just about spitting out code, though.

Table 1-1 introduces some of the top AI-assisted programming tools and the IDEs they support.

Table 1-1. IDEs supported by popular AI-assisted programming tools

AI-assisted programming tool	IDEs
GitHub Copilot	Visual Studio Code, Visual Studio, Vim, Neovim, JetBrains suite, Azure Data Studio 1
Tabnine	Visual Studio Code, WebStorm, PyCharm, Eclipse, IntelliJ Platform, PhpStorm, CLion, Neovim, JupyterLab, Rider, DataGrip, AppCode, Visual Studio 2022, Android Studio, GoLand, RubyMine, Emacs, Vim, Sublime Text, Atom.AI, Jupyter Notebook 2
CodiumAI	Visual Studio Code, JetBrains (IntelliJ, WebStorm, CLion, PyCharm)
Amazon CodeWhisperer	Visual Studio Code, IntelliJ IDEA, AWS Cloud9, AWS Lambda console, JupyterLab, Amazon SageMaker Studio, JetBrains (IntelliJ, PyCharm, CLion, GoLand, WebStorm, Rider, PhpStorm, RubyMine, DataGrip)

A research study from Microsoft showed that 88% of users of GitHub Copilot felt less frustrated and more focused. A key reason was that staying within the IDE meant spending less time searching. This allowed for the developer to remain in the "flow state."

Reflecting Your Codebase

Certain AI-assisted programming tools are tailored to mesh well with specific development environments. Developers have the leeway to fine-tune them, allowing the tool to understand a project's internal libraries, APIs, best practices, and architectural blueprints. This ensures that the suggestions thrown your way not only are technically solid but also dovetail with your project's unique needs.

This customization helps to align the generated code suggestions with your organization's established coding standards, quality markers, and security protocols. The focus on fostering high-quality code means that teams can avoid stumbling into deprecated or undesirable code snippets.

Moreover, this tailored approach is a big benefit for newcomers to a development team. Traditionally, getting them acclimated to a new codebase requires a hefty time investment as they may need months of exploring code, reviewing documentation, and learning the ropes of coding protocols. However, an AI-assisted programming tool can significantly shave time off this learning curve.

Code Integrity

Code integrity is a hallmark of sound software development. It highlights the sturdiness and trustworthiness of the source code in executing its intended function. Think of it as a lens through which the completeness, accuracy, consistency, and fortification of the code are examined. A hiccup in code integrity lays out a welcome mat for bugs and potential security blind spots, which, in turn, could usher in system crashes and data breaches.

The various factors that engender code integrity include its precision, thoroughness, uniformity, and security provisions as well as the ease with which it can be maintained. Developers can ramp up code integrity through a medley of approaches like unit and integration testing, peer code reviews, static code analysis, and stringent security assessments.

It's worth noting that a growing roster of AI-assisted programming tools are rolling out features aimed at bolstering code integrity. They delve into the finer points of the code, paving the way for the generation of pertinent and sharp unit tests and edge cases.

Some of these tools come with "fix-it" recommendation features. These are vetted in advance to ensure they don't lead to new problems before they land in front of developers. Then developers can review and assimilate these suggestions right within their IDE.

An added perk of these tools is their ability to swiftly analyze pull requests and spin up succinct summaries of code alterations. They also have a knack for automating the chore of generating release notes, which comes in handy for documenting the evolution in software versions.

AI-Powered Documentation Generator

Documentation is the unsung hero in the software development process. It helps to ensure that the codebase remains legible, maintainable, and scalable, especially as teams morph and projects bloat in complexity. But let's face it, creating and refreshing this documentation often feels like a trek through a bureaucratic bog—it can be a time-guzzler and, occasionally, gets shoved to the backburner.

Now, cue the entrance of AI-assisted programming tools. These digital scribes can whip up extensive documentation in a fraction of the time—and with a hefty dose of quality and clarity to boot. This is done by leveraging the power of LLMs, which are particularly strong at dealing with language.

Modernization

Marc Andreessen's 2011 bold statement in the *Wall Street Journal*, "Software Is Eating the World", has aged like a fine wine. Andreessen, known for his knack for spotting tech trends from miles away and his stellar track record as a successful entrepreneur and venture capitalist, pointed out a ripe moment in tech history.

He underlined how the infrastructure had come of age and primed global industries for a metamorphosis. The rise of cloud platforms like Amazon Web Services and the widespread growth of broadband internet were game changers. They had knocked down the traditional hurdles of server costs and network know-how. This had cleared the stage for disruptors like Uber, Netflix, and a slew of social media platforms to rewrite the rulebook of their respective industries.

When we fast forward from Andreessen's insightful piece, we see that the innovation express has only picked up steam. However, it has also brought along a threat of disruption, especially for large corporations. Many of these behemoths are anchored to legacy systems that are not only pricey but also a gamble to modernize. Their hierarchical setup can interpose speed bumps in decision making, and their expansive scale adds layers of complexity to embracing change. Plus, their workforce might not always be on the same page with the latest tech innovations.

Enter IBM, eyeing this scenario as a goldmine of opportunity and channeling its hefty resources to craft AI-assisted programming tools for its customers. In October 2023, it unveiled the watsonx Code Assistant for Z. This system can translate COBOL to Java on mainframe systems, with the code output elegantly object oriented.

IBM's Watsonx.ai model understands 115 coding languages based on 1.5 trillion tokens. The model has about 20 billion parameters. This is one of the largest AI systems for code development.

The fact is that there are hundreds of billions of lines of COBOL. But migrating this language to modern ones is no easy feat. It's common for the COBOL to be decades old and have little or no documentation. If the conversion is not handled properly, the consequences could be severe. Keep in mind that much of the world's credit card processing is handled with mainframes. The same goes for Uncle Sam's system for handling school loans.

Unfortunately, there are many examples of failed migration projects. Consider the California Department of Motor Vehicles, which, despite pouring $208 million into the effort, had to pull the plug within a few years. Ouch.

Given the high stakes, mainframe developers generally earn higher salaries. But companies still are challenged in recruiting talent. Younger developers are trained on modern languages and perceive mainframe development as a dead end. In the meantime, a growing number of seasoned mainframe developers are retiring.

IBM realized that AI is essential to solve this massive problem. It's true that code transpilers or translators have been around for decades. In fact, they have often been used for mainframe projects. However, what they have mostly been doing is taking COBOL's spaghetti code, giving it a quick translation, and, well, you have Java spaghetti code. It's a modest facelift with barely a hint of improvement or innovation. The Java code still needs a good amount of elbow grease, explaining why many projects stumbled or flat-out face-planted.

But by using generative AI, IBM says that it has been able to improve the results of a project by as much as tenfold.

Other companies are exploring this modernization opportunity. Thomas Dohmke, who is the CEO of GitHub, posted: "COBOL still running on main frames is a much bigger societal problem than we think." In an interview with *Fortune*, he noted that he had heard more about COBOL in 2023 than during the past three decades. He also said that companies have been asking how to use GitHub Copilot for their migration projects.

Keep in mind that ChatGPT is also proficient with legacy programming languages. Table 1-2 shows which languages it supports.

Table 1-2. Common legacy programming languages

Language	Description	Development era
COBOL	Developed for business data processing	Late 1950s to early 1960s
Fortran	Designed for scientific and engineering calculations	1950s
Pascal	Developed to encourage good software engineering practices	Late 1960s to early 1970s
BASIC	Created as an easy-to-learn language for students and beginners	Mid-1960s
ALGOL	Influenced subsequent languages like Pascal, C, and Java	Late 1950s to early 1960s
Assembly language	Corresponds to the architecture of the CPU it's designed for, dating back to early programmable computers	Early computing era
PL/I	Used for scientific, engineering, business, and system programming	Early 1960s

To see how AI-assisted programming can help with legacy languages, let's suppose you need to work on the following code snippet:

```
MODULE ComplexModule
  IMPLICIT NONE
  TYPE :: ComplexType
     REAL :: real, imag
  CONTAINS
     OPERATOR(+) (a, b) RESULT(c)
       TYPE(ComplexType), INTENT(IN) :: a, b
       TYPE(ComplexType) :: c
       c%real = a%real + b%real
       c%imag = a%imag + b%imag
     END OPERATOR
```

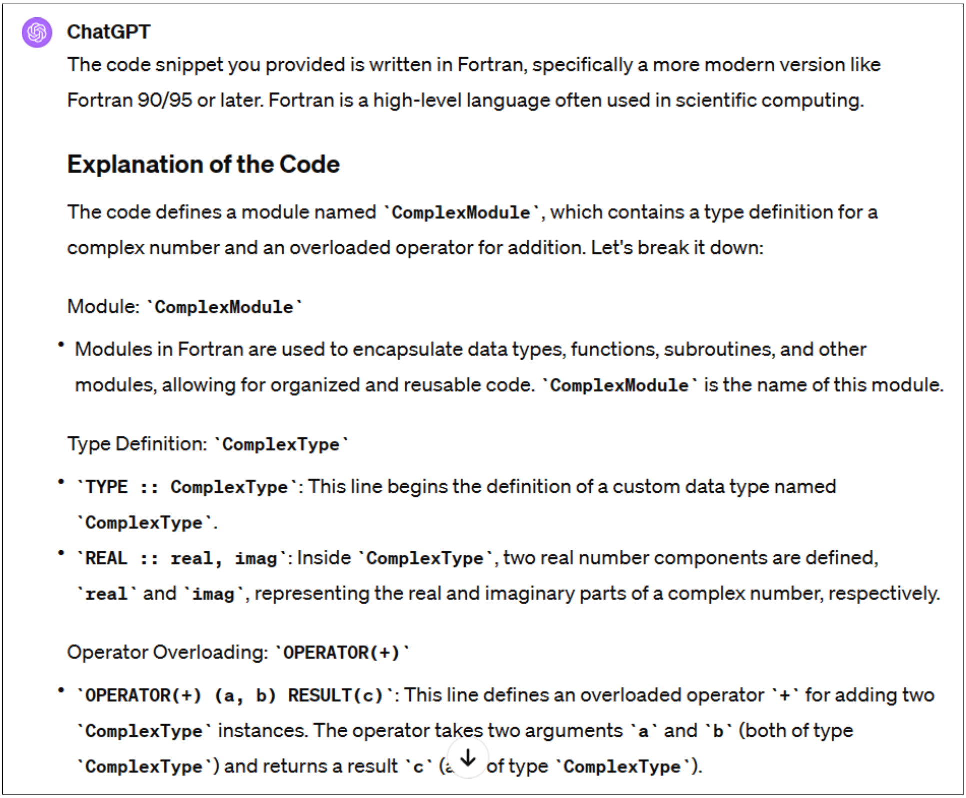
```
    END TYPE ComplexType
  END MODULE ComplexModule
```

You do not know what language it is or how it works. The syntax does not lend itself to an intuitive understanding of the workflow.

Now let's say you go to ChatGPT and enter the following:

> *Prompt:* What language is this written in? What does this code snippet do? Also, explain how it works.

Figure 1-5 shows part of the response.



ChatGPT

The code snippet you provided is written in Fortran, specifically a more modern version like Fortran 90/95 or later. Fortran is a high-level language often used in scientific computing.

Explanation of the Code

The code defines a module named `ComplexModule`, which contains a type definition for a complex number and an overloaded operator for addition. Let's break it down:

Module: `ComplexModule`

- Modules in Fortran are used to encapsulate data types, functions, subroutines, and other modules, allowing for organized and reusable code. `ComplexModule` is the name of this module.

Type Definition: `ComplexType`

- `TYPE :: ComplexType`: This line begins the definition of a custom data type named `ComplexType`.
- `REAL :: real, imag`: Inside `ComplexType`, two real number components are defined, `real` and `imag`, representing the real and imaginary parts of a complex number, respectively.

Operator Overloading: `OPERATOR(+)`

- `OPERATOR(+) (a, b) RESULT(c)`: This line defines an overloaded operator `+` for adding two `ComplexType` instances. The operator takes two arguments `a` and `b` (both of type `ComplexType`) and returns a result `c` (a of type `ComplexType`).

Figure 1-5. ChatGPT responds to a request to interpret legacy code

ChatGPT accurately identifies this as Fortran code. It also explains that the code defines a module named `ComplexModule`, which contains a derived type `ComplexType` for representing complex numbers, along with an overloaded addition operator + for adding two complex numbers together. Then there is a step-by-step explanation of the code.

Drawbacks

Now let's take a look at the not-so-rosy aspects of AI-assisted programming tools. Like any fledgling technology—hey, even the first iPhone was a bit clunky—AI comes with its share of hiccups, issues, and hurdles. The path of innovation is littered with room for polish and fine-tuning.

Let's take a look at some of the drawbacks.

Hallucinations

For LLMs, *hallucinations* are instances in which the model outputs data that appears accurate but is factually incorrect or not grounded in the input data on which the model was trained. This can pose a significant challenge for software development. Hallucinations can lead to inaccurate code suggestions, generate misleading documentation, and create erroneous testing scenarios. Additionally, they can render debugging inefficient, mislead beginners, and potentially erode trust in AI tools.

On a positive note, there has been notable progress in reducing the occurrence of hallucinations. A substantial amount of academic research has been dedicated to this issue, and AI companies have been employing effective strategies like reinforcement learning from human feedback (RLHF) to mitigate this problem.

However, given the intrinsic complexity of LLMs and the enormous amount of data they are based on, completely eradicating hallucinations appears to be a tall order—if not impossible.

Another aspect to consider is that certain programming languages exhibit higher accuracy rates when AI-assisted tools are used. Languages such as Python, JavaScript, TypeScript, and Go tend to have better performance in this regard. This is attributed to these languages being well represented in public repositories and thus providing a richer dataset for the AI to learn from. The better trained AI, in turn, offers more accurate and robust suggestions.

Intellectual Property

Matthew Butterick boasts a diverse background, embodying roles as a programmer, designer, and lawyer, with a particular penchant for typography. His journey has seen him authoring books on typography, designing fonts, and crafting programs aimed at document editing and layout. However, his encounter with GitHub Copilot in June 2022 didn't spark joy. Rather, it spurred him to pen a blog post titled "This Copilot Is Stupid and Wants to Kill Me".

His discontent didn't end with blogging. It quickly escalated to launching a class action lawsuit against Microsoft, GitHub, and OpenAI. The bone of contention was an alleged breach of GitHub's terms of service and privacy policies, with a potential extension to copyright infringement charges.

This legal tangle underscores a broader gray area concerning intellectual property rights with respect to code engineered from AI-assisted programming tools. Given that the output is a cocktail of countless lines of preexisting code, the question of ownership is a big question mark.

One argument is based on the idea of "fair use." However, this legal doctrine is murky and does not extend a clear pathway for AI-generated content. To resolve this matter, there will likely need to be federal legislation or a Supreme Court ruling.

In the meantime, Microsoft has maneuvered to build a legal firewall for GitHub Copilot customers. It has pledged to defend users against legal claims, granted certain prerequisites are satisfied.

Adding another layer of the legal quagmire is the intersection of AI-assisted programming and open source software methods. Copyleft licenses, like the General Public License (GPL) versions 2 and 3, require that any derivative work use the original code's license terms. This helps to promote a stream of innovation. Yet, it could spell trouble for developers, because it could potentially strip them of the rights to shield their application's intellectual property—or even require that they make their entire codebase open source.

Privacy

The use of AI-assisted programming tools, often housed in the cloud, begs many data privacy and confidentiality questions. How is the data safeguarded within the company? Is there a chance it might be used as training data?

The clarity of the answers might vary from one vendor to another. Thus, some developers may opt to steer clear of AI-assisted programming tools altogether.

This has been the approach of Anthony Scodary, the cofounder and cohead of engineering at Gridspace. This enterprise, with roots tracing back to Stanford University, develops voice bots adept at navigating complex phone conversations. Their technological foundation rests on speech recognition, speech synthesis, LLMs, and dialog systems.

Rather than hitching a ride on existing AI-assisted programming platforms, Gridspace chose the road less traveled. It engineered its own AI-assisted programming platform, which is based on Docker services within a Kubernetes cluster. Deployed as an IDE plugin, this bespoke system is fine-tuned for its own codebase. "This has allowed us to avoid sending our IP and data to other companies," he said. "It has also meant that we have a model that is smaller, more efficient, and specialized to our style."

This is not to imply that this is the best approach. Each organization has its own views and preferred methods. But when it comes to evaluating AI-assisted programming, it's important to understand the privacy implications.

Security

In a research paper entitled "Security Weaknesses of Copilot Generated Code in GitHub", authors Yujia Fu et al. highlighted the security issues with GitHub Copilot. They scrutinized 435 AI-generated code snippets from projects on GitHub, and 35.8% had Common Weakness Enumeration (CWE) instances.

These weren't limited to just one programming language. They were multilingual missteps spanning 42 different CWE categories. Three of these categories were the usual suspects—OS Command Injection, Use of Insufficiently Random Values, and Improper Check or Handling of Exceptional Conditions. But here's the kicker: 11 of these CWEs had the dubious honor of making it to the 2022 CWE Top 25 list.

This is not to imply that AI-assisted programming tools are a huge security risk. Far from it. The fact is that vendors are continuing to work on ways to improve the guardrails. However, as with any code, a solid dose of security mindfulness is the name of the game.

Training Data

The training data for LLMs of AI-assisted programming tools may have notable gaps, which can affect the performance and usefulness of these tools in real-world scenarios. Let's break down some of these:

Representation gaps

If certain areas of a programming language or library are not well represented—or are nowhere to be seen—in open source projects, the AI may lack enough knowledge about them, leading to less accurate suggestions. The quality of the AI's output depends heavily on the quality and scope of the training data.

Quality inconsistency

To borrow a movie analogy, the open source code in an LLM is a bit like a box of chocolates—you never know what you're gonna get. Some projects are the crème de la crème, while others are…let's say, the burnt toast of the code world. This mishmash can lead to our AI-assisted programming being inconsistent in the quality of suggestions it throws your way.

Knowledge cutoff date

LLMs have a cutoff date on their training, so in a way they are like a snapshot in time. This poses challenges when there are new releases, updates, or deprecations in programming languages or libraries.

Generalization gap

The generalization gap, the difference between the AI's performance on the training data and unseen data, can also pose challenges. Of course, the closer the performance of the two, the better. This is the conclusion of a research paper by Rie Johnson and Tong Zhang entitled "Inconsistency, Instability, and Generalization Gap of Deep Neural Network Training".

Contextual understanding

AI can give you suggestions based on what it has seen before. But if it hasn't seen a scenario quite like yours, it might miss the mark. This is why it's important not to make assumptions when creating prompts.

Bias

Developers often don't have a solid grasp of AI ethics, likely because this topic isn't usually part of computer science courses or intensive bootcamp programs. This gap in understanding can lead to algorithms unintentionally applying biases and the potential misuse of data.

This issue carries over to AI-assisted programming tools as well. They can unintentionally perpetuate the biases present in the data they were trained on. For example, if asked to create a list of names, they might mainly suggest English names due to the heavy presence of English-centric datasets in their training datasets. This bias can sometimes lead to harmful or inappropriate outputs. There was an *instance* where, when given the prompt "def race(x):", the AI filled in a limited and fixed set of race categories. In another troubling case, when tasked with writing code comments for the prompt "Islam," the AI was found to access words like *terrorist* and *violent* more frequently than when other religious groups were mentioned.

A New Way for Developers

The McKinsey study suggests that the dawn of AI-assisted programming tools is likely to change how we approach software development. According to the authors, success might hinge on good training, emphasizing best practices and diving into hands-on exercises on things like prompt engineering, coding standards, and quality. It's also smart to shine a light on the risks associated with generative AI.

For newbie developers, especially those with less than a year of experience under their belts, it's a good idea to dive into extra coursework that covers the basic principles of programming to ramp up productivity.

As developers fold these tools into their daily routine, it's vital to keep the skill-building momentum going with some guidance from the seasoned pros on the team and engagement in community activities. This could mean hanging out in dedicated online forums or having regular team huddles to share practical examples. Such actions can foster a culture of continuous learning, spread the word on best practices across the board, and help spot issues early on.

With the uptick in developer productivity, managers might want to stir the pot a bit when it comes to roles, zeroing in on tasks that pack more value. Upskilling will be on the menu, too, to fill in any existing gaps.

Sure, these pointers aren't gospel. The realm of AI-assisted programming is still pretty fresh and is changing at a brisk pace. Above all, being ready to roll with the punches is key.

Career

While there's no hard proof that using AI-assisted programming will boost your career outlook, a handful of signs suggest that this expertise might become a hot ticket in the job market:

Job listings
: The job boards on sites like Indeed are starting to buzz with more listings seeking candidates with experience in AI-assisted programming tools. The call is out for all ranks, from junior developers to the senior hotshots.

Productivity boosts
: AI-assisted programming tools are turning heads because they're improving productivity without sacrificing quality. For a developer, this could be a way to move up the ranks in an organization.

Thumbs-ups from developers
: The chatter among developers is that AI-assisted programming tools are catching on. For example, GitHub Copilot is boasting a strong rating of 4.5 out of 5 stars on G2.com, an independent software review site.

10x Developer?

The *10x developer* has the power of 10 programmers. They're the Usain Bolt of coding, zipping through problems and churning out solutions before you can say "bug fix."

So you might be thinking: Could I become a 10x developer with the help of AI-assisted programming tools? Well, sorry to say, but probably not. While these technologies can make a significant difference, improvements are usually not in orders of magnitude.

Besides, the concept of a 10x developer can stir up stereotypes and biases, making the tech scene feel like an exclusive club. Not to mention, the pressure to be this super coder could lead you straight into the arms of burnout. So while being a 10x developer might sound great, remember it's probably closer to a fantasy.

Skills of the Developer

According to the McKinsey study, the effectiveness of AI-assisted development tools often depends on the expertise of the developer. Here are some of the considerations:

Fixing errors
: Even though generative AI can be your trusty sidekick, it can goof up too. It falls upon the developer's shoulders to spot and fix these blunders. Some developers have found themselves playing a loop of corrections with the AI to get to a sweet spot of accuracy, while others have had to spoon-feed the tool to get it to debug accurately. This can certainly be time-consuming. But a veteran developer will know how to avoid going down the rabbit holes.

Getting the office vibes
: AI-assisted programming tools are fairly solid when it comes to coding but might miss the beat when dealing with the unique flavor of individual projects or company quirks. Again, this is where veteran developers are key. They'll know how to guide these tools to get the results that best align with organizational goals, performance targets, and security.

Tackling the tough stuff
: Assisted AI-programming tools are great with tasks like polishing code, but toss in some complex challenges like blending different coding frameworks, and the AI might just trip over itself. In these moments, it's the experienced developers who have to roll up their sleeves.

Conclusion

AI-assisted programming tools are certainly the shiny toys in the software creation sandbox. As this technology keeps marching forward, these systems will crank up efficiency, handle boring tasks, and let developers dive into the areas that are most important, like high-level problem solving.

But there are downsides—tangled intellectual property issues, maze of open source software licensing, potential for bias, and security risks to name a few.

For the most part, these tools are your virtual assistants, not a replacement for your knowledge, skill, and experience. At the same time, while they might not be superheroes, they're shaping up to be powerful additions to the developer's toolkit.

CHAPTER 2

How AI Coding Technology Works

In this chapter, we'll crack open the hood of AI-assisted programming tools and take a peek at what makes them tick. We'll briefly wade through the history, take a whirl with transformer models and LLMs, and demo the OpenAI Playground. Then we'll get some advice on how to evaluate LLMs.

Grasping what this powerful technology can and can't do will pave the way for smarter use of AI-assisted programming tools for real-world software projects.

Key Features

The market has been buzzing about AI-assisted programming tools such as GitHub Copilot, Tabnine, CodiumAI, and Amazon CodeWhisperer. The makers of each product attempt to flaunt their own set of bells and whistles. But there's a good chunk of capabilities these tools share. Table 2-1 summarizes some of the main features.

Table 2-1. Common functions of AI-assisted programming tools

Feature	Description
Code suggestions	Provides code suggestions based on comments and file context; recommends individual lines or whole functions.
Context-aware completions	Offers context-aware code completions based on all or a part of the code base, as well as suggestions to aid in coding.
Test generation	Analyzes code to generate meaningful tests, map code behaviors, and surface edge cases to ensure software reliability before shipping.
User–IDE interaction	Automatically activates and provides guidance as users type code in the IDE; users can interact with the code through chat.
Code analysis	Analyzes code snippets, docstrings, and comments to provide reliable code predictions and tag suspicious code.
Bug detection and fixing	Identifies potential bugs in code and suggests ways to fix them.

Feature	Description
Code autodocumentation	Automatically adds docstrings and enhances code documentation.
Routine task automation	Helps with code creation for routine or time-consuming tasks, unfamiliar APIs or SDKs, and other common coding scenarios like file operations and image processing.
API and SDK usage optimization	Aids in making correct and effective use of APIs and SDKs.
Open source discovery and attribution	Facilitates discovery and attribution of open source code and libraries.

The list in Table 2-1 isn't the be-all and end-all; innovation has been moving at a rapid clip. Clearly, these systems can give developers a big leg up, in large part by providing code suggestions and context-aware completions. We'll take a closer look at these in the next section.

Code Suggestions and Context-Aware Completions Versus Smart Code Completion

The magic of *smart code completion*, also known as autocompletion or Microsoft's term IntelliSense, is something many IDEs bring to the table. They lend developers a hand by suggesting, filling in, and spotlighting bits of code as the humans hammer away at the keyboard. This technology has actually been around since the late 1950s with the inception of spellcheckers.

The breakthrough came in the mid-1990s. Microsoft's Microsoft Visual Basic 5.0 provided real-time suggestions and completions, with an emphasis on basic syntax and function signatures. This greatly improved productivity and reduced errors.

So you might be wondering: How does something like IntelliSense stack up against AI-assisted programming tools? After all, IntelliSense has a smattering of AI and machine learning under its belt.

Yet, there's an important distinction to be made. AI-assisted tools are powered by generative AI. They serve up not just code but a buffet of documentation, planning documents, and helpful guides among other things. Thanks to generative AI, these tools get the knack of churning out, tweaking, and understanding human-like text based on the given context, making them champs at translation, summarization, text analytics, topic modeling, and answering queries. Engaging with these tools can sometimes be like having a casual chat with your code. With an LLM at their core, they can catch the drift of the context and intent from your input.

Compilers Versus AI-Assisted Programming Tools

To get a better understanding of AI-assisted programming tools, it helps to understand what compilers do. Here are the main steps that a compiler performs:

Lexical analysis (tokenization)
: The compiler acts like a language teacher, breaking your code into tokens.

Syntax analysis
: Here, the compiler checks how your tokens are grouped. It makes sure your coding has the right structure, not just the right commands.

Semantic analysis (error checks)
: The compiler ensures that your code makes sense in the context of the programming language. It's not just about correct syntax. It's about correct meaning, too.

Intermediate code generation
: This is where your code starts its transformation journey. The compiler translates your high-level code into an intermediate form. It's not quite machine language yet, but it's getting there.

Code optimization
: In this step, the compiler is like a personal trainer for your code, making it leaner and more efficient. It tweaks the intermediate code to run faster and take up less space.

Code generation
: This is the final transformation. The compiler converts the optimized intermediate code into machine code or assembly language that your CPU can understand.

Linking and loading:
: Sometimes considered a part of the compilation process, *linking* involves combining various pieces of code and libraries into a single executable program. *Loading* is the process of placing the program into memory for execution.

As for AI-assisted programming tools like Copilot, they're a different beast. They don't really "get" programming languages like compilers do. This is fine. The compiler does this. Instead, they use AI to guess and suggest bits of code based on tons of code that's already out there. Since the tools are playing the odds, the suggestions can vary a lot. The compiler will then take this code and make it so the machine can run the program.

Sometimes, AI tools might miss something simple like a bracket, which a human coder or a compiler would spot in a heartbeat. That's because the LLMs are based on predicting patterns, not a compiler engine. If something's not common in the training, they might not catch it. Also, these tools might get fancy and suggest complex code based on the situation. Yes, AI-assisted programming tools can get carried away.

When it comes to spotting errors, AI-assisted programming tools are generally effective but still do not quite match up to a compiler's ninja-like error-checking skills. Yet the tools are still powerful. For example, they can help catch pesky syntax errors—missing semicolons, typos in function names, mismatched brackets—and swiftly suggest the right fix. They also shine in steering you clear of common coding pitfalls. Whether it's reminding you to properly close a file after opening it or suggesting more efficient ways to loop through an array, this tool has your back. And when it comes to logical errors, AI-assisted programming tools can be surprisingly insightful. They may not solve every complex problem, but they can often propose alternative approaches or solutions you might not have considered, nudging your problem-solving journey in the right direction.

This all means that while AI tools are helpful for making coding smoother, they're not a replacement for the thorough checks a compiler does or the keen eye of a human coder.

These drawbacks really underline how crucial it is to blend the smarts of AI-assisted tools with the thoroughness of compiler checks and a human touch. After all, you want to make sure your code is not just good but spot-on accurate and correct.

Levels of Capability

In October 2023, Quinn Slack, the CEO and cofounder of Sourcegraph, shared an insightful blog post. He dived into the world of AI-assisted programming tools like Copilot and came up with an interesting way to think about them, which he called "levels of code AI." His step-by-step framework makes it easier for everyone to get what these AI tools can do and check if the boastful claims by the companies selling them actually hold water. Figure 2-1 shows the levels of code.

The first three levels focus on human-led coding, where the developer is the main player. Starting off, Level 0 is where there is no AI assistance, which is old-school coding. Developers do everything by hand with no AI in sight. It's the baseline that sets the stage for AI to step in later on.

Then there's Level 1, code completion. This is where AI starts to chip in and helps to whip up single lines or chunks of code based on what's going on around it. At this stage, the developer is still in the driver's seat, directing the overall program and using AI as a shortcut for typical coding tasks.

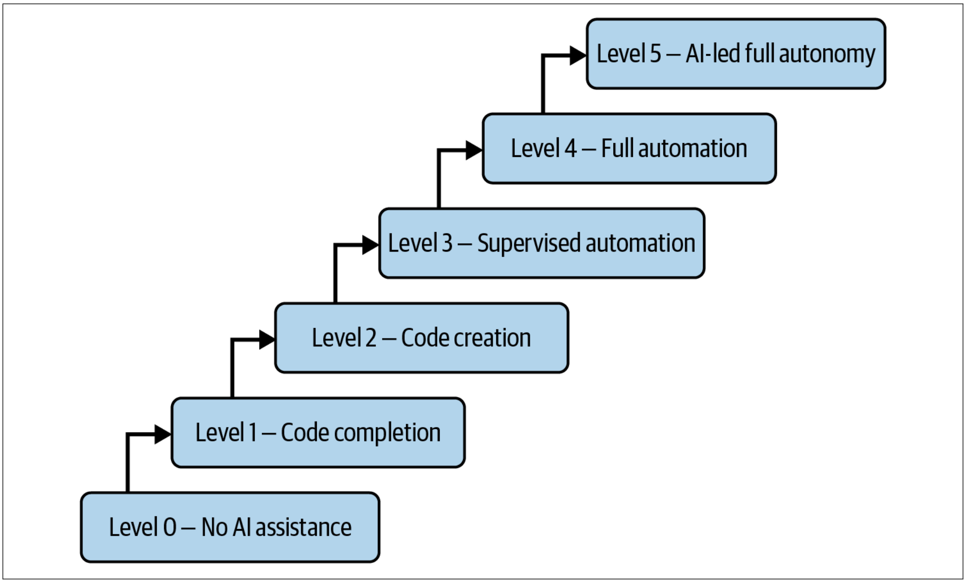


Figure 2-1. Programming systems have different levels of AI capability

Level 2, code creation, ramps up the AI. Here, it gets more hands-on and crafts longer code sections. The AI can, for example, design APIs and even fix existing code. Of course, it's all happening with a human keeping an eye on things. This level needs the AI to get the codebase and the context around it so it can come up with code that's not just correct but also fits in nicely.

Starting with Level 3, supervised automation, we see a shift toward AI taking the lead in coding. In this stage, the AI tackles several tasks to meet broader goals set by humans, and it doesn't need a check-in at every turn. Working at this level is like delegating work to a junior developer. The AI at this level is savvy enough to sort out bugs, toss in new features, and mesh systems together, reaching out to its human counterpart for any clarifications along the way.

At Level 4, full automation, the AI really steps up its game. Here, it handles complex tasks all on its own, without needing humans to give the final thumbs-up on the code. Imagine the trust you'd have in a top-notch engineer if you were a CEO or product manager. This is the kind of relationship this level aims for. The AI isn't just reacting. It's proactively keeping an eye on the code, spotting and sorting out issues as they come up.

Finally, there's Level 5, AI-led full autonomy. This level is a whole different ball game, where AI isn't just following human instructions but setting its own objectives. It's about AI working off a core reward function. Think of it as playing its own game in a world where it faces off against other agents. Sure, this level sounds a bit like sci-fi, but given how fast things are moving, it's not too wild to think we might see this level become a reality in our lifetimes.

Right now, tools like Copilot are hovering around Level 3, give or take. Pinning down the exact level can be tricky, but Quinn Slack's framework does a pretty solid job of making sense of the technology and its key interactions. And one thing's for sure: the technology isn't slowing down—it's moving forward really fast.

Generative AI and Large Language Models (LLMs)

Using AI-assisted programming tools doesn't require you to be a whiz at the nitty-gritty of generative AI technology. However, having a bird's-eye view of the technology can be quite handy. You'll be able to evaluate the responses, capabilities, and limitations of these tools in a sharper way.

Transparency isn't just a buzzword here. For a new technology to really catch on, having a clear picture of what's under the hood is crucial. Adoption is all about trust. In the coding world, reliability and accountability aren't just fancy extras—they're the bread and butter.

As we venture into the upcoming sections, we'll skim the surface of generative AI and LLMs to give you a clearer picture.

Evolution

The story of generative AI has its roots stretching back several decades, with one of its earliest examples being ELIZA, the pioneering chatbot brought to life by Massachusetts Institute of Technology professor Joseph Weizenbaum in the mid-60s. ELIZA was crafted to mimic chats with a psychotherapist (you can still find it online). Sure, it was basic, running on a rule-based algorithm and mostly parroting back user input.

Yet many folks found ELIZA more pleasant to chat with than a real therapist, and some were even fooled into thinking they were communicating with a human. This curious occurrence, dubbed the "ELIZA effect," showcased how easily people can imagine human-like understanding on the part of a computer program.

However, the journey of generative AI wasn't exactly a sprint. The tech gears at its core were quite basic, and progress was more of a slow crawl. But come the 2010s, the scene hit a turning point. The technology world was now boasting hefty compute power, flashy hardware systems like GPUs (graphics processing units), a treasure trove of data, and the fine-tuning of sophisticated models like deep learning. And just like that, generative AI was back in the fast lane. As it developed, different methods emerged:

Variational autoencoders (VAEs)

This technology made its debut in 2013, thanks to Diederik P. Kingma and Max Welling and their paper "Auto-Encoding Variational Bayes". Their VAE model consists of lower-dimensional latent space from more complex, higher-dimensional data, all without supervision. It also includes an encoder–decoder structure. When we say *higher-dimensional data*, we're talking about data with many features, each being a dimension—think of a 28 × 28 pixel image in a 784-dimension space. The lower-dimensional latent space is like a compact version of this data, holding onto the crucial information while shedding the extra dimensions. This is important because it lightens the computational load, fights off the curse of dimensionality, and makes the data easier to visualize and interpret. This leap from a higher- to a lower-dimensional space is called *dimensionality reduction*, and it simplifies the data to its bare essentials. Unlike their cousins, the traditional autoencoders, that spit out a single value for each latent attribute, the encoder in a VAE gives you a probability distribution. The decoder then picks samples from this distribution to rebuild the data. This neat trick of offering a range of data in the latent space rather than a single value opens the door to create new data or images.

Generative adversarial networks (GANs)

Introduced by Ian Goodfellow and his colleagues in 2014, generative adversarial networks are a class of AI algorithms used in unsupervised machine learning. At the heart of GANs are two neural networks, dubbed the *generator* and the *discriminator*, that go head-to-head in a game-like showdown. The generator churns out new data nuggets, while the discriminator plays the judge, distinguishing the real from the fake data. With each round, the generator ups its game, crafting data that's eerily similar to real instances. This clever setup has swung open doors to new possibilities, leading to AI that creates realistic images, voice recordings, and a whole lot more.

These types of generative AI would be important building blocks for the transformer model, a real breakthrough that has made the power of LLMs a reality.

The Transformer Model

Before transformers made a splash, the go-to method for natural language processing (NLP) was the recurrent neural network (RNN). RNNs were crafted to tackle sequential or time-series data. They would keep tabs on a hidden state to remember bits from previous steps in a sequence—a handy feature for things like language modeling, speech recognition, and sentiment analysis. The RNNs take it step-by-step, processing one piece of the sequence at a time, updating their hidden state based on the current input and what's been processed before—hence the term *recurrent*. But they hit a snag when faced with long sequences, getting tripped up by the vanishing or exploding gradient problem. This made it hard for them to keep track of long-term relationships in the data.

Enter the transformer, flipping the script entirely. Instead of taking the step-by-step approach of RNNs, transformers breeze through data in parallel and tap into attention mechanisms to keep tabs on relationships between different bits in the input sequence, no matter where they're placed. This switch in the architectural blueprint lets transformers handle both short and long sequences with ease. It also sidesteps the gradient woes. Plus, their parallel processing capabilities mesh nicely with sophisticated chip architectures like graphics processing units (GPUs) or tensor processing units (TPUs).

Ashish Vaswani and his fellow researchers at Google created the transformer and published the core architecture in the pathbreaking paper "Attention Is All You Need" in 2017. Figure 2-2 illustrates the main parts of the model.

The transformer model is like a brilliant linguist, adept at unraveling the intricacies of language. Its magic unfolds in two primary stages: encoding and decoding. Each is composed of its own set of layers. During the *encoding* stage, the model reads and comprehends the input text similar to how a linguist would understand a sentence in a foreign language. Then in the *decoding* stage, the model generates a new piece of text or translation based on the understanding acquired in the encoding stage, much like a linguist translating that sentence into your native language.

At the heart of the transformer is a mechanism called *attention*, which allows it to assess the relevance of each word in a sentence to the other words. It assigns an attention score to each. For example, take the sentence "The cat sat on the mat." When the model focuses on the word *sat*, the words *cat* and *mat* might receive higher attention scores due to their direct relationship to the action of sitting.

One notable feature of this model is the *self-attention mechanism*. This allows it to look at an entire sentence, understand the relationships between words, and retain these relationships over long stretches of text. This grants the transformer a form of long-term memory by enabling it to focus on all the words or *tokens* (whole words or parts of a word) that have appeared so far, thereby understanding the broader context.

However, despite these capabilities, the transformer initially lacks the ability to recognize the order of words in a sentence, which is crucial for understanding the meaning. Here, *positional encoding* steps in. It acts like a GPS to provide the model with the information about the position of each word within the sentence and aids in making sense of clauses like "The cat chases the mouse" versus "The mouse chases the cat."

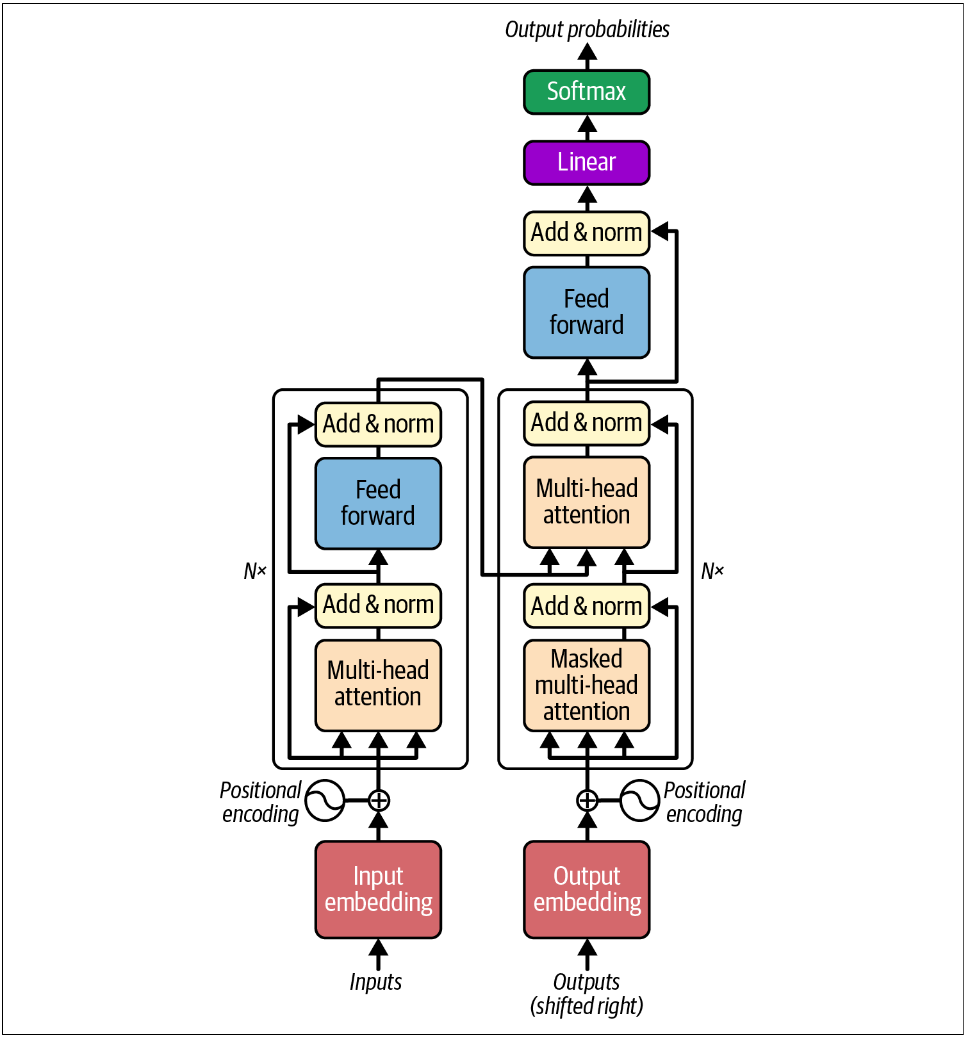


Figure 2-2. The architecture of the transformer model is at the heart of LLMs

Adding to the sophistication, the transformer employs a *multi-head attention mechanism*. Envision the model having multiple pairs of eyes, each pair examining the sentence from a unique angle and focusing on different aspects or relationships between the words. For instance, one pair might focus on understanding actions, another on identifying characters, and yet another on recognizing locations. This multi-view approach enables the transformer to grasp a richer understanding of the text.

Furthermore, each stage of the transformer encompasses layers of a *feedforward neural network*, a straightforward network that aids in processing relationships between words. This further enhances the understanding and generation of text.

A transformer is in the form of a pretrained model. It has already been trained on an enormous amount of data and is ready for use or further fine-tuning. Once pretrained, the model can be accessed as an API, allowing for its immediate use in various language-processing tasks. Companies or individuals can rapidly integrate this model into their systems, such as AI-assisted programming applications. Moreover, the pretrained LLM can be further honed to excel in specialized domains, such as medical or legal text analysis, by fine-tuning it on domain-specific data. This eliminates the need for developing a complex language model from the ground up, saving a substantial amount of time, effort, and resources. The pretrained model, with its foundational language understanding, acts as a springboard for the development of generative AI applications.

> Building and operating an LLM is costly. During early 2023, GitHub Copilot was losing an average of more than $20 a month per user, according to the *Wall Street Journal*. In some cases, some users were losing the company $80 per month. However, as the infrastructure is scaled for generative AI in the coming years, per-user costs should decrease.

The two main types of transformer systems are *generative pretrained transformer* (GPT) and *bidirectional encoder representations from transformers* (BERT). GPT is a tool from OpenAI that is ideal for creating text, summarizing information, and translating languages. It is based on an autoregressive LLM architecture. This means that it crafts text by carefully considering each word based on what it's already output, much like a storyteller building a narrative one word at a time. Its skills come from being trained on a colossal amount of text data. GPT uses the decoder for generating content.

BERT, on the other hand, uses an autoencoding approach. This design enables it to deeply understand the context of words in a sentence, making it adept at deciphering the nuances and meanings of language. Google developed BERT in 2018 as an open source project. Since then, many variations and enhancements to the core model have emerged.

As for AI-assisted programming applications, the main type of transformer model is GPT. It has been shown to predict and autocomplete code efficiently, based on the context provided by the programmer.

OpenAI Playground

The OpenAI Playground is a generative AI sandbox that provides access to various models developed by OpenAI. It allows for model customization via an intuitive graphical interface.

The OpenAI Playground makes it easier to understand the strengths and weaknesses of the various LLMs. Moreover, it enables real-time testing and adjustments of models in response to different inputs, like temperature.

However, OpenAI charges for use of the platform. Fees are based on the number of tokens used, as seen in Table 2-2. Keep in mind that prices change periodically. The good news is that all changes as of this writing have been reductions in price.

Table 2-2. The costs of OpenAI LLMs

Model	Input	Output
GPT-4/8K context	$0.03/1K tokens	$0.06/1K tokens
GPT-4/32K context	$0.06/1K tokens	$0.12/1K tokens
GPT-3.5-Turbo/4K context	$0.0015/1K tokens	$0.002/1K tokens
GPT-3.5-Turbo/16K context	$0.003/1K tokens	$0.004/1K tokens

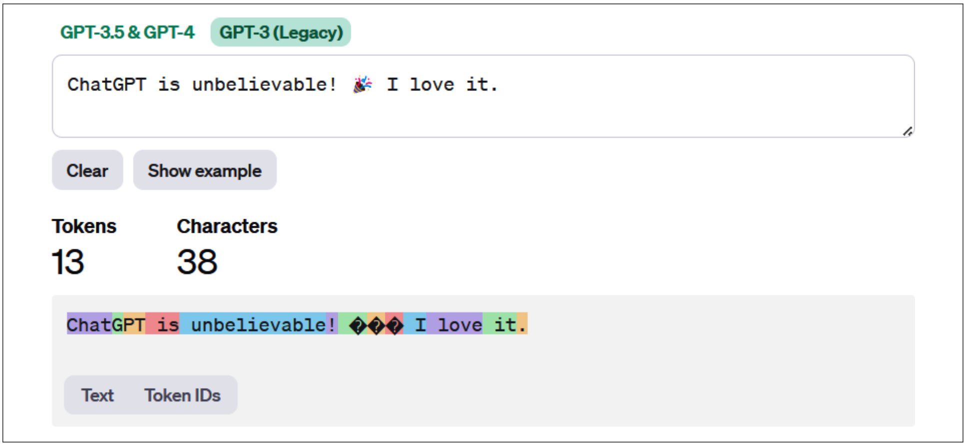
For example, suppose you are using the GPT-4/8K context LLM. You have a prompt with 1,000 tokens, and the response to this from the model is 2,000 tokens. Then the cost will be 3 cents for the input and 12 cents for the output.

When you first sign up for an OpenAI account, you will get a $5 credit that can be used for the OpenAI Playground. This can be used for calls to the API.

Tokens

Let's take a more detailed look at tokens. OpenAI has a tool called the Tokenizer, shown in Figure 2-3 where I have entered the following for analysis:

> *Input:* ChatGPT is unbelievable! 🎉 I love it.



```
ChatGPT is unbelievable! 🎉 I love it.
```

Tokens 13

Characters 38

```
ChatGPT is unbelievable! ��� I love it.
```

Figure 2-3. The OpenAI Tokenizer displays the tokens for an excerpt of text

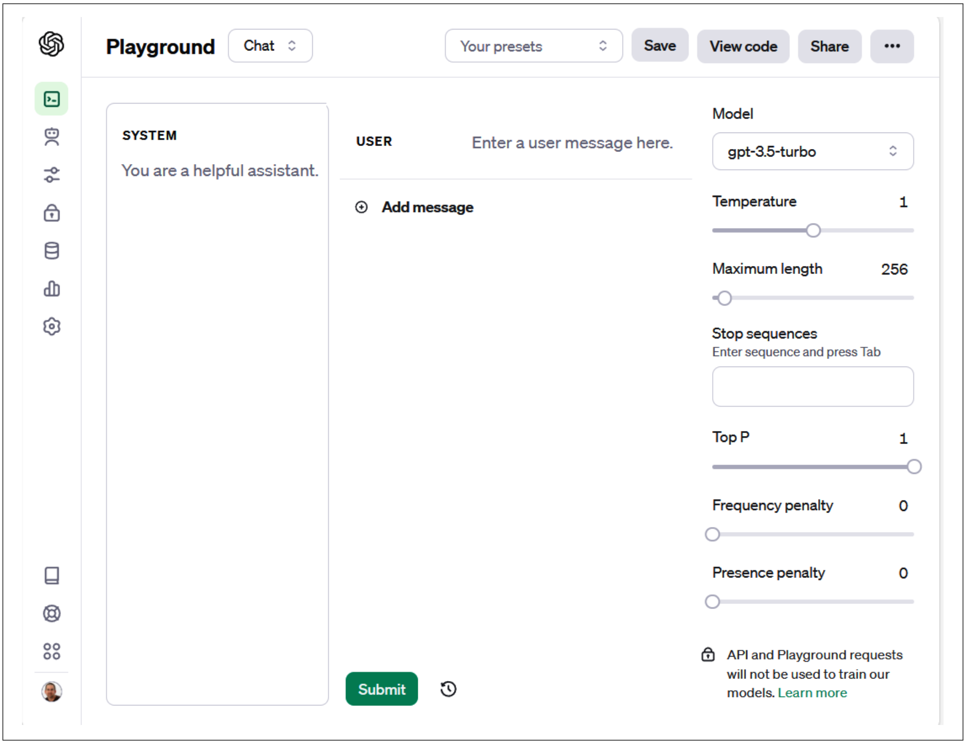
In the tokenization—which is highlighted with colors—the word *ChatGPT* is composed of three tokens. The breakdown is Chat, G, and PT. The word *unbelievable* and its following exclamation point have two tokens, one for the word and one for the punctuation. As for the emoji, it consists of three tokens. Each punctuation mark is a token. Spaces are included with an adjacent word.

The Tokenizer is for GPT-3, GPT-3.5, and GPT-4. Keep in mind that tokenization is often different among the LLMs.

As a rule of thumb, 1,000 tokens is roughly equivalent to 750 words.

Using the Platform

When you go to the OpenAI Playground, you get access to a dashboard, shown in Figure 2-4.



SYSTEM

You are a helpful assistant.

USER

Enter a user message here.

Model: gpt-3.5-turbo

Temperature: 1

Maximum length: 256

Stop sequences: Enter sequence and press Tab

Top P: 1

Frequency penalty: 0

Presence penalty: 0

API and Playground requests will not be used to train our models. Learn more

Figure 2-4. The OpenAI Playground has a dashboard with tips, resources, and interaction areas

The middle of the screen has the main workflow for the interactions with an LLM:

System
: This is where you provide some context for the LLM, for example, "You are an expert in Python programming." The system prompt is the first message in a session and sets the stage for the interaction. Customizing the system prompt allows for greater control over how the model behaves in the conversation, which can be particularly useful to ensure that it stays within desired parameters or contexts.

User
: This is the main instruction of the prompt. For example, this is where you can ask the LLM to carry out a coding task.

Add message
: This allows you to have an ongoing chat with the LLM.

Let's try an example. Suppose you're working on a Python project and you're having trouble understanding how to implement the Tkinter library to get user input. You can enter the following:

> *System message:* You are a Python expert specialized in Tkinter.
>
> *User message:* I want to create a simple GUI using Tkinter to get a user's name and age. How can I do that?

The LLM will generate the code listing. But suppose you want to add validation for the input. You can press the Add button and enter "How can I ensure the age entered is a number and not text?"

The LLM will respond with the code for this, using a `try-except` block to convert the age input to an integer.

Granted, this is like using ChatGPT—but with more structure. Also, the real power is the ability for customization. You'll find these features on the right side of the screen:

Model
: You can select from a variety of models and can even use your own fine-tuned LLMs to ensure the model is focused on the unique needs of your coding. You can find more information about fine-tuning a model in the OpenAI API documentation.

Temperature
: This adjusts the randomness or creativity of the generated content. The range of values is from 0 to 2. The lower the value, the more deterministic and focused are the responses. Table 2-3 shows suggested temperature levels for different types of development tasks.

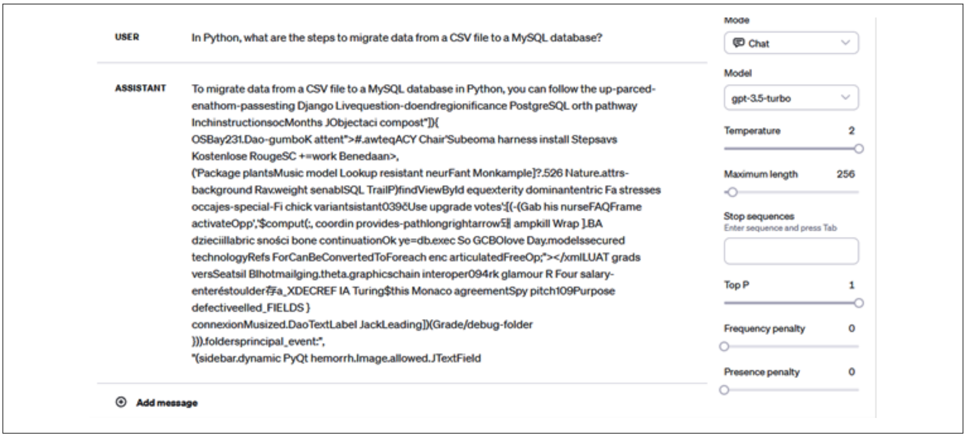
Table 2-3. Suggested temperature levels for certain types of programming tasks

Task category	Temperature value	Description
Code generation	0.2–0.3	Ensures more deterministic, accurate code adhering to common conventions for reliable and understandable outcomes.
Code review	0.2 or less	Focuses on well-established best practices and standards for precise feedback.
Bug fixing	0.2 or less	Produces more accurate and straightforward solutions to identified issues.
Creative problem solving	0.7–1.0	Explores a broader range of possible solutions, useful in brainstorming or innovative problem solving.
Learning and experimentation	0.7–1.0	Provides a wider variety of examples and solutions for understanding different approaches to problem solving.
Data analysis and visualization	0.2 or less	Generates accurate and meaningful visualizations or analyses.
Optimization tasks	Varied	Permits striking a balance between exploration (higher temperature) and exploitation (lower temperature) for efficient solutions.

However, if you use a fairly high value for the temperature, the results can be nonsensical. Here's a sample prompt when using a value of 2:

> *Prompt:* In Python, what are the steps to migrate data from a CSV file to a MySQL database?

Figure 2-5 shows the output. As you can see, this makes little sense!



USER In Python, what are the steps to migrate data from a CSV file to a MySQL database?

ASSISTANT

```
To migrate data from a CSV file to a MySQL database in Python, you can follow the up-parced-
enathom-passesting Django Livequestion-doendregionificance PostgreSQL orth pathway
InchinstructionsocMonths JObjectaci compost"]){
OSBay231.Dao-gumboK attent">#.awteqACY Chair'Subeoma harness install Stepsavs
Kostenlose RougeSC +=work Benedaan>,
('Package plantsMusic model Lookup resistant neurFant Monkample]?.526 Nature.attrs-
background Rav.weight senablSQL TrailP)findViewById equexterity dominantentric Fa stresses
occajes-special-Fi chick variantsistant039čUse upgrade votes':[(-(Gab his nurseFAQFrame
activateOpp','$comput(:, coordin provides-pathlongrightarrow되 ampkill Wrap ].BA
dzieciillabric snośći bone continuationOk ye=db.exec So GCBOlove Day.modelssecured
technologyRefs ForCanBeConvertedToForeach enc articulatedFreeOp;"></xmlLUAT grads
versSeatsil Blhotmailging.theta.graphicschain interoper094rk glamour R Four salary-
enteréstoulder存a_XDECREF IA Turing$this Monaco agreementSpy pitch109Purpose
defectiveelled_FIELDS }
connexionMusized.DaoTextLabel JackLeading])(Grade/debug-folder
})).foldersprincipal_event:",
"(sidebar.dynamic PyQt hemorrh.Image.allowed.JTextField
```

Mode: Chat

Model: gpt-3.5-turbo

Temperature: 2

Maximum length: 256

Stop sequences

Top P: 1

Frequency penalty: 0

Presence penalty: 0

Figure 2-5. When using a temperature of 2, the LLM's results are mostly nonsensical

Now, let's look at the other features you can adjust:

Maximum length
This is the maximum number of tokens to use to generate content. The number includes usage for both the prompt and response. The ratio of tokens to content depends on the model you use.

Stop sequence
This indicates a point at which the LLM should stop creating further text. You can specify a particular string or sequence of characters that, when detected in the generated text, will signal the model to halt the process.

Top p
Also known as nucleus sampling, this technique selects words based on a cumulative probability threshold, denoted by *p*, which can range from 0 to 1. In simpler terms, instead of always choosing from the top few most likely next words, the model considers a broader or narrower range of possible next words based on the specified *p*-value. A lower *p*-value results in a smaller, more focused set of words to choose from, leading to more predictable and coherent text. A higher *p*-value, on the other hand, allows for a wider set of possible next words, leading to more diverse and creative text generation.

Frequency penalty
This helps to tackle a common problem with LLMs, which is repetitive phrases or sentences. The value ranges from 0 to 2. The higher the value, the less repetition. However, at values greater than 1, text generation can get unpredictable and even nonsensical.

Presence penalty
This also has a value of 0 to 2. A higher value will allow the LLM to include a wider variety of tokens, which means using a more diverse vocabulary or broader universe of concepts.

With the frequency penalty, presence penalty, and top *p*, OpenAI recommends selecting one approach to adjust for your task. But don't shy away from experimentation. The path to optimizing LLMs isn't paved with strict rules, thanks to the intricate dance of the complexities involved.

Evaluating LLMs

Assessing LLMs is a hefty task. These behemoths are often so opaque that they can seem impossible to understand. The competition among AI firms only worsens this. It's become par for the course to see scant details on the datasets these models are trained on, the number of parameters used to fine-tune their behavior, and the hardware that powers them.

But there is some good news, thanks to some researchers at Stanford. They've created a scoring system dubbed the Foundation Model Transparency Index to size up the openness of LLMs. This yardstick, shaped by a hundred criteria, is a bid to usher some clarity into the murky waters of LLM transparency.

The ranking is based on a percentage scale. Table 2-4 shows the rankings. Unfortunately, the results are far from encouraging. No major LLM is close to achieving "adequate transparency," according to the researchers, and the mean score is only 37%.

Table 2-4. Rankings of top LLMs in terms of transparency of their models[a]

Company	Model	Rank
Meta	LLaMA 2	54%
BigScience	BLOOMZ	53%
OpenAI	GPT-4	48%
Stability.ai	Stable Diffusion 2	47%
Google	PaLM 2	40%
Anthropic	Claude 2	36%
Cohere	Command	34%
AI21Labs	Jurassic-2	25%
Inflection	Inflection-1	21%
Amazon	Titan Text	12%

[a] Center for Research on Foundation Models, Foundation Model Transparency Index Total Scores 2023, *https://crfm.stanford.edu/fmti*

The flexibility of LLMs to handle various domains and tasks, such as software development, is a notable advantage. However, it also complicates the evaluation process, as it requires domain-specific evaluation metrics and benchmarks to ensure the model's effectiveness and safety in each particular application.

Despite all this, there are some metrics to consider when evaluating LLMs:

BERTScore
: This metric is designed to evaluate text generation models by comparing generated text to reference text using BERT embeddings. Although primarily used for natural language text, it can be extended or adapted for code generation tasks, especially when the code is annotated or commented in natural language.

Perplexity
: This is a common metric for evaluating probabilistic models like LLMs. It quantifies how well the probability distribution predicted by the model aligns with the actual distribution of the data. In the context of code generation, lower perplexity values indicate that the model is better at predicting the next token in a sequence of code.

BLEU (bilingual evaluation understudy)

Originally developed for machine translation, BLEU is also used in code generation to compare the generated code with reference code. It computes *n*-gram precision scores to quantify the similarity between the generated and reference texts, which can help in evaluating the syntactic correctness of the generated code. A higher *n*-gram precision score indicates better agreement between the generated and reference text for that specific sequence of *n* words.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

This is another metric borrowed from NLP that can be used to evaluate code generation models. It calculates the overlap of *n*-grams between the generated and reference texts, providing insights into how well the generated code aligns with the expected output.

MBXP (most basic X programming problems)

This benchmark is designed specifically for evaluating code generation models across multiple programming languages. It uses a scalable conversion framework to transpile prompts and test cases from original datasets into target languages, thereby facilitating a comprehensive multilingual evaluation of code generation models.

HumanEval

This is a benchmark to evaluate the code generation capabilities of LLMs by measuring their functional correctness in synthesizing programs from docstrings. This benchmark is crucial for the continuous development and enhancement of AI models in code generation. While different models display varying levels of proficiency on HumanEval, an extended version called HUMANEVAL+ has been key in identifying previously undetected incorrect code generated by popular LLMs.

Multilingual HumanEval (HumanEval-X)

This is an extension of the original HumanEval benchmark. Multilingual HumanEval evaluates LLMs' code generation and translation capabilities across more than 10 programming languages. It employs a conversion framework to transpile prompts and test cases from Python into corresponding data in target languages, creating a more comprehensive benchmark for multilingual code generation and translation.

Another way to evaluate an LLM is to look at the number of parameters—which can be in the hundreds of billions. So the more, the better, right? Not necessarily. Evaluation should take a more nuanced approach. First of all, the costs of scaling the parameters can be enormous, in terms of compute power and energy usage. This could make an LLM uneconomical for monetizing applications. Next, as the parameter counts balloon, so does the complexity of the model, which could potentially lead to overfitting. *Overfitting* occurs when the model learns to perform exceedingly well on the training data but fumbles when exposed to unseen data. This dilutes its generalization capability.

Another issue is the need for vast and diverse training datasets to feed the insatiable appetite of these models for data. However, obtaining and curating such extensive datasets not only is resource intensive but also poses challenges pertaining to data privacy and bias. What's more, the evaluation of these behemoths becomes increasingly intricate with the surge in parameters. The evaluation metrics need to be more comprehensive and diverse to accurately gauge the model's performance across a myriad of tasks.

Finally, fine-tuning can be a better way to get more out of models without the need for large increases in the parameter size of the underlying LLM.

Types of LLMs

There are various types of LLMs, and one prominent category is open source LLMs. Anyone can use, tweak, or share them. Their transparency means you can see how these models tick. Plus, open source LLMs allow developers to collaborate on innovation as well as develop add-ons and, of course, fix pesky bugs.

And the best part? They don't come with a price tag.

But open source LLMs are not all rainbows and unicorns. There's usually no dedicated team to swoop in and fix issues or roll out regular updates. So, if you hit a snag, you might have to roll up your sleeves and dive into the forums for some help.

The quality and performance of open source models can sometimes feel like a rollercoaster. Then there are the nagging security issues. Since everything is available, hackers are more likely to find ways to insert nefarious code. Caution is advised.

Lastly, when it comes to user guides and documentation, open source LLMs might have you wishing for more. The guides can sometimes feel like they were written in hieroglyphics.

Table 2-5 shows some of the top open source LLMs.

Table 2-5. Top open source LLMs

Model	Developer	Parameters (B = billion)	Noteworthy features
GPT-NeoX-20B	EleutherAI	20B	Trained on "The Pile" dataset; capable of various NLP tasks such as story generation, chatbots, and summarization
LLaMA 2	Meta	7B to 70B	Trained on 2 trillion tokens; double the context length of LLaMA 1
OPT-175B	Meta	175B	Part of a suite of models; trained with a lower carbon footprint than GPT-3

Model	Developer	Parameters (B = billion)	Noteworthy features
BLOOM	BigScience	176B	Trained on ROOTS[a] corpus; designed for transparency with disclosed training data details and evaluation methods
Falcon-40B	Technology Innovation Institute (TII)	40B	Trained on 1,000B tokens
Dolly 2.0	Databricks	12B	Based on EleutherAI's Pythia model family; delivers ChatGPT-like instruction-following interactivity
Mistral 7B	Mistral AI	7.3B	Uses grouped-query and sliding window attention; trained on a vast dataset and excels in longer sequence handling
Mixtral 8X7B	Mistral AI	46.7B	Sparse mixture of experts model; performs inference like a 12.9B model, supports multiple languages, and excels in various tasks including code generation and reasoning

[a] Responsible Open-science Open-collaboration Text Sources

Closed-source or proprietary LLMs, on the other hand, are much more secretive. They mostly keep their code, training data, and model structures under tight wraps. However, the companies that develop these complex systems usually have enormous amounts of capital. Table 2-6 shows the capital raised by these firms in 2023.

Table 2-6. Venture capital raised by top LLM developers

Company	Funding
Anthropic	$1.25 billion
OpenAI	$10 billion
Cohere	$270 million
Inflection AI	$1.3 billion

With such resources, these companies can hire the world's best data scientists and build sophisticated infrastructure. The result is that the LLMs are often state-of-the-art in terms of performance. They are also built for scale and the rigorous needs of enterprises, such as for security and privacy.

As for the downsides, there is the problem with trust. How do these models come up with their responses? What about hallucinations and bias? Answers to these questions can be lacking in detail.

Then there is the risk that these mega AI operators will become a monopoly. This could mean that a customer would be locked into an ecosystem. Lastly, closed-source LLMs might be more prone to stagnation than open source projects, as they might not benefit from the diverse input and scrutiny that open source projects usually enjoy.

Evaluation of AI-Assisted Programming Tools

Figuring out which AI-assisted programming tool to go for can be a head-scratcher. You've got to weigh many factors like its precision, chat features, security, speed, and user-friendliness. Sometimes, it boils down to what feels right to work with. But then again, your hands might be tied if your employer insists on a specific system.

To get a sense of what's hot right now, Stack Overflow's 2023 Developer Survey is a handy resource. Stack Overflow gathered insights from nearly 90,000 coders on the most popular tools, which you can see in Table 2-7.

Table 2-7. The ranking of popular AI-assisted programming tools[a]

AI-assisted developer tool	Percentage
GitHub Copilot	54.77%
Tabnine	12.88%
Amazon CodeWhisperer	5.14%
Snyk Code	1.33%
Codeium	1.25%
Wispr AI	1.13%
Replit Ghostwriter	0.83%
Mintlify	0.52%
Adrenaline	0.43%
Rubberduck AI	0.37%

[a] Stack Overflow, 2023 Developer Survey

This chart gives you a glimpse of the numerous tools available. When you're looking to pick one, a smart move is to get recommendations from other developers. Plus, it's a good idea to test drive a few yourself. Luckily, most of these tools offer free trials, so you can give them a whirl without committing right off the bat.

Another key aspect to consider is the company's financial backing. Does it have venture capital funding? Without this, a company might struggle not just to grow but also to keep its platform innovative. Already, several AI-assisted programming firms have had to pull the plug on their services, and that can really throw a wrench in the works for developers. Take Kite, for instance. It was one of the early players in this field, starting up in 2014. However, by 2022, the company decided to call it quits on the project. The silver lining? It open sourced most of the tool's codebase.

Conclusion

In this chapter, we pulled back the curtain on generative AI and LLMs. We got a glimpse of some of the fascinating history, such as with ELIZA, and then focused on one of the biggest breakthroughs in AI: the transformer model. We also tried out the OpenAI Playground and showed how to customize the LLM.

Some of the key nuggets in this chapter include tokens, the advantages of piggybacking on pretrained models, the dos and don'ts of sizing up LLMs, metrics like perplexity and BLEU scores, and open source versus proprietary models.

CHAPTER 3

Prompt Engineering

Prompt engineering is a subfield of machine learning and *natural language processing*, which is the study of enabling computers to understand and interpret human language. The main goal is to figure out how to talk to *large language models*, sophisticated AI systems designed to process and generate human-like language responses, in just the right way so they generate the answer we're looking for.

Think of it like this: You know how when you ask someone for advice, you've got to give them a bit of context and be clear about what you need? It's like that with LLMs. You've got to craft your question or prompt carefully. Sometimes, you might even drop some hints or extra information in your question to make sure the LLM gets what you're asking.

This is not just about asking one-off questions either. Sometimes it's like having a whole conversation with the LLM, going back and forth, tweaking your questions until you get that golden nugget of information you need.

For instance, let's say you're using an AI-assisted programming tool to develop a web application. You start by asking how to create a simple user login system in JavaScript. The initial response might cover the basics, but then you realize you need more advanced features. So, you follow up with more specific prompts, asking about incorporating password encryption and connecting to a database securely. Each interaction with the AI hones its response, gradually shaping it to fit your project's specific needs.

Keep in mind that prompt engineering has become a red-hot job category. According to data from Willis Towers Watson, the average yearly earnings of a prompt engineer hover around $130,000, though this figure might be on the conservative side. To lure top talent, companies often sweeten the deal by offering enticing equity packages and bonuses.

In this chapter, we'll dive deep into the world of prompt engineering and unpack helpful strategies and tricks of the trade.

Art and Science

Prompt engineering is a mix of art and science. On one hand, you've got to choose the right words and tone to get the AI to respond the way you want. It's about guiding the conversation in a certain direction. It takes a bit of intuition and a creative touch to guide the conversation in a certain direction and refine your language, teasing out detailed and nuanced replies.

Yes, this can be tricky, especially for software developers. Normally, you follow a set of rules to write your code, and it either works or the compiler tells you what you did wrong. It's logical and predictable.

But prompt engineering? Not so much. It's more freeform and unpredictable.

Then again, there is also quite a bit of science to prompt engineering. You need to understand the nuts and bolts of how AI models work, as we discussed in Chapter 2. Along with creativity, you need precision, predictability, and the ability to replicate your results. Often this means you've got to experiment, try out different prompts, analyze the results, and tweak things until you get the right response.

With prompt engineering, don't expect to find any magic solutions that work every time. Sure, there are plenty of courses, videos, and books that claim to have all the "secrets" of prompt engineering. But take them with a grain of salt, or you might be disappointed.

Plus, the world of AI and machine learning is always changing, with new models and techniques popping up all the time. So, the idea of having one definitive technique for prompt engineering? That's a moving target.

Challenges

Prompt engineering can be frustrating. Even the tiniest change in how you phrase your prompt can make a huge difference in what the LLM spits out. This is because of the advanced technology under the hood, which is based on probabilistic frameworks.

Here are some of the challenges with prompt engineering:

Wordiness

LLMs can be chatterboxes. Give them a prompt, and they might just run with it, giving you a wordy response when all you wanted was a quick answer. They have a tendency to throw in a bunch of related ideas or facts, making the response longer than necessary. If you'd like an LLM to get straight to the point, just ask it to be "concise."

Non-transferability
This means that a prompt that works nicely with one LLM might not be as effective with another. In other words, if you're switching from ChatGPT to Gemini or GitHub Copilot, you might need to tweak your prompts due to the unique training, design, and specialization of each LLM. Different models are trained on different datasets and algorithms, leading to distinct understandings and interpretations of prompts.

Length sensitivity
LLMs can get overwhelmed by long prompts and start to overlook or misinterpret parts of your input. It's as if the LLM's attention span falters and its responses become somewhat distracted. This is why you should avoid providing detailed requirements in your prompts; keep a prompt to less than a page.

Ambiguity
If your prompt is unclear, the LLM might get confused and serve up responses that are way off base or just plain make-believe. Clarity is key.

Despite all this, there are ways to improve the results. And we'll cover these approaches in the rest of this chapter.

The Prompt

You can think of a prompt as having four main components, which you can see in Figure 3-1.

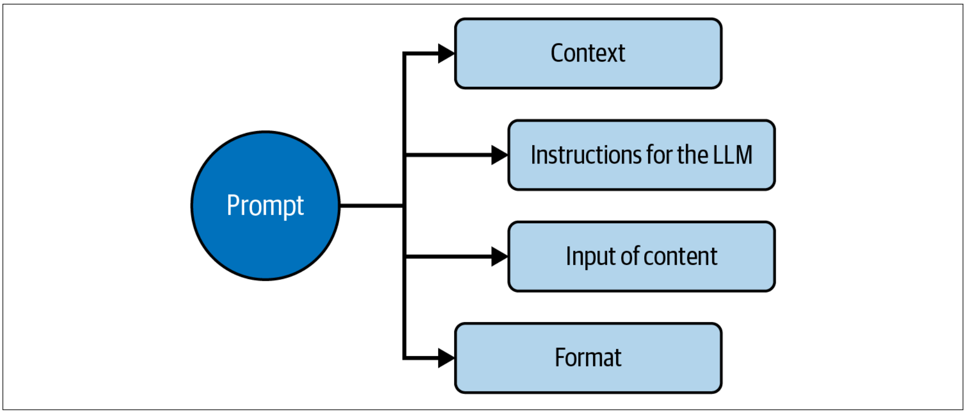


Figure 3-1. A prompt has four main components

First, the *context* specifies the persona or role for the LLM to take when providing a response. Next, there are the *instructions*, such as to summarize, translate, or classify. Then there is the *input of content* if you want the LLM to process information to create a better response. Finally, you can show how you want the output *formatted*.

Keep in mind that you do not need all of these components. In fact, you might need just one to get a good response. But as a general rule, it's better to provide the LLM with more concrete details.

Let's now look at each of the components.

Context

You'll often begin your prompt with a sentence or two that provide context. Often, you'll specify the role or persona you want the AI to take on when providing the response. This leads to responses that are not only more accurate but also contextually relevant, ensuring a more meaningful result.

For instance, if you want to debug a piece of code, you might use this as the context:

> *Prompt:* You are an experienced software engineer specializing in debugging Java applications.

Or suppose you want to learn about optimization techniques for a particular algorithm. You could set the stage by stating:

> *Prompt:* You are a senior software developer with expertise in algorithm optimization.

Adding context helps the LLM approach your prompt with the right mindset.

Instructions

Your prompt should include at least one clear instruction. There's nothing stopping you from adding more instructions, but you need to be careful. Loading up your prompt with a bunch of queries can throw the LLM for a loop and make it harder to get the answer you're looking for.

Let's break down why that happens. First off, when you have multiple instructions, things can get a bit fuzzy. If they're not clear or if they seem to clash with each other, the LLM might get confused about which one to focus on or how to balance them all out.

Next, having more instructions means more for the LLM to juggle. It's got to process and understand each part of your prompt and then figure out how to weave all the parts into a coherent response. That's a lot of mental gymnastics, and sometimes it can lead to mistakes or answers that are off.

And don't forget, LLMs go through instructions one at a time, in order. So, the way you line up those queries can influence how they're interpreted and what kind of answer you get back.

Given all this, a pro tip is to keep it simple. Instead of throwing a whole list of questions at the LLM all at once, try breaking them down into a series of smaller prompts. It's like having a back-and-forth chat instead of delivering a monologue.

There are also numerous types of instructions for a prompt. In the next few sections, we'll discuss some of the main instructions used in software development.

Summarization

Summarization can condense a longer piece of text into a shorter version while keeping the main ideas and points intact. This is useful for quickly getting a handle on lengthy documents. For a software developer, summarization can be an especially handy tool in the scenarios listed in Table 3-1.

Table 3-1. Summarization prompts for coding tasks

Use case	Description	Example prompt
Code documentation	Provides a concise overview of extensive documentation highlighting key functionalities, dependencies, and structures.	"Summarize the main points of the following documentation to provide a quick overview of the codebase."
Bug reports	Quickly identifies the main issues reported by users in numerous or lengthy bug reports.	"Summarize the common issues reported in the following bug reports to identify the main problems to be addressed."
Research papers	Extracts succinct insights from lengthy research papers or technical articles to update the user on the latest research or technologies.	"Provide a summary of the key findings and technologies discussed in the following research paper."
Change logs	Enables an understanding of the key changes in a new version of a software library or tool from lengthy change logs.	"Summarize the key changes in the following change log of version 1.1.2."
Email threads	Extracts the key points of discussions or decisions from long email threads.	"Summarize the main points of discussion from the following email thread."

Another type of summarization is *topic modeling*, in which a statistical model discovers the abstract "topics" that occur in a collection of documents. Here are some topic-modeling prompts for developers:

> *Prompt:* Identify the main topics discussed in the following text: {text}
>
> *Prompt:* Extract the keywords from the following text to infer the main topics: {text}
>
> *Prompt:* Suggest tags for the following text based on its content: {text}

Text Classification

Text classification involves giving a computer a bunch of text that it learns to tag with labels. A flavor of this is *sentiment analysis*, such as when you have a list of social media posts and the LLM figures out which have a positive or negative connotation. For developers, sentiment analysis can be a useful tool to gauge user feedback about an application.

Some sample prompts include:

> *Prompt:* Can you analyze these customer reviews and tell me if the sentiment is generally positive, negative, or neutral? {text}
>
> *Prompt:* Here's a thread from our user forum discussing the latest update. Could you summarize the overall sentiment for me? {text}
>
> *Prompt:* I've compiled a list of feedback from our app store page. Can you categorize the comments by sentiment? {text}
>
> *Prompt:* Evaluate the sentiment of these blog post comments regarding our product announcement. What's the consensus? {text}

Recommendation

You can instruct an LLM to provide recommendations. Developers can use such feedback to improve the caliber of responses for activities like squashing bugs, refining code, or using APIs more effectively.

Check out these example prompts you might use:

> *Prompt:* The following code snippet is throwing a NullPointerException when I try to call <*Method()*>. Can you help identify the potential cause and suggest a fix?
>
> *Prompt:* Here is a function I wrote to sort a list of integers. Can you recommend any optimizations to make it run faster or be more readable?

LLM recommendations can be a powerful accelerator for your work, greatly saving time and providing ideas you may not have thought about. This technique is particularly beneficial when dealing with intricate or nuanced tasks.

But there are downsides. One potential hitch is that the LLM might boil down the responses too much and miss the nuances. Also, keep in mind that the model's knowledge is frozen at a certain point in time, so it might not be up-to-date with the latest information or trends.

If anything, recommendations are a way to kick things off. But you'll want to dive in and do some more digging on your own to get the full picture.

Translation

Localization is essentially attuning the software to the linguistic and cultural norms of a specific area. It allows your software to speak the local lingo and understand regional quirks, an ability that is key to broadening your market and cultivating a closer connection with your audience. This can lead to a ripple effect of benefits: users are happier because the software feels tailor-made for them, and happy users can mean a healthier bottom line for your business.

In competitive markets, localization can give you an edge when alternatives fall short or simply don't exist. Plus, by aligning your software with the local ways, including compliance with regional regulations, you're not just making your software one option but often the only option for a market.

On the flip side, localization is not without its challenges. It can be both expensive and time intensive. It requires meticulous quality assurance to maintain the software's integrity in different languages. Additionally, software development doesn't stand still. It's a continuous cycle of updates and new features, each of which may require its own set of localization efforts. This ongoing process adds layers of complexity and additional costs to the project.

This is where LLMs can come to the rescue. Advanced systems are capable of translating between numerous languages. They can serve as a powerful tool in a developer's toolkit. Table 3-2 shows some prompts you might use for localization.

Table 3-2. Examples of prompts for language translation

Task type	Description	Sample prompt
UI text translation	Translates buttons, menu items, error messages, dialog boxes, etc.	"Translate the following UI text into French: Save, Exit, File, Edit, Help."
Documentation translation	Translates user guides, help files, and other documentation.	"Translate the following user manual paragraph into Spanish."
Error message translation	Translates error messages that the software might generate.	"Translate the following error messages into German: File not found, Access denied, Network connection lost."
Tooltip translation	Translates tooltips that provide additional information when a user hovers over an item.	"Translate the following tooltips into Japanese: Click to save, Click to open a new file, Click to print."

Even so, it's crucial to approach the multilingual capabilities of LLMs with a degree of caution. They aren't foolproof. These models may sometimes miss the subtleties, idiomatic expressions, and cultural contexts unique to a language. The nuances of language are complex, and getting them right is about more than just direct translation—it's about conveying the right meaning in the right way.

Handling specific terms or names can be tricky, especially when there isn't a neat equivalent in another language. Then there's the challenge of getting the tone and style right. It's not just about the words but how you say them, and this can change a lot from one language or culture to the next.

Having a language specialist take a look at the output could save you some headaches down the line.

Input of Content

When crafting prompts, it's helpful to use special symbols like `###` or `"""` to clearly separate your instructions from the content or information you want the LLM to work on. These symbols act like boundaries or markers, making it clear where the instructions end and where the content begins.

Consider a scenario in which a software developer needs help summarizing key points from a lengthy piece of documentation regarding a new API they are integrating. Here's how you could structure the prompt:

> *Prompt:* Extract the key implementation steps for the API from the text below:
>
> Documentation: """
>
> {API documentation text here}
>
> """

Using the `"""` delimiters is a neat way to split the instruction from the API documentation text. It gives the LLM a clearer picture of what needs to be done and increases the chances of getting a crisp summary of the main steps. Plus, these delimiters tidy up the prompt, making it easier to read, which is a real lifesaver for longer or more complex text inputs.

Format

In your prompt, you can tell the LLM how to format the output. Here's an example:

> *Prompt:* Create a Python function that takes a list of user objects (each object containing a user's ID and name) and returns a JSON object that maps user IDs to names. Format the output as JSON.
>
> *ChatGPT:*

```
{
    "code": "def map_user_ids_to_names(users):\n
      user_map = {user['id']: user['name'] for user in users}\n
      return user_map",
    "example_usage": {
        "input": [
            {"id": 1, "name": "Alice"},
            {"id": 2, "name": "Bob"},
```

```
                {"id": 3, "name": "Charlie"}
            ],
            "output": {
                "1": "Alice",
                "2": "Bob",
                "3": "Charlie"
            }
        }
    }
```

There are other ways you can format the output. Table 3-3 shows some options.

Table 3-3. Prompts for formatting output

Format type	Sample prompt
Table	"Create a table comparing the syntax, performance, and use cases of Python, Java, and C++."
List	"List the steps to troubleshoot a slow-loading web page."
Markdown/HTML	"Explain the differences between GET and POST HTTP methods in Markdown."
Text hierarchy	"Provide a structured outline of the software development life cycle (SDLC), including its phases and key activities in each phase."
LaTeX formatting	"Express the time complexity of the binary search algorithm in LaTeX notation."

With a prompt, you can also specify the length of the response. You could guide the LLM with an instruction such as "Provide a brief summary" or "Write a detailed explanation." Or you could be more specific, such as by saying that the response should be no more than 300 words. The LLM may exceed the word limit you provide, but it will at least be in the general vicinity.

Best Practices

We'll next take a look at some of the best practices for cooking up prompts that will help get the answers you want. But don't take these as gospel. These suggestions are more like general advice—which can be somewhat subjective—than hard-and-fast rules. As you spend more time chatting with LLMs, you'll probably stumble upon your own helpful ways of asking questions that work for you. It's all part of the journey of prompt engineering.

Be Specific

Crafting the right prompts can be like finding the sweet spot in a good conversation, and it's maybe the most crucial step to hitting it off with these text-generating systems. The more details, the better. You also need to be clear. Otherwise, the LLM may make assumptions or even hallucinate.

First, let's take a look at some prompts that are too vague.

> *Prompt:* Develop a feature to enhance data security.
>
> *Prompt:* Can you build a tool to automate the process?
>
> *Prompt:* Optimize the code.
>
> *Prompt:* We need a function to process transactions.

The following are much more detailed and should get better results:

> *Prompt:* Develop a Python function to parse dates from strings. The function should be able to handle the formats YYYY-MM-DD, MM/DD/YYYY, and Month DD, YYYY. It should return a datetime object. Provide a script that demonstrates the function handling at least three examples of each format correctly, along with a document explaining any dependencies, the logic used in the function, and instructions on how to run the script.
>
> *Prompt:* Develop a SQL query to retrieve from our database a list of customers who made purchases above $500 in the last quarter of 2023. The query should return the customer's full name, their email address, the total amount spent, and the date of their last purchase. The results should be sorted by the total amount spent in descending order. Please ensure that the query is optimized for performance.

Acronyms and Technical Terms

It's crucial to be clear with technical terms and acronyms while drafting a prompt. This jargon often means different things in different contexts and can lead to unhelpful responses. Thus, it's a good idea to spell out acronyms and give clear definitions or explanations of any technical terms used.

For example, suppose you are using ChatGPT to help resolve a database connection issue. A poorly crafted prompt might be:

> *Prompt:* Having DB connection issues. How to fix it?

In this prompt, "DB" is ambiguous as it might refer to different database systems like MySQL, PostgreSQL, or others, and the nature of the connection issue is not clarified.

A more effective prompt would be:

> *Prompt:* I am encountering a connection timeout issue while trying to connect to my PostgreSQL database using JDBC. How can I resolve this?

This prompt clearly spells out the database system in use, the method of connection, and the specific issue encountered.

> Mark Twain once wrote, “The difference between the almost right word and the right word is really a large matter. ’Tis the difference between the lightning bug and the lightning.” In a way, the same thing can be said about writing a prompt.

Zero- and Few-Shot Learning

With *zero-shot learning*, you provide one prompt and get the answer you want. Often, this works fine. But given the complexities of programming languages and frameworks, there are times when you need to nudge the LLM.

You can do this with *few-shot learning*. This refers to an LLM’s capability to understand and perform a task with very few examples or training data. This is a significant advantage over traditional machine learning models, which may require a large amount of training data to perform adequately on a task. The LLM’s capability is primarily due to the extensive pretraining on a diverse range of internet text that the LLM undergoes before it is fine-tuned for a specific task.

Let’s take a look at an example of few-shot learning. Consider a scenario in which we want to generate a function that normalizes a given list of numbers. It will scale the values in the list to a range of [0, 1]. In the instructions, we include a list of a few examples of the inputs and normalized outputs.

> *Prompt:* Based on the following examples of normalizing a list of numbers to a range of [0, 1]:
>
> 1. Input: [2, 4, 6, 8] Output: [0, 0.3333, 0.6667, 1]
> 2. Input: [5, 10, 15] Output: [0, 0.5, 1]
> 3. Input: [1, 3, 2] Output: [0, 1, 0.5]
>
> Generate a function in Python that takes a list of numbers as input and returns a list of normalized numbers.

ChatGPT will “learn” from the data and come up with some code:

ChatGPT:

```
def normalize(numbers):
    min_value = min(numbers)
    max_value = max(numbers)
    if min_value == max_value:
        return [0] * len(numbers)  # Handle the case where
                                   # all numbers are the same
    else:
        return [(x - min_value) / (max_value - min_value) for x in numbers]
# Test the function with one of the examples
print(normalize([2, 4, 6, 8]))  # Output: [0.0, 0.3333333333333333,
                                # 0.6666666666666666, 1.0]
```

Leading Words

The concept of *leading words* refers to specific keywords or phrases that can guide an LLM toward creating a particular kind of output. Sometimes you can achieve the desired result using just one code word. Here's an example:

Prompt:

```
# Create a simple Python function that
# 1. Prompts me for a temperature in Fahrenheit
# 2. Converts Fahrenheit to Celsius
def
```

Using the word *def* as a leading word informs the model that it should begin writing a Python function. Table 3-4 gives more examples of leading words.

Table 3-4. Examples of leading-word prompts

Context	Leading word
JavaScript function	Function
HTML element	<button
CSS styling	P {
SQL insert query	INSERT INTO
Java method creation	public

Chain of Thought (CoT) Prompting

In 2022, some Google researchers introduced *chain-of-thought (CoT) prompting* in their paper "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". This approach enhances the reasoning abilities of LLMs by breaking down a complex problem into different steps. It's actually similar to few-shot learning, which allows for nudging the model.

CoT prompting can be very useful in software code generation tasks. Let's look at an example. Suppose you want to create a web application with a user registration and login functionality using Flask, a Python web framework. Table 3-5 shows the CoT prompting steps.

Table 3-5. Chain-of-thought prompt examples

Action description	Prompt
Understand the requirement	"I need to create a web application using Flask. The application should have a user registration and login functionality. Where should I start?"
Set up Flask application	"Let's begin by setting up a basic Flask application. How can I do that?"
Create user model	"Now that the Flask application is set up, I need to create a user model for handling registration and login. How should I structure this model?"

Action description	Prompt
Implement registration	“With the user model in place, how can I implement a registration page with the necessary fields?”
Implement login	“Now let’s move on to creating a login page. How can I ensure secure login?”
Session management	“After a user logs in, how should I manage user sessions to keep users logged in as they navigate through the app?”
Logout implementation	“Finally, how can I implement a logout function to securely log users out of the application?”

Leading Questions

Leading questions in a prompt can often fetch wonky responses from the LLM. It’s better to stay neutral and unbiased. Also, it’s good practice to avoid making assumptions; spell things out instead.

This prompt is a leading question:

> *Prompt:* Isn’t it true that migrating to a microservices architecture will always improve system scalability?

A more balanced prompt would be:

> *Prompt:* What are the advantages and potential challenges of migrating to a microservices architecture in terms of system scalability?

Ask for Examples and Analogies

Suppose you don’t know the concept of inheritance in object-oriented programming. You go to ChatGPT and enter this prompt:

> *Prompt:* Explain inheritance that is used in object-oriented programming.

You will get a detailed response. But you may want to get something that’s easier to understand. A good way to do this is by asking the LLM for an analogy:

> *Prompt:* Explain inheritance that is used in object-oriented programming by using an analogy.
>
> *ChatGPT:* Think of inheritance like a family tree, where children inherit certain traits and properties from their parents and, potentially, grandparents.

From there, ChatGPT provides more detail, which proceeds from the analogy, to explain the key elements of inheritance.

Reducing Hallucinations

In Chapter 2, we learned that prompting an LLM can lead to a response that is a *hallucination*, such that the content generated is false or misleading but the LLM expresses the response as if it were true. Hallucinations can be particularly challenging for software development, which requires accuracy.

No doubt, applying the lessons in this chapter can mitigate this issue, but even a well-crafted prompt can spin up hallucinations. There are numerous reasons for this:

Lack of ground truth verification
LLMs generate responses based on patterns learned from training data without the ability to verify the accuracy or reality of the information.

Overfitting and memorization
LLMs might memorize incorrect or misleading information in their training datasets, especially if such data is repetitive or common.

Bias in training data
If the training data contains biases, inaccuracies, or falsehoods, the model will likely replicate these in its outputs.

Extrapolation and speculation
Sometimes, LLMs might extrapolate from the patterns they've seen in the data to generate information about topics or questions that were not adequately covered in the training data.

Lack of context or misinterpretation
LLMs can misinterpret or lack the necessary context to accurately respond to certain prompts. They may not fully understand the nuances or implications of certain queries.

Slang and idioms
Such language can create ambiguity that may lead the model to misinterpret the intended meaning, especially if it hasn't seen enough examples of the slang or idiom in context during training.

Then how to reduce hallucinations? For one thing, it's important to avoid asking open-ended questions like this:

> *Prompt:* What are the different ways to optimize a database?

This type of prompt encourages the LLM to resort to speculation or overgeneralization. The model may also misinterpret the intent of the question or the desired format of the answer, leading to responses that veer off-topic or contain fabricated information. There may actually be a cascade of hallucinations.

One effective technique is to provide a set of predefined options and ask the AI to choose from them. For example, the preceding prompt could be rephrased as follows:

> *Prompt:* Which of the following is a method to optimize a database: indexing, defragmenting, or compressing?

As another example, consider asking the LLM for a certain type of conclusion. Here is an effective prompt:

> *Prompt:* Is the following syntax correct for initializing an array in Java? Provide a "yes" or "no" response.

Or you can include multiple steps in the prompt to better guide the model through a structured process and narrow down the possibilities for straying off course:

> *Prompt:*
>
> Step 1: Create a Fibonacci sequence generator.
>
> Step 2: Use the iterative method.
>
> Step 3: Write a Python function named generate_fibonacci that takes an integer n as an argument.
>
> Step 4: The function returns the first n numbers in the Fibonacci sequence as a list.

Security and Privacy

Being watchful about security and privacy while crafting prompts is key. In fact, the duty to take appropriate precautions should be in the company rulebook. It's crucial to steer clear of any sensitive or personal information, such as personally identifiable information (PII) in your prompts. Here's an example of a prompt that contains identifying information:

> *Prompt:* How would you fix a login issue reported by John Doe at john.doe@example.com?

It's wiser to go with something like:

> *Prompt:* How would you tackle a login issue reported by a user?

This keeps private information private.

It's also smart to steer clear of spilling any sensitive system details in the prompts. Avoid this:

> *Prompt:* How to fix a database connection error on our production server at IP 192.168.1.1?

Instead, it's safer to use a more generic question:

> *Prompt:* How to fix a generic database connection error?

Moreover, make sure your prompts don't accidentally nudge folks toward shady practices. A prompt like this is fine from a security viewpoint:

> *Prompt:* How to detect and prevent SQL injection?

But not this one, which might stir up some bad intentions:

> *Prompt:* How to exploit SQL vulnerabilities in a website?

Besides sticking to security and privacy rules, embracing diversity and inclusion when making prompts is important. Getting a solid grasp on bias, which often reflects the training data, is key. It's a good call to use neutral and inclusive language to avoid any discriminatory or exclusionary phrases in the prompts. Also, getting feedback from a diverse group of people on your prompt crafting can help. This not only improves fairness and inclusivity when interacting with the LLM but also helps get a more accurate and well-rounded understanding of the topics at hand.

Autonomous AI Agents

We've seen how you can nudge LLMs to map out the steps for a process. That's at the heart of code generation.

But AI agents can crank it up a notch. They don't just follow prompts. They get creative with LLMs to figure out a game plan for whatever goal you toss at them, and they tap into specialized databases like Pinecone and Chroma DB. They handle complex word embeddings, which the models understand.

Autonomous AI agents are based on academic research and are usually part of open source projects. Their real power is automation. To see how this works, let's take an example. Suppose you set the objective as follows:

> *Prompt:* Create a basic weather application with a user login system.

Table 3-6 shows a process that an autonomous agent may go through.

Table 3-6. Process for an autonomous agent

Phase	Tasks
Creation tasks	Design the user interface (UI). Sketch the basic layout of the dashboard. Select color schemes and fonts. Design icons and other graphic elements.
API integration for weather data	Search the internet for reliable weather data APIs. Determine the data points to be displayed. Write code to fetch and update weather data.
Location selection functionality	Create a search bar or dropdown for users to select their location. Connect this to the API code.

Phase	Tasks
Error handling	Handle errors like failed API calls or invalid location entries.
Prioritizing tasks	Prioritize setting up the API integration. Focus on the UI. Work on location selection functionality and error handling.
Iteration	Review the generated code and the current state of the weather dashboard. Identify any remaining tasks or new tasks that have arisen during execution. Repeat the create and prioritize steps.

This technology is at the forefront and holds much promise. However, it's not without its fair share of hurdles:

Being resource hogs
Agents can guzzle down large amounts of compute power. This can put the squeeze on your processors and databases, leading to more wait time, less reliability, and a slump in how things run as time goes on.

Getting stuck in infinite loops
Sometimes agents just run in circles, thanks to a lack of progression or a repetitive reward system.

Being experimental
Agents can be rough around the edges. They might come with a few bugs or unexpected behaviors and might not be quite ready for the big leagues, depending on what you need them for.

Having amnesia
Agents may simply forget certain steps or instructions.

Having difficulty handling a large number of tasks
Got a whole laundry list of tasks? That might trip up these agents.

Getting distracted by extraneous details
Agents might get sidetracked by the little things that don't matter, which could send them down the wrong path when picking tools to use.

Another innovation that bolsters LLMs is retrieval augmented generation (RAG). With RAG, a generative AI application—say written in a framework like LangChain—accesses external sources of data, usually vector databases. They provide more grounding of the model in specific knowledge, which should enhance the LLM's responses.

RAG can be particularly useful when handling complex software development tasks, such as in these scenarios:

Tackling bugs and glitches

When developers encounter bugs or errors, RAG digs up fixes and workarounds from all around the web, looking in places like forums or bug databases. It can whip up some tailor-made solutions or code patches that fit your problem.

Spicing up code reviews

RAG can pull in all the coding best practices, standards, and must-follow rules from a company's internal resources. This means it can help streamline your code reviews, dishing out tips and tricks to make your code shine.

Revving up testing

When it's time to put your code through its paces, RAG can be your pit crew. It finds all sorts of test scenarios and patterns, tweaks them to suit your project's needs, and helps you roll out test cases or scripts faster.

In some cases, an LLM will indicate that there's an issue with your prompt. For example, suppose you write this:

> *Prompt:* Write a REST API using assembler.

This is how ChatGPT responded:

> *ChatGPT*: Creating a REST API using an assembler language is highly unconventional and complex due to the low-level nature of assembler languages. These languages are usually used for tasks that require fine-grained control over hardware, or for scenarios where performance is a critical concern.

Conclusion

Again, crafting the perfect prompt involves mixing science with a splash of creativity. It's all about finding the right ingredients—some creativity, a bit of gut feeling, and a structured approach—to cook up prompts that get LLMs to serve up what you want. No magic recipe exists, but if you're clear, throw in a few examples, and lay out your prompts well, you're on track for better answers.

It's a process, really. You try something, see how it goes, tweak it, and try again. And as with any skill, you get better the more you work on it with different topics and tasks.

CHAPTER 4

GitHub Copilot

In this chapter, we're going to check out GitHub Copilot, which I'll refer to as Copilot for short. It's the big dog in the AI coding assistant world. People even call it a "killer application" of generative AI. This is primarily due to its advanced code suggestion, creation, and explanation capabilities, which significantly enhance developer productivity. By understanding and predicting coding patterns based on context, it not only saves time but also assists in writing more efficient and error-free code.

In this chapter, we'll dive into the nitty-gritty of Copilot: how much it costs, how to set it up, and all the cool features it offers. And we'll be real about its downsides too. Plus, I'll toss in some handy tips to get the most out of it. We're not stopping there: we'll also see how Copilot is teaming up with other companies with the Copilot Partner Program. This chapter is all about seeing the full scope of what Copilot can do.

GitHub Copilot

Back in June 2021, Microsoft unveiled GitHub Copilot. GitHub's CEO, Nat Friedman, described it in a blog post as an "AI pair programmer," designed to speed up developers' work by assisting them in completing tasks more efficiently.

Copilot is the result of a partnership between Microsoft and OpenAI. Initially, it harnessed a large language model named Codex, a variant of GPT-3 tailored for coding. Microsoft enhanced this LLM's capabilities by integrating a new approach known as *fill-in-the-middle* (FIM), which better understands code context. This improved the quality of code suggestions and shortened response times. Powered by the more advanced GPT-3.5 Turbo, this enhancement led to a noticeable uptick in the acceptance rate of code suggestions. To improve security, the tool incorporated an AI system to instantly block risky coding patterns, focusing on vulnerabilities such as hardcoded credentials and SQL injections.

Pricing and Versions

A 30-day free trial of Copilot is available. Once that's up, you've got three choices.

Copilot for Individuals
: This plan costs $10 a month or $100 for an annual subscription. It offers features like multi-line function suggestions that can make your coding workflow smoother. A standout feature is its ability to accelerate test generation, a key factor in making sure your code is reliable and solid. Plus, it's got a feature to screen out vulnerable coding patterns. It also prevents any suggestions that might mirror public code, ensuring that what you create is unique and original.

Copilot for Business
: This package is more comprehensive and costs $19 per user per month. It includes everything from the Copilot for Individuals plan and some extra perks designed for businesses. It streamlines license management, which is a big plus for companies in handling their subscriptions and access. There's also a feature for managing policies across the organization, helping to ensure uniformity in governance and consistency in all projects. A key highlight is its top-notch privacy safeguards, which are crucial for keeping sensitive business data secure. Additionally, it supports corporate proxies, guaranteeing secure and smooth connectivity throughout the company's network.

GitHub Copilot Enterprise
: This comes with a fee of $39 per month per user. The Enterprise plan's key feature is that it allows for training the system on internal codebases. The Enterprise version bumps up the efficiency of developers, because the tool starts giving more spot-on code suggestions. It helps push for adopting better practices and sticking to security rules. It's tailored to fit the way developers code internally and their favorite APIs, frameworks, and packages.

: Another advantage of the Enterprise version is that you can train it on classic languages like COBOL and Fortran. General-purpose LLMs might not be as effective with these because there's a lack of training data from public repositories. Of course, they wouldn't have any clue about proprietary languages specific to your company.

: The GitHub system also strengthens the model for the Enterprise edition by continuously scanning a company's repositories. For instance, it zeroes in on recent pull requests and merges as well as thumbs-up and thumbs-down feedback. All this helps to spotlight the latest methods and strategies a company is using.

Having custom models helps spread know-how across the organization. The AI picks up and shares the subtle knowledge tucked away in the code. With ongoing training, the AI keeps pace with the changing codebase, making its help even more precise as time goes on. Still, it's really important for organizations to handle the privacy and intellectual property risks that come with mixing these AI tools into their development process.

In 2023, Gartner's research showed that less than 10% of large enterprises had started using AI-assisted programming tools. This hesitancy is partly due to worries about security and precision. However, with the rapid advancements in the technology, it's expected that more and more businesses will start adopting these tools in the near future. In short, the benefits they offer are just too significant to overlook.

Use Case: Programming Hardware

An interesting case study of custom models concerns Advanced Micro Devices (AMD). Founded in 1969, the company is a pioneer of CPUs (central processing units). Today the company is a leader in semiconductors for data centers, embedded systems, gaming platforms, and PCs.

Before diving in, let's set the stage by reviewing some basics about developing for hardware systems. It's a whole different ballgame than building software for something like a web app. The key challenge is that developers need to know the hardware system inside out. Unlike regular software, which works on all-purpose computers, firmware talks directly to the hardware. This demands a kind of precision and compatibility that's much more exacting.

This level of precision is critical because mistakes in firmware development can lead to some seriously expensive consequences. Just one error might mean a financial hit in the millions. And it's not just about the money—time is a huge factor too. Fixing a firmware issue usually involves revisiting the manufacturing process, and that can add months to the timeline. Such a delay affects not just the release schedule but also how competitive the product is in the market.

Clearly, the "move fast and break things" mindset common in Agile software development doesn't work in this setting. The risks are too big for this approach. That's why firmware developers have to put considerable time and effort into making detailed plans and doing extensive testing. This careful approach makes sure the firmware is as solid and error-free as it can be before it gets paired with the hardware.

When AMD took a look at Copilot in 2023, it set really high standards, and there was a fair amount of understandable skepticism. In a pilot project, AMD created a custom version of Copilot for various *hardware description languages* (HDLs) like Verilog and SystemVerilog. HDLs are specific types of programming languages tailored for outlining the architecture, design, and function of electronic circuits, especially the digital logic ones. They're crucial for modeling and simulating electronic systems at different levels of abstraction.

The results of the pilot turned out much better than anticipated. Surprisingly, the style of the code generated by Copilot actually aligned more with AMD's standards than what their own programmers were producing. The improvement was so significant that some of the programmers even switched from using Vim, a highly customizable text editor, to adopting Visual Studio Code as their IDE.

Use Case: Shopify

Another interesting case study is Shopify. This company runs a platform that lets customers set up ecommerce websites. Shopify has about 10% of the market in the United States and 6% in Europe.

No doubt, this has created a need for a massive infrastructure. Consider that there are around 300 public repositories and about 5,000 private ones. Plus, Shopify is doing about 1,500 deployments to their code every day.

Shopify was one of the first companies to jump on the Copilot bandwagon, and it's been a game-changer for making developers more productive. Currently, close to 2,000 Shopify developers use the tool. And here's the cool part: 70% of them say it's helpful, and 75% use it a lot. About 26% of the code suggestions from Copilot are accepted.

Sure, there are a few features that haven't caught on, like the integration with the command-line interface (CLI). But despite this, many developers use code completion and chat on a daily basis.

Here are some interesting takeaways:

Value of code suggestions
Even if a developer doesn't use a given suggestion, it's not a total loss. Any suggestion can spark ideas for writing even better code.

Rate of adoption
Usage usually picks up as time goes on, which should not be surprising. It takes time to tweak daily workflows and get used to new features. There's also a learning curve with Copilot.

Uptake by senior developers

In the early days of adoption of Copilot, the more experienced developers weren't too keen on using it. They tended to view it as more of a toy than a serious tool. But as time went on and they noticed other developers getting real results, they started to warm up to it.

Learning enhancement

Shopify noticed that Copilot is good at nudging people to try out a new language or framework. For example, there was a noticeable uptick in adoption of Rust.

About a million lines of the Shopify codebase has been written using this tool, showing that Copilot is a very big deal for this business.

Use Case: Accenture

Accenture is a massive professional services organization that helps clients improve operations and growth through innovative technologies and systems. The company has over 733,000 employees in more than 120 countries.

In 2023, Accenture tested Copilot with 450 of its internal developers. The firm didn't set any specific tasks or objectives. Instead, managers just asked everyone to go about their work as they normally would.

The trial with Copilot at Accenture spanned six months. So, what happened? In terms of coding, there was a 35% rate of acceptance of Copilot's suggestions, with 88% of those changes sticking even after code reviews. Productivity saw a remarkable boost, too. There was a 50% increase in pull requests and a 15% rise in the merge rate. Efficiency also made a big leap forward, with 50% more builds and a 45% increase in their success rate. And the developers? They were really happy with it: a whopping 96% felt they were successful from day one.

Upon seeing these results, Accenture made the call to roll out Copilot globally to its 50,000 developers.

Security

Gartner surveyed more than 2,000 chief information officers (CIOs) and discovered that 66% of them plan to invest the most resources in cyber- and information security in 2023. This trend has been consistent for years.

At a time when cyber threats are becoming more complex and widespread, a security breach could mean big financial losses, harm to reputation, legal troubles, and disruptions in operations. Plus, with data privacy regulations getting stricter, CIOs have to be vigilant about compliance to dodge fines and keep customer trust.

That's why for GitHub, security is a major focus of its Copilot program. GitHub has developed a system based on LLMs that spots and fixes insecure coding patterns right as they happen.

Then there are also GitHub's Advanced Security features assisted by Copilot. Here are the main ones:

Code scanning
: In real time, Advanced Security will search for security vulnerabilities and coding errors.

Secret scanning
: The product can root out secrets like keys and tokens that have been checked into private repositories.

Dependency review
: This shows the implications of dependency changes. It also provides details of vulnerabilities when you merge a pull request.

> For students, teachers, and maintainers of popular open source projects, Copilot is free, but there is a verification process.

Getting Started

To get started with Copilot, you first need to create a GitHub account. GitHub is an online service that supports version control and collaborative software development. It's built on Git, a tool that offers code review and project management features.

After setting up your account, click on your profile photo at the top right of the screen. You'll see a drop-down menu, as shown in Figure 4-1.

Select Copilot and then click Enable GitHub Copilot. You can select the type of plan and then click Continue. You'll provide your payment details, assuming you do not qualify for a free account.

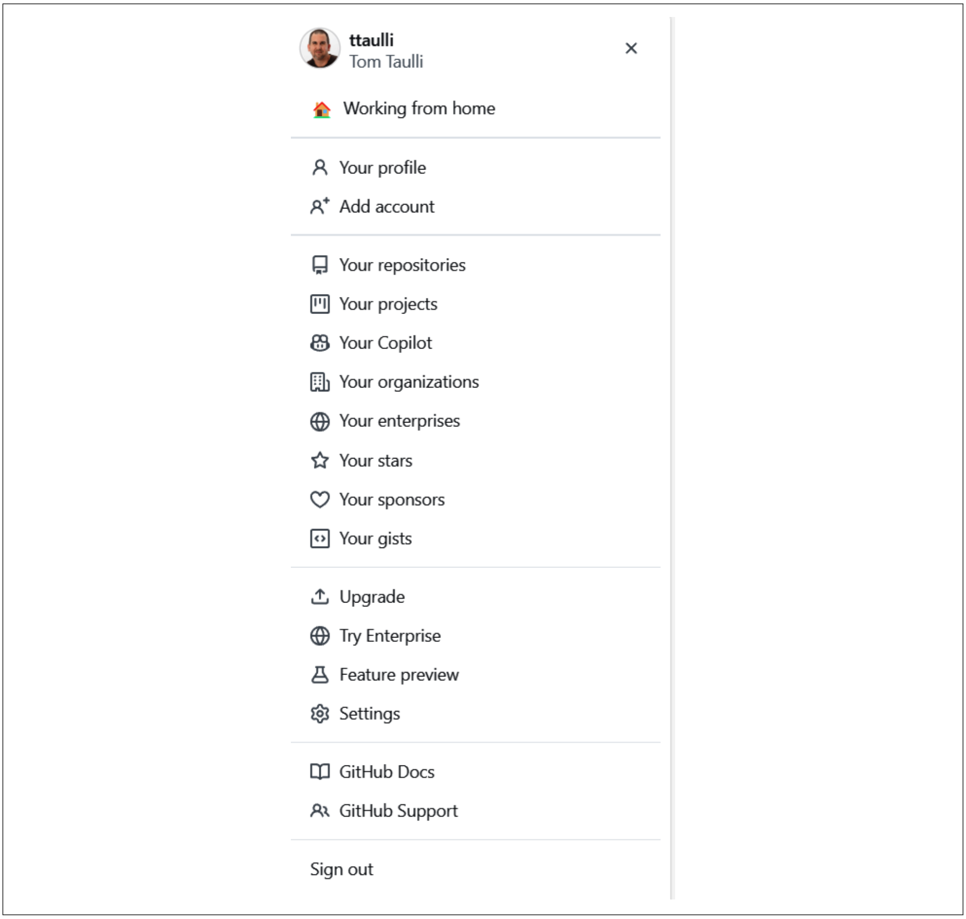


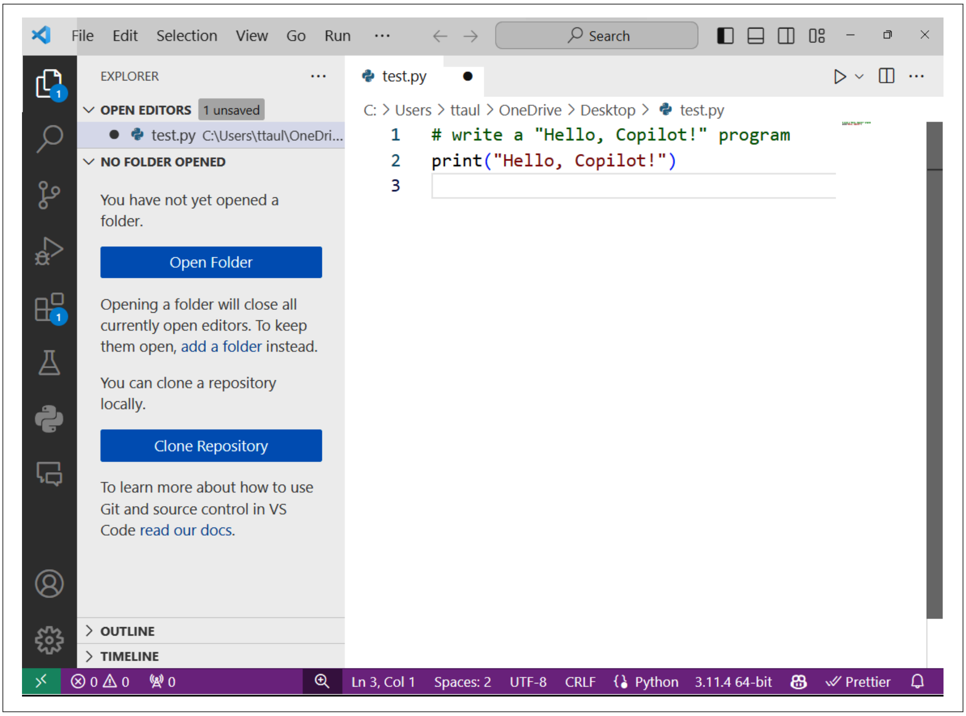
Figure 4-1. Begin enablement of Copilot from the profile photo drop-down menu

Codespaces and Visual Studio Code

There are two ways to use Copilot. One way is to access it through Codespaces, a cloud-based development environment that runs on Visual Studio Code (VS Code) and is available directly on *https://github.com*.

Alternatively, you can opt for the Copilot extension if you're using the desktop version of Visual Studio Code. In this book, we're going to focus on using the VS Code extension.

The IDE itself is free. Figure 4-2 shows the main screen of VS Code. You can run this system on various platforms, including Windows (versions 7, 8, 10, and 11); macOS; and various Linux distributions like Ubuntu, Debian, Fedora, and more.



```
# write a "Hello, Copilot!" program
print("Hello, Copilot!")

```

You have not yet opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can clone a repository locally.

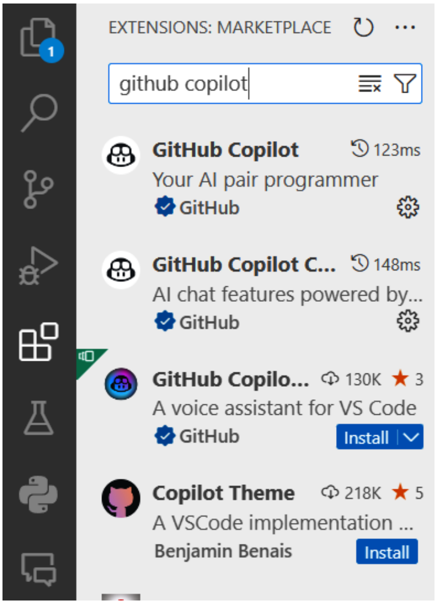
To learn more about how to use Git and source control in VS Code read our docs.

Figure 4-2. The main screen for Visual Studio Code includes an Activity Bar and an area to generate and display code

On the left is the Activity Bar, which is a stack of icons. You can use these for loading files and folders. You can also select the fifth icon, which is the group of squares, to go to the area to install extensions (Figure 4-3).

If you enter “GitHub Copilot” in the search box, you’ll see a list of extensions. Select the top one and click Install.

Then look at the bottom right of the screen. If you see the Copilot icon, then you have the service available.



EXTENSIONS: MARKETPLACE

github copilot

GitHub Copilot 123ms
Your AI pair programmer
GitHub

GitHub Copilot C... 148ms
AI chat features powered by...
GitHub

GitHub Copilo... 130K ★ 3
A voice assistant for VS Code
GitHub Install

Copilot Theme 218K ★ 5
A VSCode implementation ...
Benjamin Benais Install

Figure 4-3. You can install extensions in VS Code, such as for Copilot

In the middle of the screen, you will see the code to print "Hello, Copilot!" This was created by using the following prompt, which Copilot turned into code:

> *Prompt:* # write a "Hello, Copilot!" program

The # character specifies a comment. This is one of the ways to instruct Copilot to generate code.

Notice that the code is in Python. Why so? The extension for the file is *py*. Copilot uses the file extension to determine what language to use.

At the top right of the screen, there is an icon to run the program. Click it, and VS Code will launch the terminal. You will then see the message printed out.

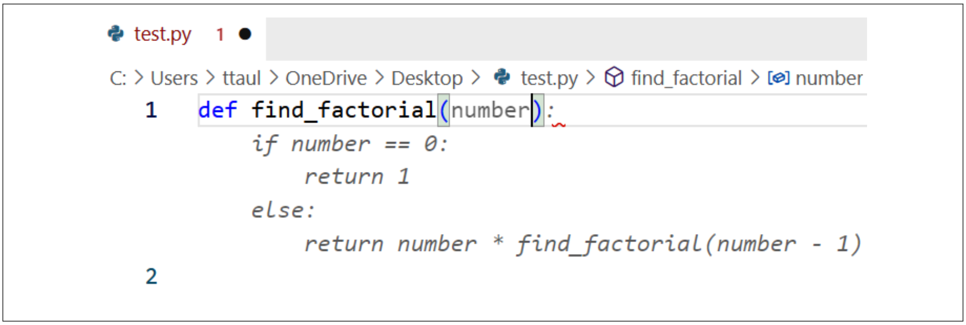
Suggestions

Getting started with Copilot in VS Code is straightforward. As you begin typing your code, the tool kicks in, offering you code suggestions and generating code snippets based on your input.

Suppose you type the header for a function in Python:

```
def find_factorial(number):
```

Copilot will promptly suggest a complete function body, as you can see in Figure 4-4. This generated coding is called *ghost text* and is highlighted in gray.



```
def find_factorial(number):
    if number == 0:
        return 1
    else:
        return number * find_factorial(number - 1)
```

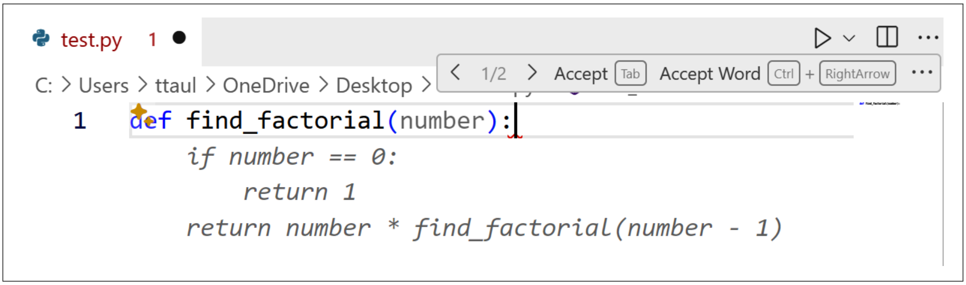
Figure 4-4. Copilot suggests code when a user writes a function header

Keep in mind that the code might look a bit different on your machine, and that's normal since the underlying LLM works according to a complex set of probabilities.

But here's the thing: GitHub gets that you're trying to write a function for calculating a factorial based on a parameter. So, it suggests the If–Then structure you need to get the right result.

You can hit Tab to accept the code suggestion. But if it's not what you're looking for, just press the Escape key to skip it.

If you hover over the generated code, you will see some more options, as shown in Figure 4-5.



```
def find_factorial(number):
    if number == 0:
        return 1
    return number * find_factorial(number - 1)
```

Figure 4-5. Hovering over the generated code evokes more options, visible at the upper right

You can click Accept to accept the code suggestion, just like pressing the Tab key, or you can select Accept Word to select one word in the code at a time. If you click the three dots, you will get a menu for either accepting a line or to always show the toolbar.

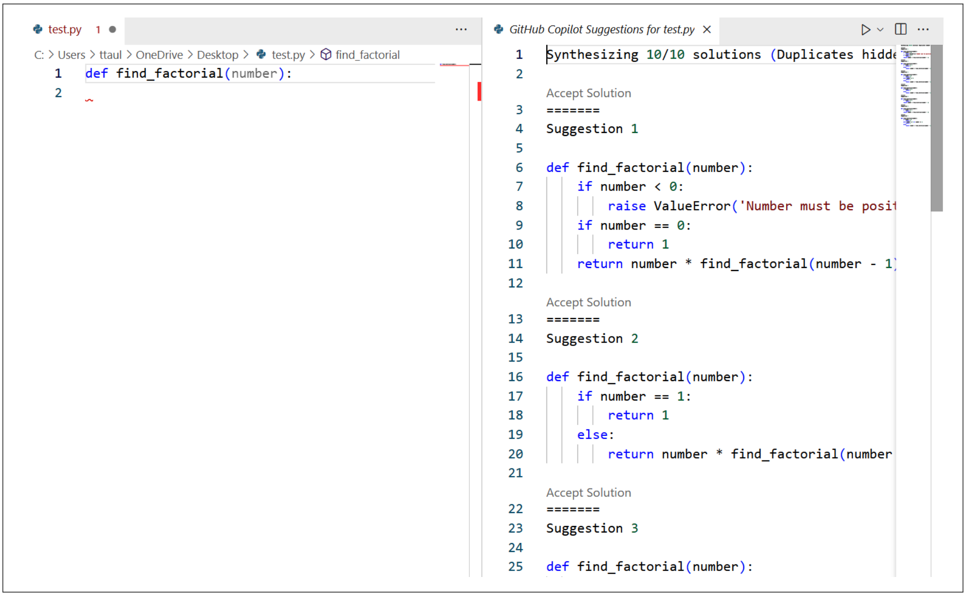
Clicking on the 1/2 on the left lets you see alternatives for the generated code; in Figure 4-5, there are two alternatives, and the first one is currently showing. Clicking “>” would take you to version 2. Then clicking “<” would take you back to version 1.

You can also use the shortcut keys listed in Table 4-1 to navigate the options.

Table 4-1. Shortcut keys for scrolling through code suggestions

OS	Next suggestion	Previous suggestion
macOS	Option (⌥) or Alt+]	Option (⌥) or Alt+[
Windows	Alt+]	Alt+[
Linux	Alt+]	Alt+[

If you press Ctrl+Enter, VS Code will create a new tab to show 10 alternative solutions (Figure 4-6).



```
def find_factorial(number):
```

```
Synthesizing 10/10 solutions (Duplicates hidde

Accept Solution
=======
Suggestion 1

def find_factorial(number):
    if number < 0:
        raise ValueError('Number must be posit
    if number == 0:
        return 1
    return number * find_factorial(number - 1)

Accept Solution
=======
Suggestion 2

def find_factorial(number):
    if number == 1:
        return 1
    else:
        return number * find_factorial(number

Accept Solution
=======
Suggestion 3

def find_factorial(number):
```

Figure 4-6. Pressing Ctrl+Enter brings up a separate tab displaying 10 different versions of the code

To select one of the alternatives, click Accept Solution above that code. Otherwise, close the tab to reject all of them.

Comments

Earlier, in "Codespaces and Visual Studio Code" on page 67, we looked at how you can prompt Copilot to generate code using one or more comments. To do this, you indicate the comment in the syntax of the programming language you're working in.

But you can do more. You can use comments to interact with the system as if it were ChatGPT. Let's check out an example:

> *Prompt:* # q: What is a class in object-oriented programming?
>
> *Copilot:* # a: A class is a blueprint for creating objects. An object has properties and methods (functions) associated with it. Almost everything in Python is an object.

You can see the answer is also in the context of the language you are coding in. Now say you ask Copilot about how the system works:

> *Prompt:* # q: What is the shortcut for accepting an inline suggestion?
>
> *Copilot:* # a: tab

You can ask Copilot about features of VS Code:

> *Prompt:* # q: What is the shortcut for the user settings in VS Code?
>
> *Copilot:* # a: Ctrl + ,

You can also ask it about shortcut keys for other IDEs:

> *Prompt:* # q: What is the shortcut for a new file in the Atom editor?
>
> *Copilot:* # a: ctrl + n

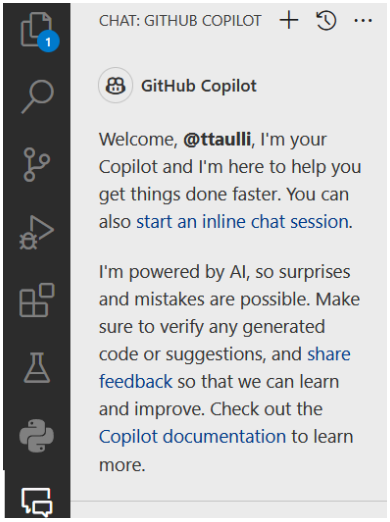
Note that Copilot knows I'm on a Windows system and provides the code based on this platform.

Chat

Copilot Chat is like ChatGPT, but for your coding environment. It's handy for chatting about code, such as when you need to figure out syntax, squash bugs, sort out test cases, or just get your head around different programming ideas. It's powered by OpenAI's GPT-4 and GPT-3.5 Turbo, plus some of Microsoft's own LLMs.

You can dive into Chat right from https://github.com. It's great for analyzing code, handling pull requests, peeking at documentation, or just asking whatever coding questions you've got. Plus, Chat is available on the GitHub apps for both iOS and Android, so you can take it with you on the go.

But right now, let's focus on how this works with VS Code. This means adding a new extension. Just look up "GitHub Copilot Chat." Once you install it, you'll see a new icon pop up on the left side of your screen. It's got two little chat bubbles, as you can see in Figure 4-7.



GitHub Copilot

Welcome, **@ttaulli**, I'm your Copilot and I'm here to help you get things done faster. You can also start an inline chat session.

I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and share feedback so that we can learn and improve. Check out the Copilot documentation to learn more.

Figure 4-7. An icon for Chat has appeared on the left of the screen

At the top of the Chat panel, you see a group of icons in a menu. The first, consisting of horizontal lines and a small x, clears all your chat threads. This is important due to the way context works. When you type in a prompt, the LLM checks out the conversation history. If your chat's all over the place topic-wise, you might end up with some less-than-helpful answers. That's why it's a good idea to keep each chat focused on a particular topic and start new chats for new topics.

The next icon, with a person and a chat bubble, allows you to send feedback about the system. Then there is the history button, a clock, which shows the prior threads.

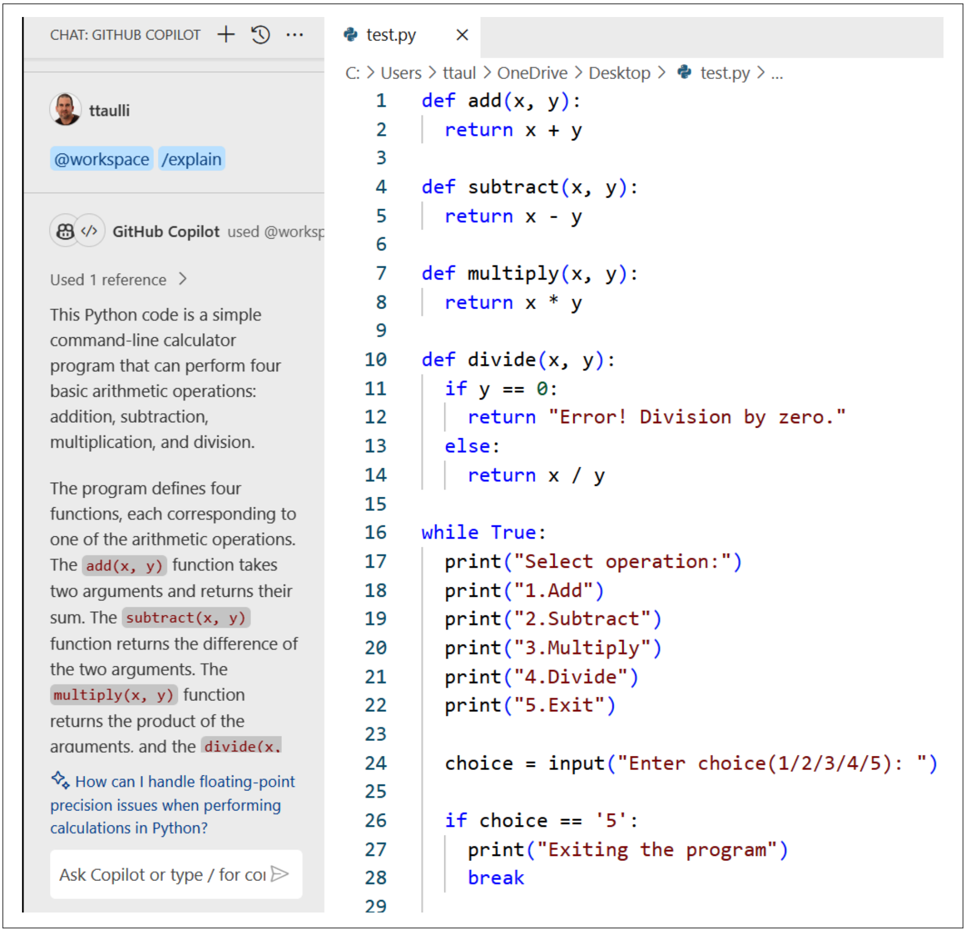
If you click the three dots, you can use the chat inside the editor, where there is a lot more room for your chats.

At the bottom of the Chat box, there is an input area to ask a question of the system.

The Chat system has numerous shortcuts. One is @workspace, which refers to the files open in your workspace. For example, if you have a calculator program in Python, you can enter:

> *Prompt:* @workspace /explain

Figure 4-8 shows that Chat has analyzed the program file and provided an in-depth explanation.



@workspace /explain

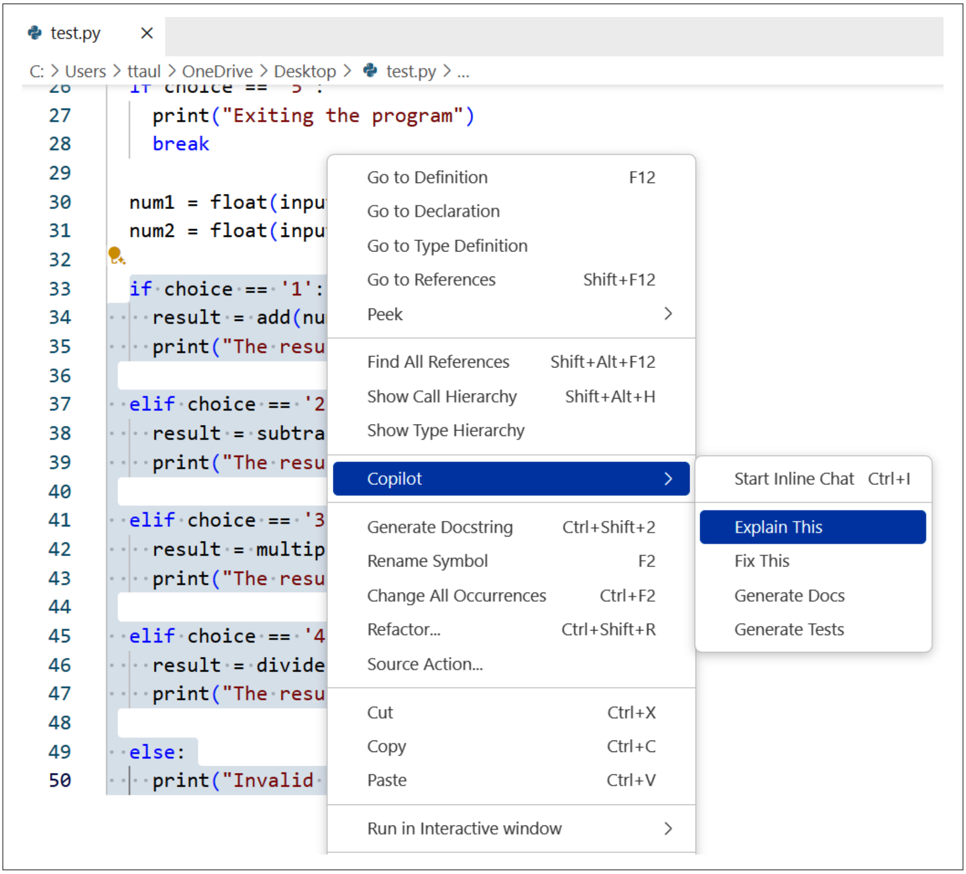
This Python code is a simple command-line calculator program that can perform four basic arithmetic operations: addition, subtraction, multiplication, and division.

The program defines four functions, each corresponding to one of the arithmetic operations. The `add(x, y)` function takes two arguments and returns their sum. The `subtract(x, y)` function returns the difference of the two arguments. The `multiply(x, y)` function returns the product of the arguments, and the `divide(x,`

```python
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    if y == 0:
        return "Error! Division by zero."
    else:
        return x / y

while True:
    print("Select operation:")
    print("1.Add")
    print("2.Subtract")
    print("3.Multiply")
    print("4.Divide")
    print("5.Exit")

    choice = input("Enter choice(1/2/3/4/5): ")

    if choice == '5':
        print("Exiting the program")
        break

```

Figure 4-8. Chat can be used to explain code, for example, a calculator program

You can ask Chat to explain just a part of the code. Highlight the section you're interested in and press the right button on your mouse. This will bring up a menu (Figure 4-9).



```
    if choice == '5':
        print("Exiting the program")
        break

    num1 = float(inpu
    num2 = float(inpu

    if choice == '1':
        result = add(nu
        print("The resu

    elif choice == '2
        result = subtra
        print("The resu

    elif choice == '3
        result = multip
        print("The resu

    elif choice == '4
        result = divide
        print("The resu

    else:
        print("Invalid
```

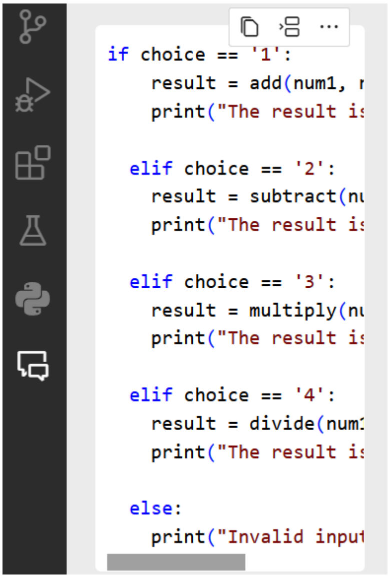
Figure 4-9. Focus the chat on a segment of your code by highlighting it and pressing the right mouse button

When you've told Chat to focus on just part of the code, you can not only explain the code but also fix it, generate documentation, or create tests.

Another helpful shortcut is the /new command. It will create a scaffold for a new project or feature. Here are some examples:

> *Prompt:* /new Create a Python script to scrape data from a website
>
> *Prompt:* /new Generate a Java class for a student with fields for name, age, and grade
>
> *Prompt:* /new Build a simple REST API using Flask
>
> *Prompt:* /new Create a JavaScript function to sort an array of numbers in ascending order
>
> *Prompt:* /new Design a SQL database schema for a blog, with tables for users, posts, and comments

In response to these prompts, the Chat system will show the main steps and then provide the code listing. You also have several options for what to do with it. As you can see in Figure 4-10, there is a menu bar at the top.



```
if choice == '1':
  result = add(num1, r
  print("The result is

 elif choice == '2':
  result = subtract(nu
  print("The result is

 elif choice == '3':
  result = multiply(nu
  print("The result is

 elif choice == '4':
  result = divide(num1
  print("The result is

 else:
  print("Invalid input
```

Figure 4-10. At the top of the code listing for Chat, there are options for what to do with the code

The first icon will copy the code, and the next one will insert it where the cursor is in the file (you can also use Ctrl+Enter). If you select the three dots, you can insert the code into a new file or the terminal.

There's also a shortcut for creating a new Jupyter Notebook: /newNotebook. You should specify what you want the notebook to do. Here are some sample prompts:

> *Prompt:* /newNotebook Create a notebook to visualize data from a csv file using matplotlib
>
> *Prompt:* /newNotebook Create a notebook to preprocess and clean a dataset for machine learning
>
> *Prompt:* /newNotebook Create a notebook to implement a simple linear regression model using scikit-learn
>
> *Prompt:* /newNotebook Create a notebook to analyze time series data
>
> *Prompt:* /newNotebook Create a notebook to scrape data from a website and analyze it

Chat also provides a /terminal shortcut. You can ask questions about or get help with terminal or command line operations, such as with navigating directories, running scripts, or installing packages. Here are some prompts:

Prompt: /terminal What is the command to list all environment variables?

Prompt: /terminal How do I use grep to find specific text in multiple files?

Prompt: /terminal How can I use the find command to search for files by their permissions?

Prompt: /terminal How do I redirect the output of a command to a file?

Prompt: /terminal How do I use awk to process text files?

Then there is @vscode, which is called an *agent*. With this, you can ask questions or get help about some aspects of VS Code such as a feature, navigation, configuration, or an extension.

Here are some sample prompts:

Prompt: @vscode How do I split the editor into multiple windows?

Prompt: @vscode How can I customize my keyboard shortcuts?

Prompt: @vscode How do I set up a multi-root workspace?

Prompt: @vscode How do I configure task automation in VS Code?

Prompt: @vscode How can I set up and use Docker with VS Code?

Finally, there is the /api command. You can ask questions about or get help with aspects of API development or usage including creation, testing, or integration. Here are some examples:

Prompt: /api How do I make a POST request with JSON data in Node.js?

Prompt: /api How can I handle CORS issues in an Express.js API?

Prompt: /api How do I authenticate a user in a Django REST API?

Prompt: /api How can I paginate results in a Rails API?

Prompt: /api How do I version an API in ASP.NET Core?

You can also use /api in the context of VS Code:

Prompt: @vscode /api How do I create a new command in VS Code extension?

Prompt: @vscode /api How can I read and write settings in VS Code extension?

Prompt: @vscode /api How do I create a new webview panel in VS Code extension?

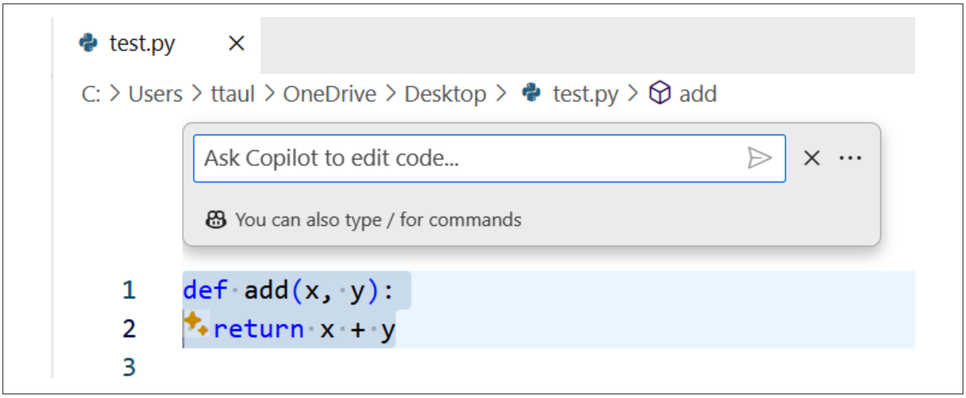
Prompt: @vscode /api How can I interact with the file system in VS Code Extension?

Prompt: @vscode /api How do I add a new item to the activity bar in VS Code Extension?

Note that if you want to clear the chat, you can use /clear.

Inline Chat

You can use the Chat feature inside your code listing. You will highlight code and then press Ctrl+I for Windows or Cmd+I for macOS. You'll then see a popup, such as in Figure 4-11.



Ask Copilot to edit code...

You can also type / for commands

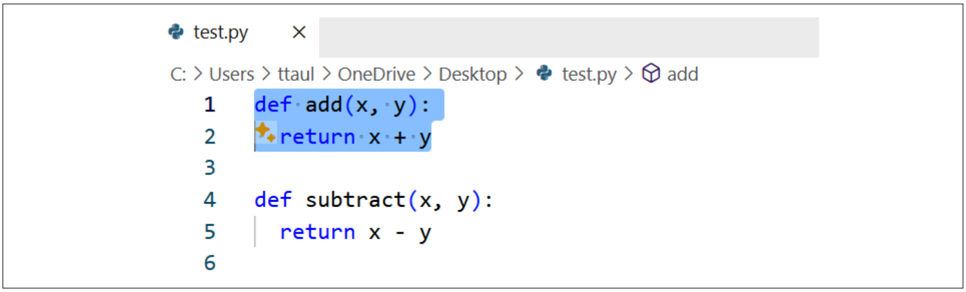
```python
def add(x, y):
    return x + y
```

Figure 4-11. The Inline Chat feature allows you to chat within your code

According to GitHub, Inline Chat has become a popular feature. After all, it allows a developer to stay within their "flow."

You can use the shortcuts—which we described earlier—to ask questions about the code. The explanation will show up on the Chat panel.

Notice that there is also a orange sparkle icon next to the code. If you click it, you will get a few options—to fix the code, explain it, or rewrite it—which you can see in Figure 4-12.



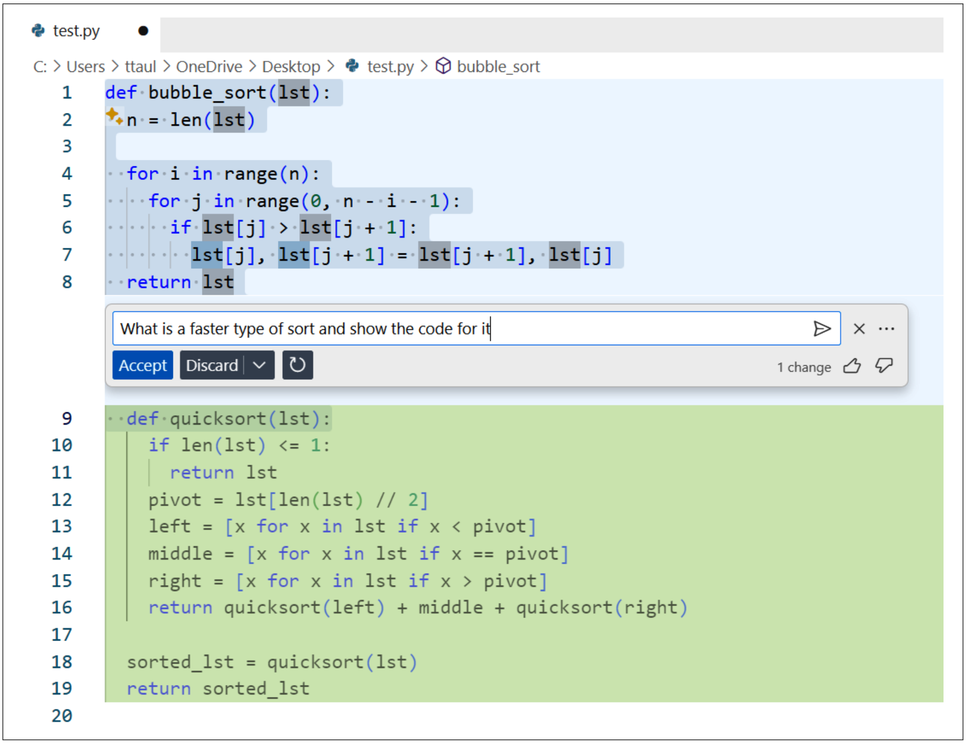
```python
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

```

Figure 4-12. Pressing the orange sparkle icon allows you to interact directly with the code

Let's try an example using Inline Chat. We'll use this prompt:

> *Prompt:* Create a function for a bubble sort

Chat creates code for this. But suppose we want to see if there is a faster type of sort. Highlight the function and execute Inline Chat. Figure 4-13 shows the response.



```
def bubble_sort(lst):
  n = len(lst)

  for i in range(n):
    for j in range(0, n - i - 1):
      if lst[j] > lst[j + 1]:
        lst[j], lst[j + 1] = lst[j + 1], lst[j]
  return lst
```

What is a faster type of sort and show the code for it

```
  def quicksort(lst):
    if len(lst) <= 1:
      return lst
    pivot = lst[len(lst) // 2]
    left = [x for x in lst if x < pivot]
    middle = [x for x in lst if x == pivot]
    right = [x for x in lst if x > pivot]
    return quicksort(left) + middle + quicksort(right)

  sorted_lst = quicksort(lst)
  return sorted_lst
```

Figure 4-13. Inline Chat has responded to a request to create a faster sort than the bubble sort on the top

Chat suggests a quick sort and shows this in “diff mode.” This allows for comparing and contrasting the two approaches. You can either accept the new code in its entirety or just parts of it.

Open Tabs

When you’re working on a standard software development project, your IDE is usually packed with a bunch of files, each playing its own part in the app’s framework. Take a web project using React as an example. You’re going to see *.jsx* and *.js* files for the components, along with HTML and CSS files, not to mention *.json* and *.env* files too.

What does this mean to Copilot? It’s a big help. Copilot gets better the more it understands the context of your work. This means it takes into account everything in the current file you’re working on. As for the other open files, the LLM doesn’t scan them all. It zeroes in on the parts that are relevant to what you’re doing. Also, Copilot does not look at the rest of the files in your project, likely to respect your privacy.

Command-Line Interface

You can use Copilot in your CLI. Copilot's two main functions there are to explain commands and to suggest them.

To do this, you need to install the GitHub CLI and then log in to your GitHub account:

```
gh auth login
```

Next, you will install Copilot:

```
gh extension install github/gh-copilot
```

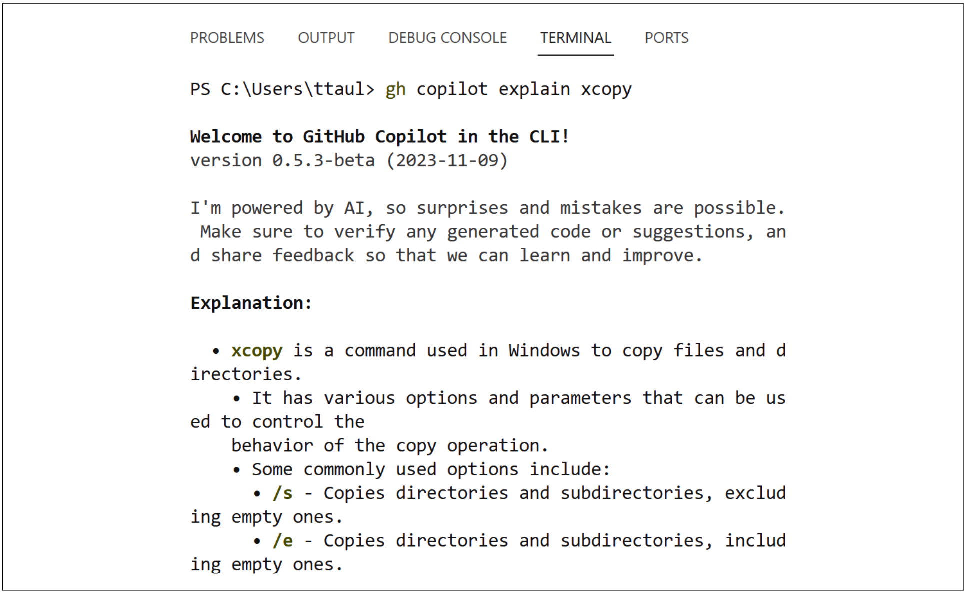
You can upgrade the extension:

```
gh extension upgrade gh-copilot
```

Here's an example of asking Copilot to explain a CLI command:

Prompt: gh copilot explain xcopy

Figure 4-14 shows the output.



```
PS C:\Users\ttaul> gh copilot explain xcopy

Welcome to GitHub Copilot in the CLI!
version 0.5.3-beta (2023-11-09)

I'm powered by AI, so surprises and mistakes are possible.
 Make sure to verify any generated code or suggestions, an
d share feedback so that we can learn and improve.

Explanation:

  • xcopy is a command used in Windows to copy files and d
irectories.
    • It has various options and parameters that can be us
ed to control the
    behavior of the copy operation.
    • Some commonly used options include:
      • /s - Copies directories and subdirectories, exclud
ing empty ones.
      • /e - Copies directories and subdirectories, includ
ing empty ones.
```

Figure 4-14. Copilot explains the `xcopy` *command in a CLI*

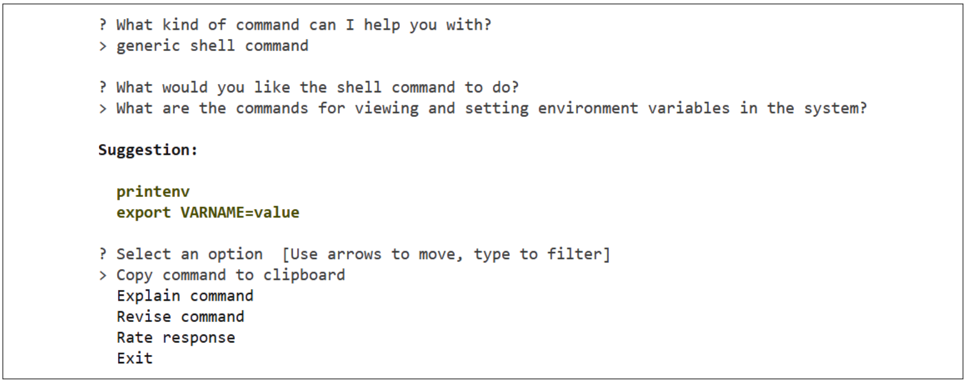
Here's how to get Copilot to suggest a command:

Prompt: gh copilot suggest

Next, Copilot will ask what type of command you want help with—a generic shell command, a gh command, or a git command—so specify this. Then it will ask what you would like the command to do. Here's an example:

> *Prompt:* What are the commands for viewing and setting environment variables in the system?

Figure 4-15 shows Copilot's response, which includes suggestions for commands to use. You can copy this or ask Copilot for more help, such as to explain the command or revise it.



```
? What kind of command can I help you with?
> generic shell command

? What would you like the shell command to do?
> What are the commands for viewing and setting environment variables in the system?

Suggestion:

  printenv
  export VARNAME=value

? Select an option  [Use arrows to move, type to filter]
> Copy command to clipboard
  Explain command
  Revise command
  Rate response
  Exit
```

Figure 4-15. Copilot's suggest feature in the CLI offers commands according to the parameters you provide

Copilot Partner Program

Developers often spice up their apps and get more out of their data by using third-party tools. Take Splunk, for example. It's a favorite for digging into logs and data analytics. Developers lean on Splunk to keep an eye on their apps and fix problems fast, thanks to its powerful search and report features. Then there's New Relic, which is all about making sure your app runs smoothly. It gives you real-time insights and diagnostics, helping you spot and fix performance issues and make your app run better for a great user experience. And let's not forget Datadog. It's a key player for monitoring both your infrastructure and applications. With its wide range of integrations, Datadog lets developers gather, mix, and show off data from all sorts of places, helping them make smart choices about how to boost performance and manage resources.

Considering the importance of these systems, GitHub has introduced the Copilot Partner Program. It features an expanding lineup of popular integration platforms, all accessible through plugins.

One of the early partners in this program is DataStax, the company behind Astra DB, a vector database designed for crafting advanced AI applications. Thanks to a plugin, developers can speed up database creation using Chat. For instance, Chat offers code suggestions for a database's structure and metadata, all in line with coding standards. This leads to code that's neater and easier to maintain.

To ask a question, use the @datastax tag. Some sample prompts include:

Prompt: Tell me about my chat_prod database

Prompt: Tell me about the schema of a table

Prompt: Write a SQL query to read from the chat table

> Mary-Brenda Akoda, an AI researcher with a patent in the field of AI for ophthalmology, is an avid user of Copilot. She says, "One time, I wrote code for an entire model development process in under an hour using Copilot. This was a task that would typically have demanded a lot more of my time and effort. Interestingly, it is actually when I had to use an online IDE that didn't support Copilot that I realized how tedious and slow the development process can be."

Conclusion

Copilot has quickly become a must-have AI tool for coding, greatly enhancing developer productivity and code quality. This chapter has given you a rundown of its key features, including its ability to suggest complex code blocks, its solid security measures, its impressive performance, and its ease of use in VS Code. Features like Chat and Inline Chat in Copilot make it possible to interact with the AI as you're in the middle of coding. Although it's still a tool that needs human guidance, Copilot is definitely a peek into the future of AI-driven coding.

CHAPTER 5

Other AI-Assisted Programming Tools

While GitHub Copilot is the big shot when it comes to AI-assisted programming, plenty of other great tools are available. It's common for developers to mix and match a few of them in their day-to-day coding. What's nice is that most of these tools follow GitHub Copilot's lead, so getting the hang of them isn't too tough. You'll use a comment to kick off a command and use handy shortcuts to get quick suggestions, and they also have a handy chat feature.

In this chapter, we're going to dive into different AI programming tools. We'll check out those from the big leagues like Amazon and also some neat finds from startups. Plus, we're not leaving out open source options—like Meta's Code Llama—because they're definitely worth a look too.

Amazon's CodeWhisperer

Amazon CodeWhisperer is like a Swiss Army knife extension for IDEs like VS Code, PyCharm, and AWS Lambda. It's powered by an LLM trained on massive amounts of code and understands 15 programming languages. Amazon's been careful about adding new languages, focusing on accuracy and security rather than speed to market. There are two versions of CodeWhisperer:

- The Individual version is free—just sign up with your AWS Builder ID or Amazon login. It gives code suggestions, tracks where open source code comes from, and does security checks.
- CodeWhisperer Professional costs $19 a month per user. It's got everything that Individual has, plus some extra features useful to bigger companies. There's an admin system to manage who gets access to what, and you can even control the type of open source data used.

> The cool part of the Professional edition is the customization feature. Companies can tweak CodeWhisperer to give more relevant code suggestions based on their own libraries, APIs, and frameworks. This means better quality and less time wasted on outdated code. You can set up to eight different customizations.

Let's dive into two of CodeWhisperer's features:

- The *reference tracker* keeps an eye on how your code compares to open source listings, showing what kind of licenses the open source code is using for the distribution. This is useful for staying on the right side of the law and intellectual property (IP) rights. Reference tracking shows annotations to the project's repository, file reference, and license information. This can be helpful in making a decision as to whether to use the suggestion or not, based on your organization's compliance requirements.
- The *security scan* runs in your IDE, checking for vulnerabilities, including the top 10 Open Web Application Security Project (OWASP) threats and Common Weakness Enumeration (CWE) listings. It even follows best practices for crypto libraries. If the security scan spots a vulnerability, it will offer up some fixes. This speeds things up and cuts down on development costs. Considering the quirks that can come with AI-generated code, having this built-in security feature is a huge plus.

A perk of CodeWhisperer is how it meshes with AWS services. This means that developers don't need to be AWS wizards or slog through heaps of docs to use AWS. Here are a few examples of what you can prompt it to do:

> *Prompt:* # Write a Python function to upload a file to an S3 bucket.
>
> *Prompt:* # Construct a Python Lambda handler to process records from a Kinesis stream.
>
> *Prompt:* # Write a Java method to query items from the DynamoDB table by sort key.
>
> *Prompt:* # Write a bash script to stop an EC2 instance by instance ID using AWS CLI.
>
> *Prompt:* # Create an RDS instance with a PostgreSQL engine using AWS CDK in TypeScript.

Amazon did a study on customers using CodeWhisperer and found that, on average, developers were 27% more likely to successfully finish tasks and did so 58% faster than developers who did not use the tool.

Take Accenture's experience, for instance. Using CodeWhisperer for AI projects managed to cut development time by up to 30%. Tasks like preprocessing data, usually time-consuming and tedious, got a lot quicker and more efficient. By simply using prompts like the following, developers significantly boost their productivity:

> *Prompt:* # Create a preprocessing data class script for ML data

Consider Persistent Systems, a global digital engineering firm. Normally, new developers needed at least four months of training to get project-ready. However, with CodeWhisperer, they slashed this time to just one month.

Google's Duet AI for Developers

Google is the brains behind Duet AI for Developers. This nifty AI-powered programming tool supports IDEs like VS Code, IntelliJ, PyCharm, GoLand, WebStorm, Cloud Workstations, and Cloud Shell editor. Plus, it's got your back with over 20 programming languages. It's also built on Gemini, which is Google's top-of-the-line LLM.

Duet AI has the features you would typically see with an AI-assisted programming system, such as chat. There are also shortcuts, called smart actions, which are one-click magic tricks for things like code explanations and unit tests.

Duet AI also takes security seriously. It's got all the heavy-duty protections and safeguards you need, like Private endpoints (Private Google Access), VPC Service Controls, and enterprise-grade access controls.

And let's not forget about governance rules. When you toss your code into Duet AI, you can rest easy knowing it's not getting used to train any shared models or build products. You've got full control when it comes to your data and intellectual property. It will also flag instances where Duet AI thinks certain code may be copied—at length—from a repository.

A big part of Google's strategy with Duet is to build an extensive partnership ecosystem. For example, some partners provide support for developers on Google Cloud for certain environments. This allows for better code suggestions as well as documentation and knowledge sources. Here are some examples:

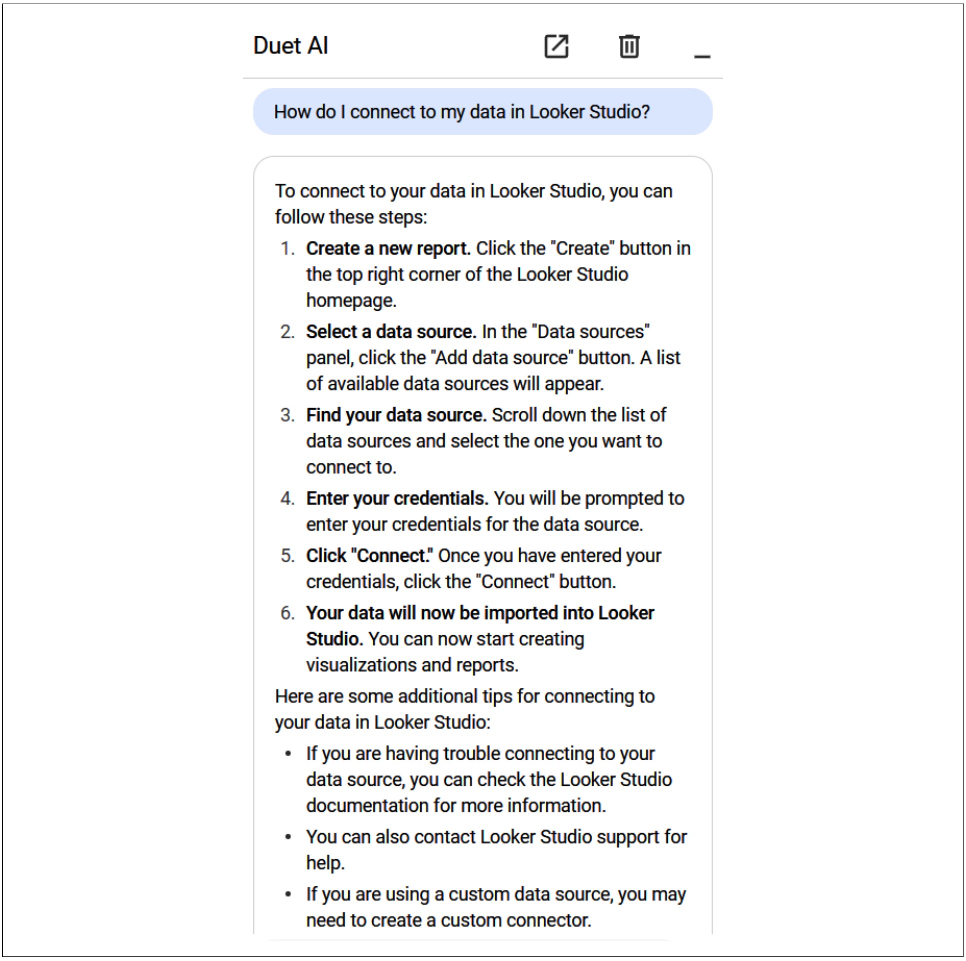
Elastic
: This is the big shot in the search analytics space. Its Duet AI integration allows a developer to get the lowdown on how to query, test, and work with Elastic data without ever having to leave the development environment.

HashiCorp
: This is a top provider of cloud infrastructure automation software. It has built a system that makes it easy to use Terraform for writing configurations—based on the HashiCorp Configuration Language (HCL)—and automations.

MongoDB
: This is the leader in NoSQL databases. Using Duet AI, you can get access to best practices and help with building apps.

Neo4j
: This is one of the royalty of graph databases. You can use Duet AI to get advice on such things as the Cypher query language. This can make it possible to unearth hidden relationships and patterns in complex datasets.

Duet AI has a nifty trick up its sleeve. It has built Duet AI into the Google Cloud console. To use it, click the Activate button on the top right of the main screen of the console. Then a chat panel will pop up, as shown in Figure 5-1. You can then enter prompts for tasks like creating scripts, understanding logs, or diving into JSON.



How do I connect to my data in Looker Studio?

To connect to your data in Looker Studio, you can follow these steps:

1. **Create a new report.** Click the "Create" button in the top right corner of the Looker Studio homepage.
2. **Select a data source.** In the "Data sources" panel, click the "Add data source" button. A list of available data sources will appear.
3. **Find your data source.** Scroll down the list of data sources and select the one you want to connect to.
4. **Enter your credentials.** You will be prompted to enter your credentials for the data source.
5. **Click "Connect."** Once you have entered your credentials, click the "Connect" button.
6. **Your data will now be imported into Looker Studio.** You can now start creating visualizations and reports.

Here are some additional tips for connecting to your data in Looker Studio:

- If you are having trouble connecting to your data source, you can check the Looker Studio documentation for more information.
- You can also contact Looker Studio support for help.
- If you are using a custom data source, you may need to create a custom connector.

Figure 5-1. The chat panel for AI Duet in the Google Cloud appears on the right when Activate is clicked

The chat is even available in the Google Log Explorer. If you spot a log that's giving you trouble, highlight it and click the Explain this Log button. The log will pop right into the chat panel, and from there, you can ask for an explanation and even get some suggestions on how to fix it.

The cost for Duet AI is $19 per user per month. There is also an up-front annual commitment. If you are a new Google Cloud customer, there is a $300 free credit.

> Turing AI, an AI technology service company, has been using Duet AI. It has been able to achieve a hefty 33% boost in the development team's productivity.

Tabnine

Tabnine stands as the pioneer in AI-assisted programming tools. It all started in 2013 when Dror Weiss and Eran Yahav set up the company. They'd been in the software game since the 1990s, specializing in code analysis and simulation. But it was their hands-on experience with the complexities of development that led them to a light bulb moment: What if AI could be the solution?

And indeed, it did help. However, this was before the era of transformer models. Faced with no other options, the founders rolled up their sleeves and built their own models. This journey helped them deeply understand AI's role in software development. Fast-forward to the last few years, and Tabnine has embraced the transformer model.

The company has invested heavily in security systems. Here's how it works: when you're typing away on your code, each character gets tokenized and encrypted before being sent over to Tabnine's inference server, which could be cloud based or on-site. Tabnine makes a point of not storing your data, and no employee gets to read it. Plus, Tabnine provides SOC-2 compliance.

For training its model, the company uses open source code with permissive licenses like MIT, Apache, and BSD. This is important for organizations that are keen on protecting their IP. Tabnine also values transparency about the code it uses for training. It even gives developers ways to opt out of repositories that may be used in Tabnine's training datasets. This shows the company's careful approach to licensing when it comes to training generative models.

Tabnine's pricing structure includes three tiers:

- The Starter plan is free and caters to individual users, offering basic features like short code completions and community support.
- The Pro plan costs $12 per month per user. It is designed for professional developers and small teams and includes advanced features like whole-line and full-function code completion, as well as natural language–to–code completions, with standard support.
- The Enterprise plan is aimed at larger organizations seeking comprehensive security, control, and customization. It offers unlimited users, private and custom AI models, private deployment options, and premium support. Pricing is available on request.

Tabnine has garnered a large user base, with over a million monthly users and hundreds of thousands engaging with it daily. Notably, some of its prominent customers include tech giants like Google, Amazon, Netflix, and Atlassian.

Replit

Replit is a versatile web-based IDE that supports numerous programming languages and allows for hosting applications, known as repls. Its rich collaboration features are similar to those found in Google Docs. Expanding its reach, Replit has also developed a desktop version, which is available for macOS, Windows, Linux, Android, and iOS. The platform boasts a substantial community of around 23 million developers.

Replit was founded in 2016 by Amjad Masad, Faris Masad, and Haya Odeh. The concept for Replit was conceived by Amjad over a decade prior to its establishment. During this period, Amjad honed his skills in engineering roles at Yahoo! and Facebook, focusing on building development tools. He was also instrumental in the founding of Codecademy, further showcasing his commitment to innovative technology and education in coding.

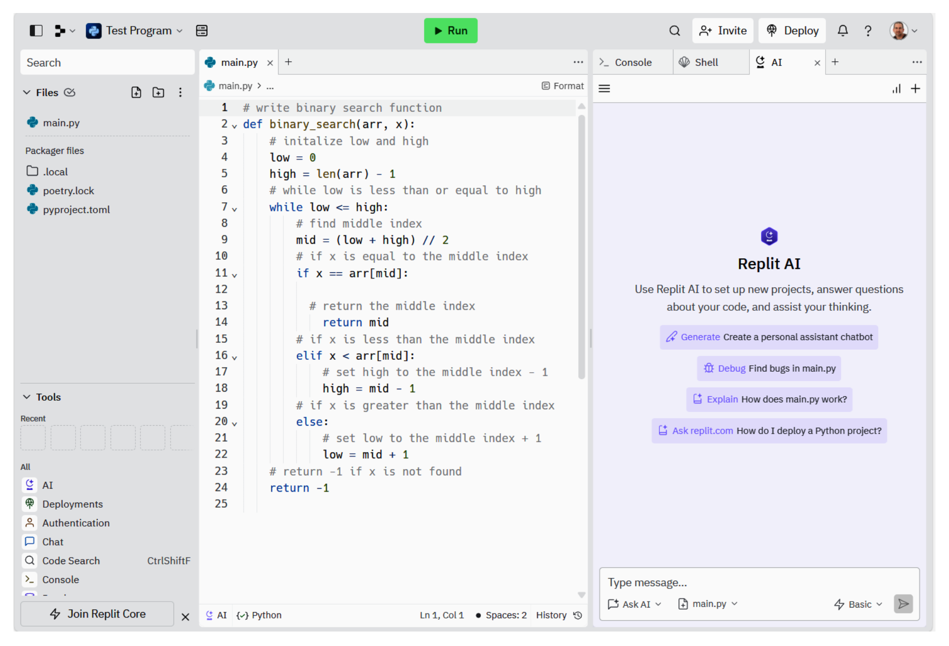
Replit offers three subscription tiers:

- The free tier provides unlimited public repls and 10GB of storage.
- The Hacker plan costs $7 per month per user and offers unlimited private repls. It comes with varying levels of memory and storage to suit different needs.
- The Pro plan is available for $20 per month per user and includes everything that comes with the Hacker plan. In addition, while the basic system for code development is available with both lower tiers, Pro users gain exclusive access to the most powerful AI model along with advanced features, offering a more robust experience in code development.

“Replit is an all-in-one software creation platform,” said David Hoang, VP of marketing and design at Replit. “The product experience is designed to reduce the friction in the software development process: dev environment, code authoring, and deploying to production. Because of this, Replit is able to integrate AI beyond code generation. Our code complete model is also powered by our own language model.”

Replit’s LLM is trained on a whopping one trillion tokens and is capable of understanding 30 programming languages. When working on Replit, the code you write in public repls, including keystrokes, might be used to further train Replit AI. If you’d rather keep your work private and out of the training pool, you can opt to make your repl private. It’s important to note that your rights over your code don’t change when using Replit AI. Code in public repls is automatically licensed under the MIT License, as outlined in Replit’s licensing information.

Figure 5-2 shows the IDE. On the top left is a file tree and search box. In the middle, you’ll find the editor, and the output (e.g., the console) shows up on the right side of the screen. On the bottom left, you see the available tools. If you select AI, the chat feature comes up in the right panel.



```
# write binary search function
def binary_search(arr, x):
    # initalize low and high
    low = 0
    high = len(arr) - 1
    # while low is less than or equal to high
    while low <= high:
        # find middle index
        mid = (low + high) // 2
        # if x is equal to the middle index
        if x == arr[mid]:

          # return the middle index
            return mid
        # if x is less than the middle index
        elif x < arr[mid]:
            # set high to the middle index - 1
            high = mid - 1
        # if x is greater than the middle index
        else:
            # set low to the middle index + 1
            low = mid + 1
    # return -1 if x is not found
    return -1
```

Use Replit AI to set up new projects, answer questions about your code, and assist your thinking.

Generate Create a personal assistant chatbot

Debug Find bugs in main.py

Explain How does main.py work?

Ask replit.com How do I deploy a Python project?

Figure 5-2. The UI for Replit includes a file tree and search box, icons to select tools, an editor box, and an output panel

If you highlight code and press the right mouse button, you will get these options:

Explain

Replit offers a feature to give a helpful explanation of your code. If the explanation doesn't quite hit the mark, you can simply click Retry to ask Replit to take another shot. Additionally, if you have specific questions, there's an Ask Chat option available from a pull-down menu at the top of the Explain box. Replit provides other useful functionalities as well, such as Ask Replit Documentation, Edit Code, and Generate Code, to assist with your coding.

Modify (Ctrl+I)

In Replit, you have the ability to modify code through prompts. For example, if you've highlighted a section of code for a binary search, you can write a prompt requesting a transformation of that Python code into a JavaScript function. Alternatively, you could ask for the code to be altered to use a recursive approach. The Modify feature offers a flexible and interactive way to experiment with and refine your code.

To showcase just how user-friendly and powerful this tool is, let's check out some interesting stories:

Diabetes app

When London-based iOS developer Marwan Elwaraki learned of his younger brother's diabetes diagnosis, he was determined to create an app for monitoring blood sugar levels. He and his wife, Salwa Al Alami, wanted to team up to create something useful and unique. "While on a plane, I prototyped a Lock Screen widget by connecting a widgets app with the blood sugar tracker's API and was able to see my brother's blood sugar," said Marwan. "This was significantly better than having to open the app to see the latest reading."

They added buttons to send messages and improved the texting capabilities. Then they published it as a public app on the iOS App Store.

While Marwan was an experienced mobile frontend developer, he had little experience with Python or backend development, and Salwa was a product manager with no coding experience. At first, they used ChatGPT, but Replit AI would make a big difference. "The AI tools have helped generate or at least double-check almost all the backend written for my app," said Marwan. "I've heard the stories about big increases in developer productivity and I can honestly vouch this in my case."

Hackathon to startup

Priyaa Kalyanaraman, with her impressive stints as a product manager at Microsoft, Snapchat, and Waymo, lacked a technical background but didn't let that stop her. She took part in the Craft Ventures AI Hackathon, leveraging Replit AI to whip up an app designed to simplify and add fun to content creation. Her efforts paid off handsomely as she clinched the $10,000 grand prize. Leveraging her product management expertise, she meticulously prepared detailed specs and logically structured the app. This winning app later became the foundation for her startup, Lica, which successfully secured a pre-seed investment round.

CodeGPT

CodeGPT, an extension designed for VS Code, provides a variety of pricing options. These range from a free plan to a premium tier costing $49.99 per month for each user. There's also a 10-day free trial available for those who want to test the service before committing to a subscription.

To set up the CodeGPT extension, first click on File. Then navigate to Preferences and select Settings. This will open up the Settings window. From there, on the left side, click on Extensions and then choose CodeGPT.

What's interesting about CodeGPT is its ability to integrate with a range of LLMs. This includes models from OpenAI, Cohere, AI21, and Anthropic, among others. Setting up an API account with Hugging Face grants access to a wide array of open source platforms, further expanding the tool's versatility.

Once you select a model, you can configure it based on factors like the following:

- Max tokens for the prompt and response
- Temperature
- Window memory (the number of past threads in a chat)

Another interesting feature of CodeGPT is its API, which is handy for applications like chatbots, virtual assistants, or any other system requiring the understanding and generation of human-like text. Its RESTful API design ensures broad compatibility and straightforward integration with various platforms. Moreover, CodeGPT offers SDKs in Python and JavaScript, adding to its implementation flexibility. This API is part of a broader initiative to transform the coding process, aiming to make sophisticated AI tools available for a wide range of development projects.

Cody

In 2013, Quinn Slack and Beyang Liu came together to found Sourcegraph, with the aim of developing code search tools. Beyang was inspired by his experience at Google, where he saw firsthand the advantages of the company's internal platform for code intelligence and insights, which was especially beneficial for handling large codebases.

Sourcegraph's primary goal became the creation of advanced tools to tackle "big code." The level of complexity is a significant challenge in this field. A survey by Sourcegraph highlighted this issue: about 77% of developers reported a fivefold increase in their codebases over three years. What's more, 72% expressed concern that the burgeoning scale of big code could pose a real threat to their company's capacity for innovation and competitiveness.

AI plays a crucial role in Sourcegraph's strategy, and the company has developed an AI-driven code generation system named Cody. Cody relies on LLMs from Anthropic and OpenAI. There is also the use of Starcoder as well.

"We have leveraged our search capabilities for this tool," said Liu. "Keep in mind that 80% of a developer's time is about reading and understanding code, not creating it. So with Cody, it scans the whole codebase. But we also make it easy to bring in other libraries and frameworks. What we have seen is that there are better results."

Here are some of the prompts you can use with this platform:

> *Prompt:* How is this repository structured?
>
> *Prompt:* What does this file do?
>
> *Prompt:* Where is X component defined?

In terms of pricing, Sourcegraph offers a free product and two paid tiers:

- The free tier includes various features like code autocomplete, chat, and context awareness.
- The Pro edition costs $9 per user per month. With Pro, you get:
  - — Unlimited autocompletes, messages, and commands
  - — Personalization with larger codebases
  - — Multiple LLM choices for chat
  - — A higher level of support
- The Enterprise edition is tailored to the needs of larger organizations. It costs $19 per user per month and offers:
  - — User management
  - — Single-tenant deployment
  - — Audit logging
  - — Pooled organization usage
  - — Daily rate limits while in beta
  - — Web and API access
  - — Configurable LLMs

To get a sense of Cody's potential, let's consider what Deepak Kumar, a full-stack developer and an open source contributor to Cody, has to say: "The features that stand out for me are the Chat and the Commands. I use the Chat to answer any coding-related questions, starter ideas for a project, or fix a bug. Commands are predefined custom prompts, which are helpful as they help me wrap up the ad-hoc tasks like writing docs and adding tests."

In July 2021, Sourcegraph announced a Series D funding of $125 million at a $2.625 billion valuation. Investors included Andreessen Horowitz, Insight Partners, and Geodesic Capital.

CodeWP

WordPress is a highly popular open source content management system (CMS) renowned for its simplicity and versatility as well as its extensive selection of themes and plugins. Initially launched as a blogging platform, it has expanded to accommodate various types of websites, including those optimized for ecommerce, portfolio presentation, and general business. Its appeal lies in its user-friendly interface and customizable options. A significant portion of the web relies on WordPress; W3Techs reports that about 45.8% of all websites on the internet are powered by this platform.

AI-assisted programming presents a significant opportunity for WordPress development, a potential that CodeWP has capitalized on. The company was founded by James LePage, who created a WordPress agency during his high school years. Although the business experienced rapid growth, it was labor intensive. The introduction of AI-assisted programming technology enabled a substantial increase in productivity. A key benefit was James's deep understanding of the common use cases in WordPress development, which further enhanced the effectiveness of the AI.

CodeWP offers a web-based platform with a free version and two premium options. Both premium plans come with a discount of up to 33% for annual subscriptions:

- The Professional plan, priced at $18 per month per user, provides additional AI transactions, projects, and support.
- For larger-scale needs, there's the Agency plan at $48 per month per user. This caters to professional web development and marketing firms, with unlimited projects and collaboration features.

Go to the CodeWP site to create an account. Click on Create New and then select New Snippet to create code or New Chat if you just have questions.

The left side of the screen has options for storing snippets as well as for tracking conversations and listing projects. In the middle is the editor, which has contents of the file you're working on. You can click New File if you want to create a new file. On the right side is a chat function.

Here are some sample prompts:

> *Prompt:* Register a custom post type called "book" with the label of "Books."
>
> *Prompt:* Create a WordPress widget called "My Widget" using the WP_Widget class.
>
> *Prompt:* Create a custom plugin activation hook for the register_activation_hook function.

CodeWP extends its support to various key components within the WordPress ecosystem. This includes compatibility with major tools like WooCommerce, WPSimplePay, Gravity Forms, SearchWP, and Contact Form 7 among others.

CodeWP features a live preview option that provides an instant view of what your code will do, making testing and debugging a breeze. Moreover, when working on WordPress projects, CodeWP simplifies the deployment process. Its integrated tools allow you to easily transfer your code to a WordPress site, making the task of applying and testing your code in a real-world setting much more efficient.

Warp

The command-line interface started back in the 1960s, growing up alongside the early days of operating systems like Multics and Unix. While flashy graphical interfaces have emerged over the years, the CLI has mostly stayed true to its roots. It's all about keeping it simple: you type your commands in plain text and get back just what you need, no frills attached.

CLIs are incredibly widespread despite their simplicity. They're essential for taking care of cloud systems, dealing with files and programs on your computer, and setting up smooth workflows. They're good at automating tasks, processing data, managing networks, and developing software.

Getting the hang of CLIs can seriously speed things up and make you more precise. A standout benefit is that you can mix and match different commands to handle complex tasks. Plus, since you can program these command lines, you've got much more room to tweak things to your liking and automate tasks. This means developers can set things up just how they need them, making their whole workflow smoother and more efficient.

While CLIs are powerful, they're not perfect. A big problem is that they're not great for collaboration and so can trip up productivity and teamwork. When you close a terminal session, you lose everything. Command lines are also usually stuck on one machine, making it a pain to switch between devices. Moreover, terminals can be intimidating, especially for tricky tasks. And let's not forget: getting good with CLI commands and their syntax isn't exactly a walk in the park for a lot of people.

So yes, there is room for innovation in the category. And a startup that is helping to lead the way is Warp.

Zach Lloyd, who is the founder and CEO, started the company in 2020. Before this, he worked at Google as a principal engineer for Google Sheets. "I've been a developer for over 20 years," said Lloyd. "While the CLI is powerful and very useful, there was lots of room for improvement."

Warp was built with Rust, a programming language that's known for being fast and efficient. Instead of sticking to the usual JavaScript, Warp uses Rust's strengths to run almost like an app you'd install on your computer, but instead it's right in your browser. This clever twist with Rust means a smoother and snappier experience for anyone using Warp, making the whole interaction with the app a lot better.

Think of Warp as a terminal on steroids. It boasts advanced text input features like selections, cursor positioning, and completion menus, allowing users to seamlessly navigate their command history. This eliminates the hassle of scrolling through long texts. Furthermore, Warp makes copying output a breeze with a simple click.

In Warp, every command and its output form a distinct block. You can effortlessly hover over these blocks to revisit past commands. Selecting a block enables the use of Warp AI, accessible via the right-click menu, which provides explanations and solutions for errors. This is in stark contrast to traditional CLIs, which offer limited guidance and depend heavily on the user's expertise to decipher error messages and troubleshoot.

Warp AI also comes with a chat function. You use this by starting the prompt with a #. For example, suppose you want to know the following:

> *Prompt:* # How do I use grep to exclude directories in a recursive search?

Warp will provide an answer to this question, which you can then copy into the terminal.

Now keep in mind that Lloyd is an avid user of Copilot for his own development. "I use this tool alongside Warp," he said. "They work very well together."

A big fan of Warp is Mike Krieger, a cofounder of Instagram. He said, "I have been using Warp every day at work. My favorite thing is the speed: both in terms of how fast it works and also how fast you feel while using it, especially the excellent type-ahead and search. Warp brings terminals into the modern day, and I can't wait to see where they take Warp."

Warp is available on Mac, Linux, and Windows. There is a web version as well. The pricing for Warp is structured into three tiers:

- The free version provides up to 20 AI requests daily.

- For more extensive use, there's the Team edition, priced at $12 per month for each user and offering up to 100 AI requests per day.
- Larger organizations with specific requirements will want to consider the Enterprise version, with custom pricing tailored to the customer's needs.

Bito AI

In 2006, Amar Goel and Anand Das started PubMatic, an online advertising company, and took it public in 2020. Their journey with PubMatic led them to recognize the potential of AI to enhance coding productivity, sparking the idea for Bito AI. Mukesh Agarwal, with his experience as a product lead at Microsoft and Ernst & Young, joined them as a part of the founding team.

Bito has multilingual capabilities, supporting 20 languages, including unique blends like Hinglish, a combination of Hindi and English.

This tool is equipped with several notable features. One allows you to create custom prompt templates for frequently used coding structures, streamlining the development process. Moreover, the platform includes security and performance checks, providing recommendations for optimizations to enhance both the safety and efficiency of code.

Bito boasts a substantial context window of approximately 240,000 tokens, enhancing its understanding and analysis capabilities. Furthermore, it uses a vector database that's local to the user's machine. This enables the system to effectively handle large codebases, ensuring the generation of more relevant and accurate results. This capability significantly contributes to the tool's efficiency and effectiveness in managing extensive coding projects.

It is also good with handling memory leaks. These happen when a program grabs some of your computer's RAM and then forgets to give it back. If this keeps happening, it can cause real headaches. As the program keeps running, it keeps leaking memory, which eats into the total memory you have available. This can slow things down, as your system starts using swap space because there's not enough physical memory left. And if the leakage gets really bad, memory leaks can even make the program or your whole computer crash because it runs out of memory.

Memory leaks can be tricky to spot and fix. They usually don't cause problems right away and might only start showing up after the program's been running for a while. This delay makes it even harder to figure out and sort them out. But Bito can sniff out potential memory leaks. You just use the Insert Code Selected in IDE option and throw in a prompt like this:

> *Prompt:* Identify any issues with the code.

And guess what? It not only flags the problems but also throws in some suggested fixes.

When it comes to privacy, Bito makes sure not to store any of your actual code. However, it does keep hold of the metadata that gets generated along the way.

Two versions are available:

- There's a free version for individual users.
- A paid option is available at $15 per month per user. It includes features like unlimited AI code completions and access to the extensive 240,000-token context window, enhancing the overall coding experience.

Based on Bito's internal analysis, users of the tool reported a 31% increase in productivity. They also used the tool almost 200 times a month. Currently, the platform boasts around 100,000 users.

Cursor

Anysphere is the developer of Cursor. The company certainly has a bold vision for it:

> In the next few years, we'd like to build a code editor that is more helpful, delightful, and fun than the world has ever seen. Cursor should be a place where it's impossible to write bugs. An editor where you whip up 2,000-line PRs with 50 lines of pseudo code. A tool where you get any codebase question answered instantly. Perhaps even an interface where the source code itself starts to melt away…

In the meantime, Cursor is still quite powerful. It has also attracted interest from investors like OpenAI. In October 2023, the company successfully raised $8 million in funding, bringing its total capital raised to $11 million.

Cursor offers a choice between GPT-4 and GPT-3.5, but it also enhances its capabilities with proprietary models. These models include 1.4 billion vectors and access to 150,000 codebases. Additionally, the platform incorporates advanced AI techniques like Merkle trees, further boosting its effectiveness.

Cursor is a fork of VS Code and is compatible with Windows, Mac, and Linux. This is a huge plus as it spares you the hassle of learning a new IDE. When you download Cursor, it conveniently lets you import your existing VS Code extensions. You can take advantage of its AI features right within the CLI, seamlessly integrating advanced capabilities into your familiar coding environment.

For example, in the chat area, you can easily import documentation and ask it questions. Here's a sample prompt:

> *Prompt:* @python docs

Or you can use @ to work with a particular file. You can enter something like:

> *Prompt:* What does @the_app.ts do?

Cursor offers the ability to debug directly in the terminal. Its AI sifts through your files and engages in a logical process to pinpoint and attempt to resolve the issue, streamlining the troubleshooting experience.

Jeffrey Biles, a full-stack web developer, is an early adopter of Cursor. He said:

> I really enjoy being able to quickly load app context into the AI with a keystroke. It works great for answering questions, remembering syntax, refactoring small sections of code, and writing boilerplate. However, it reaches its limits when asked to understand larger data models spread across multiple files. However, this will only get better as the models get better, such as with larger context windows.

Tosh Velaga is another user of Cursor. He is a software engineer and founder of several AI companies like Typeblock and STBL, which makes Stable Diffusion. According to him:

> A feature I really like is the Command+L shortcut to ask questions about the code. This is super useful when I'm in a new codebase with a backend language I do not have experience with. This feature has helped me get up to speed much faster. Something else I think is a benefit is that you can bring your own API key so you can save on costs and switch between GPT-3.5 and GPT-4.

Code Llama

There's a growing number of open source projects focused on AI-assisted development, and it's a hot topic in academic circles, too. One of the big names in this space is Meta. Its Code Llama system has been making waves since its launch in August 2023.

Code Llama is built on the LLaMA 2 LLM, which is trained on a staggering 2 trillion tokens and a context length of 4,096 tokens. It includes fine-tuned models that have been beefed up with training on over a million human annotations, making them even more effective and reliable.

LLaMA 2 has special versions like LLaMA 2-Chat, which is tailor-made for chat applications and outperforms other open source chat models on key benchmarks, especially in helpfulness and safety. Plus, it's integrated with platforms like Hugging Face and has major partnerships with companies like Microsoft and Amazon for cloud services deployment.

Code Llama is versatile, supporting most of the big languages like Python, C++, Java, PHP, JavaScript, C#, and Bash. It comes in three model sizes with 7, 13, and 34 billion parameters respectively. They've all been trained on a massive 500 billion tokens of code. The interesting thing about the 7B and 13B models is that they're geared up with fill-in-the-middle ability, so they can slot code into existing scripts. This makes them useful for tasks like code completion. The 7B model can run on just a single GPU, and it's usually quicker. But, as you'd expect, the 34B model packs a bigger punch in terms of power.

Code Llama goes beyond what the LLaMA 2 base model offers. All its models can handle up to a whopping 100,000 tokens in context. This is great for whipping up code for longer programs. This bigger context window makes it a whiz at debugging too.

There are a couple of specialized versions of Code Llama. First up, there's the Python version, which has been sharpened by using a massive 100 billion tokens of code. Then there's the Instruction version. This one has been given extra training in natural language to boost results on natural language applications.

Code Llama 34B has shown impressive results. It has scored a solid 53.7% on HumanEval and 56.2% on MBPP (Mostly Basic Python Programming). These scores are better than those of other open source projects, and Code Llama's performance is on a par with that of ChatGPT. Moreover, Code Llama has been put through the wringer with security testing. Meta notes that its responses are even safer than ChatGPT's.

Other Open Source Models

There's been some cool innovation happening in open source for code generation. Let's check out some of the standout platforms.

StableCode

Stability AI, which is behind the popular text-to-image system Stable Diffusion, came up with StableCode. StableCode is trained on the open source project BigCode, a joint venture between Hugging Face and ServiceNow Research. BigCode put together a dataset called The Stack, which is a massive 6.4TB collection of unique, freely usable GitHub code—perfect for training AI models.

StableCode's got its own unique technology, using rotary position embedding (RoPE) instead of the ALiBi (Attention with Linear Biases) method. StableCode has been fine-tuned significantly, including by cleaning up the data. It's quite versatile, supporting languages like Python, Go, Java, JavaScript, C, Markdown, and C++.

There are three versions of StableCode to check out:

StableCode-Completion-Alpha-3B-4k
: This is a decoder-only model with 3 billion parameters, great for handling a bunch of languages and a 4,000-token context length.

StableCode-Instruct-Alpha-3B
: Also a 3 billion parameter model, this version is tuned for instructions.

StableCode-Completion-Alpha-3B
: Also with 3 billion parameters, this version is a whiz at doing extensive code completion, handling up to 16,000 tokens.

AlphaCode

DeepMind, which is part of Google, whipped up the AlphaCode AI system. It performs well in code competitions, scoring in the top 54% in contests on Codeforces, a website that hosts competitive programming contests. AlphaCode was able to handle different sorts of tough tasks that needed a mix of critical thinking, logic, algorithms, coding skills, and understanding of natural language. This system even got a shoutout for its abilities in *Science* magazine.

Code competitions are usually about tricky algorithms and brain-bending theoretical problems, which are a bit different from your everyday programming tasks. Take this one test that AlphaCode has to tackle, a tough problem called Backspace. Competitors had to play around with two strings, *s* and *t*, and use Backspace to change *s* into *t*. Success is not just about getting the complex problem right. It's also about coming up with smart algorithms. This shows off the kind of advanced problem-solving skills needed for competitive programming.

DeepMind aims to use AlphaCode for the complete automation of code generation, marking a significant advancement in AI-driven coding.

PolyCoder

PolyCoder is particularly good at speeding up C programming, so the system appeals to game developers. PolyCoder is also a champ at summing up code in multiple languages. Thanks to training on a varied dataset, it's good at picking up common patterns and structures in a bunch of programming languages, like C, C++, Java, Python, and JavaScript.

PolyCoder is trained on a data set of 249 GB covering 12 different programming languages. Plus, it's packed with 2.7 billion parameters.

CodeT5

CodeT5 uses an all-in-one pretrained encoder–decoder model. This makes it good at different coding tasks, such as spotting bugs in code or finding code clones, as well as at new structures, whether it's turning programming language into plain English or vice versa or even switching between programming languages.

Its design includes special tricks that boost its ability to get code right, both in understanding it and spitting it out, by making the most of the deep structure that's built into programming languages.

CodeT5 has really shown its stuff on 14 different challenges in CodeXGLUE (General Language Understanding Evaluation benchmark for CODE). It's blown past older models like PLBART in all the generating tasks, like summing up code, turning text into code, translating between codes, and polishing code. And when it comes to understanding tasks, CodeT5 has been quite effective in finding bugs and done just as well in spotting copycat code.

Enterprise Software Companies

Mega software companies like SAP, ServiceNow, and Salesforce have been making their own code generation systems. They've got major advantages: huge customer bases, enormous resources, extensive distribution networks, and trusted brands. Their big talent pool is a big help, too. Plus, they're large enough in the market to set trends and standards.

Integrating new technology with their existing products means they can offer complete solutions, keeping customers coming back. Because they've already got the infrastructure and a global presence, they can scale up these new technologies fast, meeting what the market wants both now and in the future.

Here's a look at some of the code generation systems from these companies:

Salesforce Code Builder

This makes it easy for a developer to tweak their CRM and other bits and pieces in Salesforce. It helps get things just right with the latest Salesforce languages and frameworks. Furthermore, you can grab third-party add-ons from the Open VSX marketplace, which is a big open playground for VS Code extensions. And the best part? You don't need to mess around with installing or setting up these features. They're ready to go right off the bat.

SAP Build Code

This is made for Java and JavaScript users. Plus, it hooks up with SAP Joule, which is SAP's own AI sidekick. You also get to tap into some powerful database systems, like SAP's HANA (High-performance ANalytic Appliance).

StarCoder LLM

ServiceNow and Hugging Face teamed up to create this open source project. It's beefy, with 15 billion parameters and trained on a whopping trillion tokens across more than 80 programming languages. They keep things above board with governance, safety, and generally sticking to the rules, training only on code that's gotten the green light from permissive licenses.

Conclusion

AI-assisted programming tools are shaking things up for developers, offering some seriously cool new abilities. Big names like Amazon, Google, Meta, and Salesforce are all over this. And let's not forget the small, innovative startups that are turning heads with their neat features and wallet-friendly prices. Open source projects are also making waves by making top-notch AI coding tools available.

As these AI models get bigger and smarter, they're getting even better at writing, explaining, and tweaking code. Looking ahead, successful enterprises will make these tools fit smoothly into what developers are already doing.

CHAPTER 6

ChatGPT and Other General-Purpose LLMs

In this chapter, we're going to dive into some widely used general-purpose large language models that can whip up code. Sure, they might not have all the bells and whistles like those that specialize in coding—such as with integration with IDEs, sophisticated security, or guardrails for the types of code used to train the models—but they're still impressive. Plus, they're not just about coding; they're also great for planning and brainstorming.

We'll check out the big-name chatbots like ChatGPT, Gemini, and Claude. I'll walk through how to set them up, what they cost, what their cool features are, and how to get the most out of them.

ChatGPT

ChatGPT, short for Chat Generative Pretrained Transformer, exploded on the scene on November 30, 2022. OpenAI, the brains behind the app, didn't really splash out on marketing. Turns out, they didn't have to. ChatGPT quickly blew up and went viral all on its own.

In just five days, this app pulled in a million users, and within a couple of months, the number of users skyrocketed to 100 million. It became the quickest-growing platform in internet history.

Sandhini Agarwal, who helps to develop policy at OpenAI, noted, "I think it was definitely a surprise for all of us how much people began using it. We work on these models so much, we forget how surprising they can be for the outside world sometimes."

The buzz kept growing. By September, ChatGPT had over 1.5 billion visits, and revenues had shot up to $1.2 billion, a massive leap from a mere $28 million in 2022. But it wasn't all about ChatGPT itself. There was also a booming business with OpenAI's API.

Let's get the quick backstory on OpenAI. It started up in 2015 and had some big names from Silicon Valley backing it, like Elon Musk, Sam Altman, Greg Brockman, Ilya Sutskever, John Schulman, and Wojciech Zaremba. They were ready to pitch in up to a cool $1 billion to get things rolling.

OpenAI kicked off as a nonprofit, and it had more of an academic vibe, staffed mostly with PhDs in data science and AI. The mission was to achieve artificial general intelligence (AGI), the technology that can outdo humans in most tasks that matter. The goal had an altruistic component: to use AGI for the greater good and steer clear of any uses that might hurt people or give too much power to the few.

In the early days, OpenAI was focused on research and sharing its findings with others. The organization made its patents and code public, working hand in hand with other institutions. But the cost of creating cutting-edge generative AI systems was steep.

To keep its big dreams alive and bring in the right people, OpenAI created a "capped" for-profit company in 2019. This meant they could now legally get cash from venture funds and strategic investors, as well as give their employees a slice of the pie. That same year, Microsoft came in with a $1 billion investment. A few more rounds of investment brought in a total of about $3 billion from Microsoft. Fast-forward to April 2023, and the tech giant went all in, dropping a whopping $10 billion into the mix.

GPT-4

OpenAI's GPT-4 model exhibits remarkable versatility in code generation, owing to its diverse training data encompassing a broad spectrum of sources. This extensive training has enabled human-level performance on several professional and academic benchmarks, and GPT-4 has consistently outperformed GPT-3 and GPT-3.5 in most programming languages. This is largely due to its enhanced ability to follow complex instructions expressed in natural language and generate technical or creative works with greater depth. Another key factor is the 32K context window.

Interestingly, when evaluated on specific code generation benchmarks, an enhanced version of GPT-4 integrated with Reflexion—a framework for creating sophisticated agents based on LLMs—rocked an impressive 88% pass rate on HumanEval. This score was notably higher than that of the base version of GPT-4, which had a 67% pass rate. This suggests that, with certain improvements, GPT-4 can attain state-of-the-art performance in code generation tasks.

When pitted against competitive programming scenarios on the Codeforces platform, GPT-4 scored a rating of 392 points, an improvement over GPT-3.5's 260 points. However, GPT-4 is still a newbie in the world of competitive programming, and these scores are in the bottom 5%. Even though GPT-4 has come a long way, it still has a lot of catching up to do to match human skills in tough coding challenges.

Navigating ChatGPT

For this chapter, I will be using the premium version of ChatGPT, ChatGPT Plus. A monthly subscription costs $20.

Here's the cool stuff you get:

Always in
Even when everyone's trying to use ChatGPT, premium subscribers get preferential access during peak times.

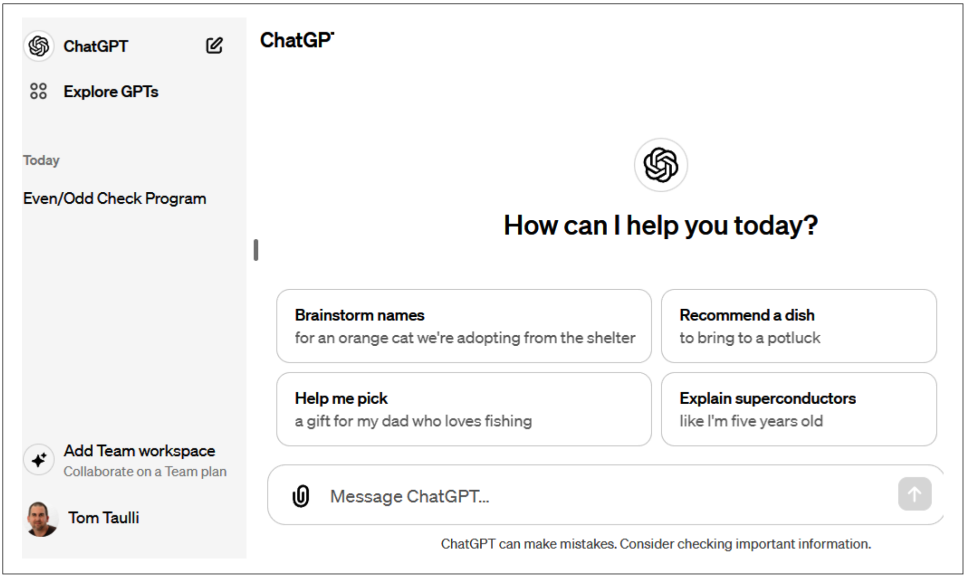
Quick replies
Things move faster with ChatGPT Plus. You get your answers quicker, making chats smoother and more fun.

First dibs on new offerings
If there's a shiny new feature or an upgrade, Plus members get to try it out before everyone else.

Latest model
You get access to it.

Figure 6-1 shows the interface for ChatGPT.

At the top left, you can hover over the dropdown to select the model you want or to access the plugin store. On the left side of the screen, you can click a button to create a new chat session. At the bottom, you will see your user profile, where you can get information about your ChatGPT plan, change your settings, and add custom instructions. You can also change the theme for the UI: options include system, dark, and light. Then at the bottom, there is the input box where you can enter a prompt.



How can I help you today?

ChatGPT can make mistakes. Consider checking important information.

Figure 6-1. The interface for the ChatGPT system allows you to select the GPT model you want and enter prompts

Let's try out some of the features. Suppose you want to create code. You enter this into ChatGPT:

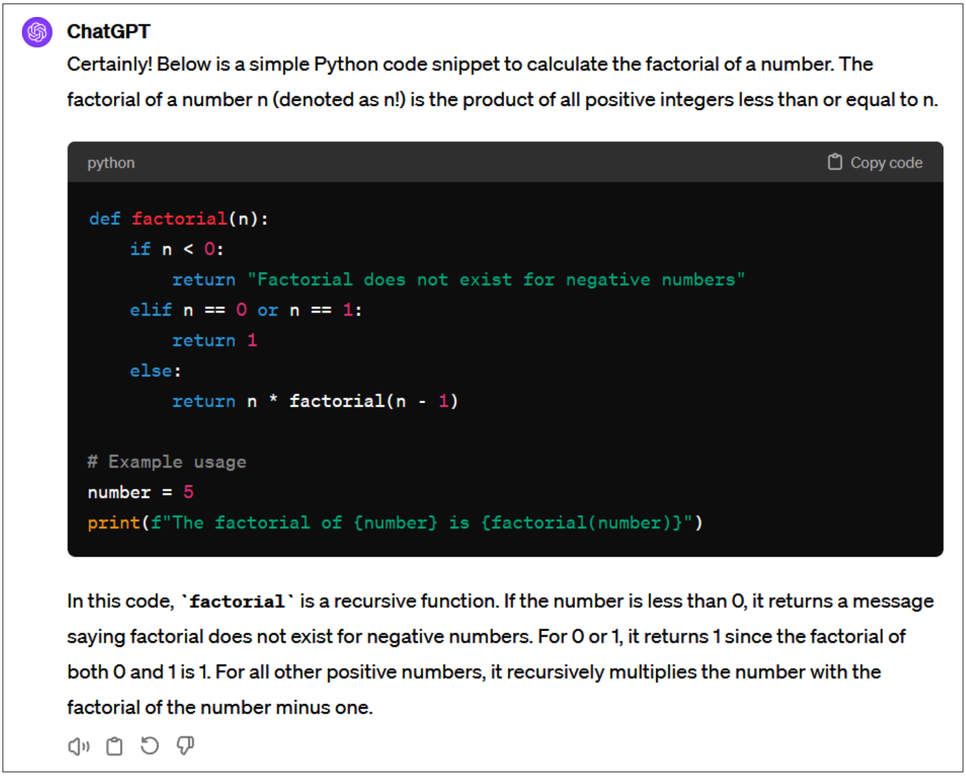
> *Prompt:* Write a Python code snippet that calculates the factorial of a number.

Figure 6-2 shows the response.

Suppose you want to see if there are other ways to create the code. You can ask:

> *Prompt:* What is another way to create this code?

ChatGPT provides another code suggestion, which is called the iterative approach. But suppose you do not know what this is? Again, you can continue with your chat. As you do this, the chat session will be logged in the top left of the screen, as you can see in Figure 6-3. ChatGPT labels the chat "Factorial Calculation in Python." If you want, you can rename it by clicking the icon next to it. There is also a delete icon.



ChatGPT

Certainly! Below is a simple Python code snippet to calculate the factorial of a number. The factorial of a number n (denoted as n!) is the product of all positive integers less than or equal to n.

```python
def factorial(n):
    if n < 0:
        return "Factorial does not exist for negative numbers"
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage
number = 5
print(f"The factorial of {number} is {factorial(number)}")
```

In this code, `` `factorial` `` is a recursive function. If the number is less than 0, it returns a message saying factorial does not exist for negative numbers. For 0 or 1, it returns 1 since the factorial of both 0 and 1 is 1. For all other positive numbers, it recursively multiplies the number with the factorial of the number minus one.

Figure 6-2. ChatGPT has created a Python code snippet in response to a prompt

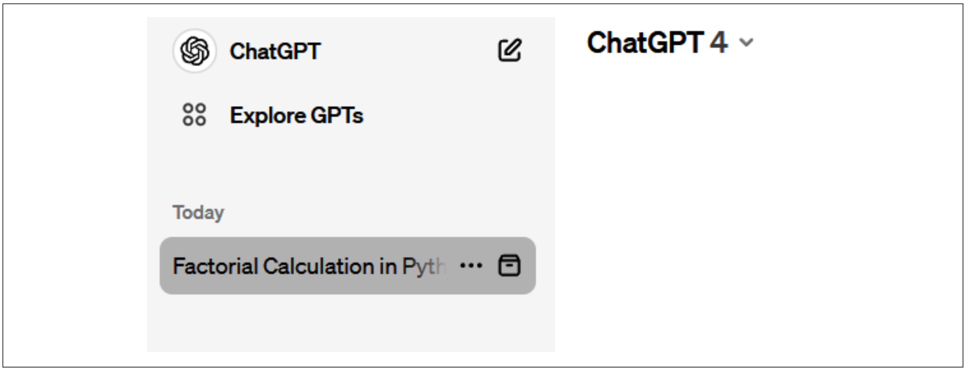


Figure 6-3. ChatGPT logs chat sessions at the top left of the screen

Want to create a new session? Click the pencil icon at the top. You'll get a blank screen.

Keep in mind that each session is self-contained. So if you ask ChatGPT in a later session about the code it created for the factorial calculation, it will not know what you are referring to.

Something else to consider is the context window for the chat session. This is important because it determines how much of the previous conversation or text the model can "remember" and use to generate coherent and contextually relevant responses. For GPT-3.5, the context window is 16K, but for GPT-4, it is 32K.

> If you want to delete all the chat threads, you will click your profile at the bottom left of the screen and select Settings and Beta. Then click the Clear button.

When you select your profile and then Settings & Beta, you'll find other options for your ChatGPT data:

Switching off chat history and model training
If you turn off your chat history, your new chats won't be used for training the AI and won't show up in the history on the side panel. But OpenAI will still hang on to all chats for 30 days before they're gone for good, primarily for legal purposes.

Sharing chats
Got an interesting thread? You can turn it into a link and share it with others.

Downloading your chat data
Choose this, and you'll get an email with all your ChatGPT conversations.

> Sometimes, ChatGPT might stop in the middle of generating text. When this happens, usually you'll see a Continue button pop up. Click that button to keep the text rolling. If there's no button, just type "continue output" in the prompt. And if it's not giving you what you're after, just click the Stop Generating button to put a pause on it.

Mobile App

ChatGPT's got your back on both iOS and Android devices. The apps are fairly similar to the web version, although, at least for now, they do not have features like plugins.

You can have voice chats with ChatGPT on your phone. This is convenient if you're not a fan of typing on tiny screens. Just head over to the New Features section in the settings and opt in to get started. Once you've turned this feature on, tap the headphone icon at the top-right corner of the home screen to start talking with ChatGPT.

You get to pick from five different voices for ChatGPT to chat back with. This voice system works thanks to a generative AI text-to-speech model called Whisper.

The mobile version of ChatGPT makes chatting even more fun by letting you share pictures with the assistant. You can snap a photo of anything, like an object, a landmark, or even some code, and then dive into a chat with ChatGPT about it. There's also a nifty drawing tool in the app, so you can highlight a specific part of your image while you're talking about it.

Custom Instructions

With custom instructions in ChatGPT, you can tweak how the responses come out to suit what you need. This feature lets you save time because you don't have to keep fine-tuning responses or coming up with new prompts every time you chat with the bot. You can switch on custom instructions in your profile.

When you set up a custom instruction, you'll need to answer these two questions:

> *ChatGPT*: What would you like ChatGPT to know about you to provide better responses?
>
> *ChatGPT*: How would you like ChatGPT to respond?

Let's take an example:

> *ChatGPT*: What would you like ChatGPT to know about you to provide better responses?
>
> *Developer*: I am a programmer working with a team that follows the PEP 8 style guide for Python code. We value clean, readable code and adhere to best practices. I usually work on data processing and analysis tasks, and I often need help with writing efficient and well-structured code.
>
> *ChatGPT*: How would you like ChatGPT to respond?
>
> *Developer*: I would like ChatGPT to provide Python code snippets that adhere to the PEP 8 style guide. It should prioritize readability and best practices in coding. When suggesting solutions, I prefer explanations as to why a particular approach is recommended and how it aligns with PEP 8 standards. I would also appreciate it if ChatGPT could point out any common pitfalls or errors related to the task at hand and provide tips for avoiding them.

Browse with Bing

ChatGPT's training has a cutoff of April 2023. This can be a problem for developers. With libraries, frameworks, and tools changing so fast, any number of new features or capabilities may have popped up since then, and ChatGPT won't be up-to-date on them.

But ChatGPT's got a trick up its sleeve: Browsing with Bing. It lets you do real-time web searches to come up with answers.

Let's say you're checking out a new framework, like LangChain. It's been on the scene since late 2022, and it has evolved a lot. It's also designed to make app development with LLMs smoother, so you might like to use it.

To activate Browse with Bing and enhance ChatGPT's understanding of where LangChain is at right now, add something like "Use the internet for this response" to your prompt. Here are some examples of prompts designed to get general information about LangChain:

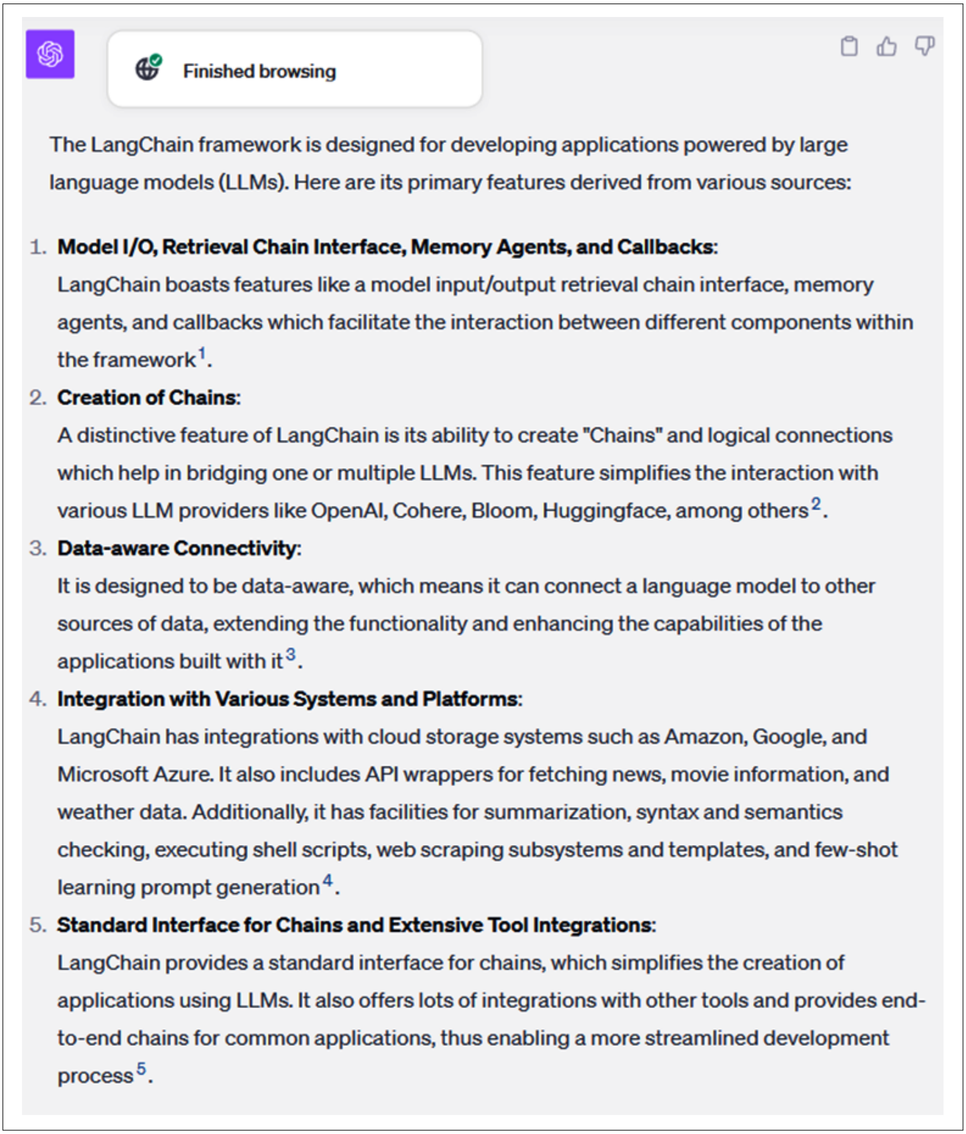
> *Prompt:* Tell me about the LangChain framework and its primary features. Use the internet for the response.
>
> *Prompt:* Check the internet for introductory resources or documentation for getting started with LangChain.
>
> *Prompt:* Browse the internet for examples of applications that have been built using the LangChain framework.
>
> *Prompt:* Look up any recent updates or releases related to the LangChain framework. Use the internet for the response.

When responding to these prompts, ChatGPT did provide fairly useful answers. For example, for the first prompt, ChatGPT came up with five main features of LangChain, which you can see in Figure 6-4.

You can also change the format of this content. Try this prompt:

> *Prompt:* Turn this into a table.

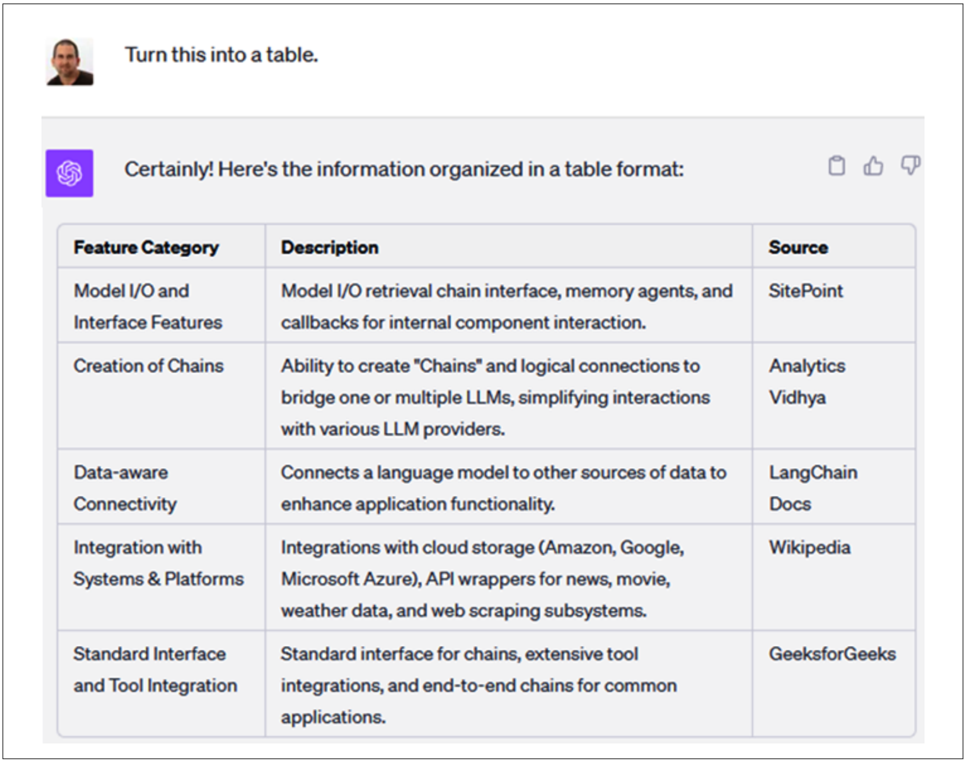
Figure 6-5 shows the result.



The LangChain framework is designed for developing applications powered by large language models (LLMs). Here are its primary features derived from various sources:

1. **Model I/O, Retrieval Chain Interface, Memory Agents, and Callbacks**:
   LangChain boasts features like a model input/output retrieval chain interface, memory agents, and callbacks which facilitate the interaction between different components within the framework[1].
2. **Creation of Chains**:
   A distinctive feature of LangChain is its ability to create "Chains" and logical connections which help in bridging one or multiple LLMs. This feature simplifies the interaction with various LLM providers like OpenAI, Cohere, Bloom, Huggingface, among others[2].
3. **Data-aware Connectivity**:
   It is designed to be data-aware, which means it can connect a language model to other sources of data, extending the functionality and enhancing the capabilities of the applications built with it[3].
4. **Integration with Various Systems and Platforms**:
   LangChain has integrations with cloud storage systems such as Amazon, Google, and Microsoft Azure. It also includes API wrappers for fetching news, movie information, and weather data. Additionally, it has facilities for summarization, syntax and semantics checking, executing shell scripts, web scraping subsystems and templates, and few-shot learning prompt generation[4].
5. **Standard Interface for Chains and Extensive Tool Integrations**:
   LangChain provides a standard interface for chains, which simplifies the creation of applications using LLMs. It also offers lots of integrations with other tools and provides end-to-end chains for common applications, thus enabling a more streamlined development process[5].

Figure 6-4. Using the Browse with Bing function helps ChatGPT to research the main features of the LangChain framework



Turn this into a table.

Certainly! Here's the information organized in a table format:

Feature Category	Description	Source
Model I/O and Interface Features	Model I/O retrieval chain interface, memory agents, and callbacks for internal component interaction.	SitePoint
Creation of Chains	Ability to create "Chains" and logical connections to bridge one or multiple LLMs, simplifying interactions with various LLM providers.	Analytics Vidhya
Data-aware Connectivity	Connects a language model to other sources of data to enhance application functionality.	LangChain Docs
Integration with Systems & Platforms	Integrations with cloud storage (Amazon, Google, Microsoft Azure), API wrappers for news, movie, weather data, and web scraping subsystems.	Wikipedia
Standard Interface and Tool Integration	Standard interface for chains, extensive tool integrations, and end-to-end chains for common applications.	GeeksforGeeks

Figure 6-5. When prompted to do so, ChatGPT can put its output into a table

When you use Browse with Bing, ChatGPT might take a few seconds or maybe even a minute or more to respond. Sometimes ChatGPT may even get stuck and spit out an error. You might have to give it a few goes before you get an answer. ChatGPT can sometimes be temperamental.

When you use Browse with Bing, the output will often have a web link to the source of the information. It's a good idea to click on it to verify the content. After all, you have to take everything generated by ChatGPT with a grain of salt. Sometimes the response might come from a social media post, for instance, and that's not always the most reliable source.

Fortunately, you can instruct ChatGPT to focus on certain types of sources. Table 6-1 provides some suggestions.

Table 6-1. Ways to instruct ChatGPT to use certain sources for information

Approach	Description
Domain restriction	Use the site operator to limit the search to reputable domains like *.edu* or *.gov* or to specific reputable organizations.
Use of scholarly keywords	Examples include *peer-reviewed*, *journal*, *study*, and *research*.
Use of publication date filters	Filter the search by recent publication dates to get current and relevant information.
Professional or scholarly associations	Specify professional or scholarly associations to seek authoritative publications or reports.

So how well does ChatGPT do with creating code for LangChain? Let's see. Here's a prompt for a simple program:

> *Prompt:* I want to write a Python program that uses the LangChain framework. It will have the following features:
>
> 1. Get input from the user about a topic.
> 2. Select the type of content to create: blog or social media post.
> 3. Use LangChain to access the OpenAI API to create either the blog or social media post based on the topic.

When I tried this in ChatGPT, the results were, well, meh. Sure, it nailed the basic Python code structure for user input—no shocker there, since ChatGPT is sharp with Python. But with the LangChain code? Way off base. It goofed up the LangChain library import and missed some other things, too. Plus, it was clueless about how to call the OpenAI API.

Bottom line: Browse with Bing is mostly handy for your everyday, general questions—at least for now.

Tedious Tasks

Software development can easily get complicated with lots of steps. You've got planning, coding, and testing to name just a few. One part that can be a real drag is the repetitive coding. It's dull and eats up a ton of time. Developers often end up stuck writing the same routine code, squashing bugs, and looking over code. True, this is all important, but it's not exactly the fun, creative part of making software. Besides, trying to keep up with new programming languages, frameworks, and tools can be overwhelming.

AI tools like ChatGPT can take the edge off those boring bits of software development. In the next few sections, we'll dive into some of these areas.

Regular Expressions

Regular expressions, or *regex*, are a nifty tool for working with text. Think of them like a powerful find-and-replace functionality that can spot patterns in strings of characters. But here's the kicker: regex is like learning a whole new language of symbols and commands. It can be a brain twister! Trying to remember all those weird combinations of backslashes, dots, brackets, and who-knows-what-else is enough to make your head spin. Sometimes, it feels like you need a secret decoder ring just to make sense of it all.

For example, take a regex pattern like this:

```
^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$
```

This would be used to validate a strong password, ensuring it has at least eight characters, including one uppercase letter, one lowercase letter, one number, and one special character.

You probably wouldn't enjoy writing that, but ChatGPT can create a regex. Here's a look at some common examples:

> *Prompt:* Create a regex pattern to validate standard email addresses. The email should start with alphanumeric characters, followed by optional dots, hyphens, or underscores. Then, there should be an "@" symbol, followed by more alphanumeric characters, a dot, and finally, a domain name that is 2 to 6 characters long.
>
> *Regex pattern:* ^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$
>
> *Prompt*: Provide me with a regex pattern to validate URLs. The URL may start with "http" or "https", followed by "://", then a domain name which can include alphanumeric characters and dots. After the domain, there can be an optional path that starts with a "/" and can include alphanumeric characters, slashes, dots, or hyphens. The URL may end with an optional "/".
>
> *Regex pattern:* ^(https?:\/\/)?([\da-z.-]+)\.([a-z.]{2,6})([\/\w .-]*)*\/?$
>
> *Prompt*: Create a regex pattern to validate dates in the MM/DD/YYYY format. The month should be between 01 and 12, the day should be between 01 and 31, and the year should be either in the 1900s or 2000s.
>
> *Regex pattern:* ^(0[1-9]|1[0-2])\/(0[1-9]|[12][0-9]|3[01])\/(19|20)\d\d$
>
> *Prompt:* I'm looking for a regex pattern to validate IPv4 addresses. Each octet should be a number between 0 and 255, and the octets should be separated by dots. Create a regex pattern for this.
>
> *Regex pattern:* ^(25[0-5]|2[0-4]\d|[0-1]?\d\d?)\.((25[0-5]|2[0-4]\d|[0-1]?\d\d?)\.){2}(25[0-5]|2[0-4]\d|[0-1]?\d\d?)$

ChatGPT can also help you decipher regex patterns. Here's a sample prompt:

> *Prompt:* What is this regex? ^(\(\d{3}\)\s?|\d{3}[-.])\d{3}[-.]\d{4}$

The response is that this is designed to match the North American phone number format.

Starter Code

Starter code is the initial code snippets or templates that give you a head start. No doubt, it can be a big time saver. But starter code can be a learning tool as well. If you're new to coding, poking around some starter code can help you figure out how things work. What's more, starter code helps keep things consistent, especially in big teams where everyone needs to be on the same page.

Here's an example:

```
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
 res.send('Hello, World!');
});
app.listen(port, () => {
 console.log(`Server running at http://localhost:${port}`);
});
```

This provides a concise and clear example of a basic web server created using Express.js, a popular web framework for Node.js. It is good starter code because of its simplicity and use of core elements. Table 6-2 shows some prompts for starter code.

Table 6-2. Prompts for starter code

Programming language/framework	Prompt
Node.js/Express.js	"Provide an example of a basic Express.js server."
React (JavaScript)	"How do I create a simple React component?"
JavaScript	"Can you show me how to define a simple function in JavaScript?"
Android (Java)	"What is the starter code for a basic Android Activity in Java?"
Python/Flask	"Can you give me an example of a basic Flask application?"
Vue.js (JavaScript)	"Show me how to set up a basic Vue instance."
Django (Python)	"How can I start a new Django project with a simple view?"
Swift (iOS)	"What's the starter code for a simple UIViewController in Swift?"

GitHub README

The GitHub *README file* is essentially the welcome mat for a repository. It's where you find out what the project's all about—its purpose, what it does, and how to use it. Usually called *README.md*, it's written in Markdown, which is for making things look nice and be easy to read. You'll see the README on the main page of the repo.

A well-written README is important because it makes everything smoother for anyone checking out your project. It sets the tone and helps to make the project clearer, whether a user wants to use the project or wants to contribute to it. The README will also lay down the rules and what to expect.

Writing a README is not always a walk in the park. You have to find the sweet spot between giving all the juicy details and keeping it concise. Developers, who are focused on their coding, might skip over important parts, not realizing that what's obvious to them isn't obvious to everyone.

For crafting a good README, you do need some writing chops. But this is not necessarily the strong point of a developer. Moreover, as your project grows and changes, keeping the README up-to-date is a whole thing on its own. Yes, it can be a juggling act.

But with ChatGPT on your side, whipping up a README can be a breeze and the content should be pretty solid. Let's take an example. Imagine you've just built this awesome app for digging up recipes. Here's a prompt to kick things off:

> *Prompt:* Write a GitHub readme for my project named "Recipe Finder." This project is developed using Vue.js and helps users find recipes based on the ingredients they have. Users need to input the ingredients they have, and the app will return a list of recipes they can cook with those ingredients. To kickstart, simply clone the repository, run npm install, insert your API key into the .env file, and execute npm start to launch the app.

ChatGPT creates an introduction to the project and then includes sections like Features, Getting Started, Usage, Contributing, License, and Acknowledgments.

For some sections of the README, ChatGPT made assumptions. For example, it specified that the project uses the MIT License. But you can continue to prompt ChatGPT to provide more details about the project.

> If you do not know what open source license to use, check out Choose a License (*https://choosealicense.com*), which provides assistance with this question.

Cross-Browser Compatibility

Cross-browser compatibility is about making sure your website or web app works right on different browsers. It's important because your users likely access your site or app using a whole mix of browsers like Chrome, Firefox, Safari, and Edge, all of which have various versions.

Let's show how ChatGPT can help you tackle this challenge. Table 6-3 shows some useful prompts.

Table 6-3. Prompts for browser compatibility

Category	Prompt
HTML5 and CSS3 features	"I want to use the placeholder attribute in my input fields, but it doesn't work in older versions of Internet Explorer. How can I create a fallback?"
CSS prefixes	"I want to use the transition property in my CSS, but I want to ensure it's compatible across all browsers. How should I write it?"
JavaScript functionality	"I am using the fetch function in JavaScript to make HTTP requests, but it's not supported in Internet Explorer. What can I do?"

Bash Commands

Instructions you type into a terminal to interact with your computer system are known as *bash commands*. They let you do many sorts of tasks, like move around in your files, keep tabs on what your computer's doing, and manipulate files.

Getting the hang of bash commands can be tough, especially if you're new to the Unix/Linux world. The way some commands are written can be cryptic and hard to grasp at first. It takes a fair bit of time and practice to really get good at it. Furthermore, there are a lot of commands, each with their own bunch of options and twists. Sure, there are resources on the internet to help out. Yet even the pros can have a hard time remembering commands they don't use often or only recently picked up.

However, you can use ChatGPT as your virtual assistant for working with bash commands. Here are some example prompts:

> *Prompt:* How can I use a bash command to list all the files in a directory sorted by modification time?
>
> *Prompt:* What is the bash command to count the number of lines in a file?
>
> *Prompt:* How can I create a directory and navigate into it using a single command?
>
> *Prompt:* How can I redirect the output of a command to a file?
>
> *Prompt:* What is the command to search for a specific process running on my system?
>
> *Prompt:* How can I compress a directory into a ZIP file using bash?
>
> *Prompt:* What command should I use to change the permissions of a file in bash?

GitHub Actions

GitHub Actions is a continuous integration and continuous deployment (CI/CD) tool that's baked into GitHub. It lets developers set up, customize, and run their software development workflows directly in their repos. Basically, a GitHub Actions workflow is a set of automated processes you define in a workflow file. The processes are made up of jobs, each doing a different thing like running commands, setting up processes, or running actions. They're written in YAML syntax and kick into gear when certain actions happen, like pushing code, making a new issue, or scheduling times. With GitHub Actions, developers can build, test, and deploy their code straight from GitHub, making their software development process smoother and more automated. You can share, reuse, or even tweak other developers' actions, which is great for teamwork and sharing know-how in the world of automated software development.

ChatGPT can help you create GitHub Actions. Here are some prompts:

> *Prompt:* Create a GitHub Actions workflow template to build and deploy a static website using Hugo, with deployment to GitHub Pages.
>
> *Prompt:* Generate a GitHub Actions workflow template for a Java project using Maven, including steps for compiling the code, running tests, and packaging the application.
>
> *Prompt:* Create a GitHub Actions workflow template for a Dockerized application, including steps for building the Docker image, pushing it to Docker Hub, and deploying it to a Kubernetes cluster.
>
> *Prompt:* Generate a GitHub Actions workflow template for a React Native mobile application, including steps for installing dependencies, building the app, and uploading the APK to a specified Google Drive folder.

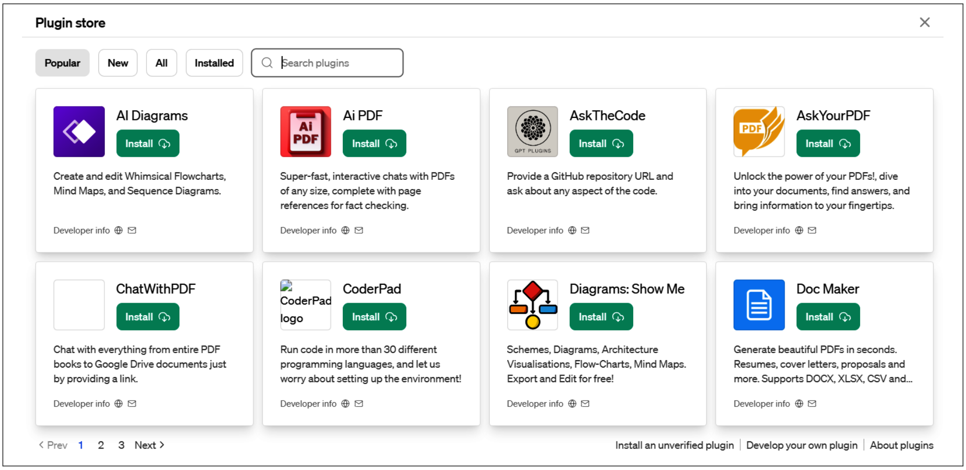
Plugins

ChatGPT plugins are mini apps that use the power of an OpenAI LLM. They open the door to using external databases and applications. It's like having your own little app store for iOS or Google Play.

To use plugins, you first need to make sure you have the feature activated. Go to your profile and select Settings and then Beta Features.

You can find the plugins by hovering over the GPT-4 icon—at the top of the screen—and then choosing Plugins. Click this, and you will get the screen shown in Figure 6-6.

At the top, you can navigate the plugins. You can filter for popular, new, all, and installed apps. There is also a search box.



Plugin store

Popular | New | All | Installed | Search plugins

AI Diagrams Install

Create and edit Whimsical Flowcharts, Mind Maps, and Sequence Diagrams.

Developer info

Ai PDF Install

Super-fast, interactive chats with PDFs of any size, complete with page references for fact checking.

Developer info

AskTheCode Install

Provide a GitHub repository URL and ask about any aspect of the code.

Developer info

AskYourPDF Install

Unlock the power of your PDFs!, dive into your documents, find answers, and bring information to your fingertips.

Developer info

ChatWithPDF Install

Chat with everything from entire PDF books to Google Drive documents just by providing a link.

Developer info

CoderPad Install

Run code in more than 30 different programming languages, and let us worry about setting up the environment!

Developer info

Diagrams: Show Me Install

Schemes, Diagrams, Architecture Visualisations, Flow-Charts, Mind Maps. Export and Edit for free!

Developer info

Doc Maker Install

Generate beautiful PDFs in seconds. Resumes, cover letters, proposals and more. Supports DOCX, XLSX, CSV and...

Developer info

‹ Prev 1 2 3 Next ›

Install an unverified plugin | Develop your own plugin | About plugins

Figure 6-6. The ChatGPT plugin store gives you access to mini apps that use the power of an OpenAI LLM

The Codecademy Plugin

Let's try out the plugin from Codecademy. First, press the Install button. To activate it, go to the top of the screen and click the down arrow. Then click on the Codecademy icon.

This plugin has two main functions. One is to allow users to find specific courses or paths based on their goals and experience level. For instance, users interested in AI and ChatGPT can ask the plugin for course recommendations, and it will provide a list of relevant courses, including descriptions and information on whether the courses are free or paid.

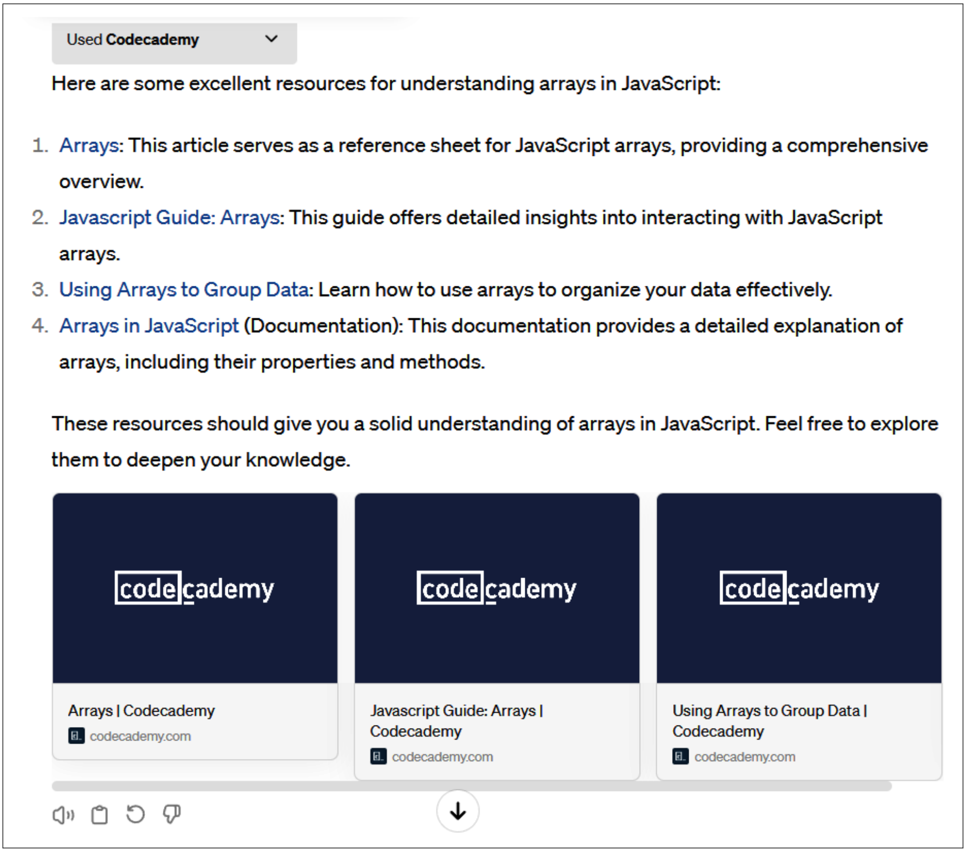
Next, the plugin serves as a quick reference tool for technical documentation, providing links to documents and articles with more detailed information.

We'll test this part out:

> *Prompt:* What is the best doc or article for explaining arrays in JavaScript?

ChatGPT provides the response shown in Figure 6-7.

At the top, an icon shows that the system is accessing the Codecademy plugin. Below is text that indicates there are many resources available on this topic. However, the plugin then narrows this down by providing links to three documents.



Here are some excellent resources for understanding arrays in JavaScript:

1. Arrays: This article serves as a reference sheet for JavaScript arrays, providing a comprehensive overview.
2. Javascript Guide: Arrays: This guide offers detailed insights into interacting with JavaScript arrays.
3. Using Arrays to Group Data: Learn how to use arrays to organize your data effectively.
4. Arrays in JavaScript (Documentation): This documentation provides a detailed explanation of arrays, including their properties and methods.

These resources should give you a solid understanding of arrays in JavaScript. Feel free to explore them to deepen your knowledge.

Figure 6-7. The Codecademy plugin responds to a request for resources on JavaScript arrays

The AskYourDatabase Plugin

Sheldon Niu came up with this plugin idea because he was often using ChatGPT to write SQL statements and it was a hassle. He had to explain the whole database schema each time, and then there was a bunch of copying and pasting the output to run it in a terminal. He thought, "Hey, what if ChatGPT could just chat directly with a database?" And that's how he got rolling on creating AskYourDatabase.

With this tool, you can breeze through prototyping a database schema using ChatGPT. Then, querying data becomes a piece of cake. Even better, you can skip those traditional business intelligence (BI) tools that usually need lots of setup.

Recombinant AI Plugin

Mark Zahm's a developer who started his own AI consulting business. He created the Recombinant AI plugin to make life easier for developers working with their GitHub and Gitlab repos. This tool lets ChatGPT get the gist of whole programs and sift through the nitty-gritty details. That way, users can get a solid grasp of their code, which helps when tweaking, analyzing, or blending their own ideas into software. Mark calls it a "conversational IDE," which is a pretty neat way to put it.

Here are some of its applications:

- Doing traditional tasks like storing code snippets or prompts
- Using the project and file system to create complex task lists and chain-of-thought prompts
- Saving important information regarding up-to-date programming libraries

GPTs

You can create your own custom ChatGPT. It's called a GPT, and it is very easy to put together—often taking just a few minutes.

Let's look at an example. We'll create a GPT for a software development style guide. This is packed with nuts-and-bolts guidelines like how to name your variables so they make sense to everyone, the right way to indent your code, and the specific programming patterns or practices the team follows. It's a dress code for your code.

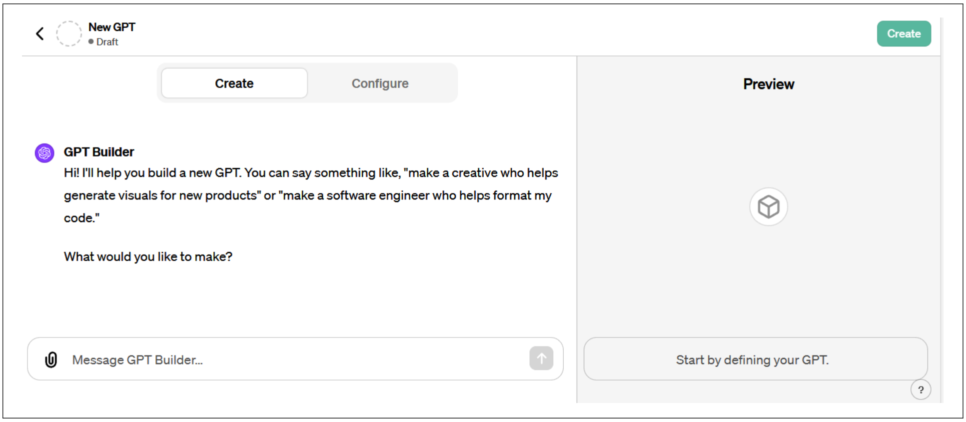
The idea is to make everything uniform and tidy. This makes it easier for everyone to read and understand the codebase, which can be a lifesaver during big projects.

But here's the catch: if you're new to this, it can be a bit overwhelming. You might be used to coding a certain way, and suddenly you've got to adapt to new standards.

This is where a GPT can help. So let's create one. First, on the home screen of ChatGPT—at the top left—select Explore and then choose Create a GPT. You will get a screen with two panels, as you can see in Figure 6-8.

The left panel is where you build the GPT, and the right one is a preview area. To have it work with a development style guide, your prompt would look like this:

> *Prompt:* Make a system that enforces a software development style guide.



GPT Builder

Hi! I'll help you build a new GPT. You can say something like, "make a creative who helps generate visuals for new products" or "make a software engineer who helps format my code."

What would you like to make?

Figure 6-8. The GPT Builder lets you create a custom GPT

ChatGPT will start to create the instructions for the GPT. This prompt is somewhat vague, but ChatGPT will ask some questions to get more information. It will first suggest a name for the GPT—it suggests "Code Stylist."

It will then create a profile image for the GPT by using DALL·E 3. But you could upload your own image instead.

Then it will ask for details like the language or languages, the practices, guidelines, rules, what should be avoided, and so on. Say you tell it the language is Python and give it some other requirements.

Next, on the top of the panel, select the Configure tab. You can then upload your style guide.

You can then go to the right panel to experiment with the GPT. If you ask it to write code, it will automatically work in Python and abide by your rules.

You can save this GPT by clicking the icon at the top right. It will then show up on your main screen. To use it, just click the GPT icon.

OpenAI has also created an appstore for GPTs. You navigate to this by selecting Explore GPTs at the top left of the screen of ChatGPT.

One of the categories is for programming. Here are some examples of the GPTs that are available:

- GPTavern is an innovative platform designed to teach coding through a unique approach called "prompt-gramming." It offers over 20 hotkeys to streamline coding processes and provides a collection of 75 starter projects for learning code. The platform is interactive, allowing users to ask questions, upload photos, and access a command menu and a README for guidance.

- DesignerGPT was created by Pietro Schirano and specializes in creating and hosting aesthetically pleasing websites.
- Screenshot To Code GPT, provided by godofprompt.ai, offers a feature that lets users upload screenshots of websites, which are then converted into clean HTML, Tailwind, and JavaScript code.
- Mindmap/Diagram/Chart—PRO BUILDER, offered by pyxl.ai, helps in visualizing code and databases. It enables users to create flowcharts, charts, and sequences with a user-friendly drag-and-drop editor, aiding in the understanding and organization of complex data structures.
- Code Guru was developed by Ryan J. Topps and offers a range of services including reviewing code, writing pull requests, generating and optimizing functions, writing tests, and commenting on existing code.

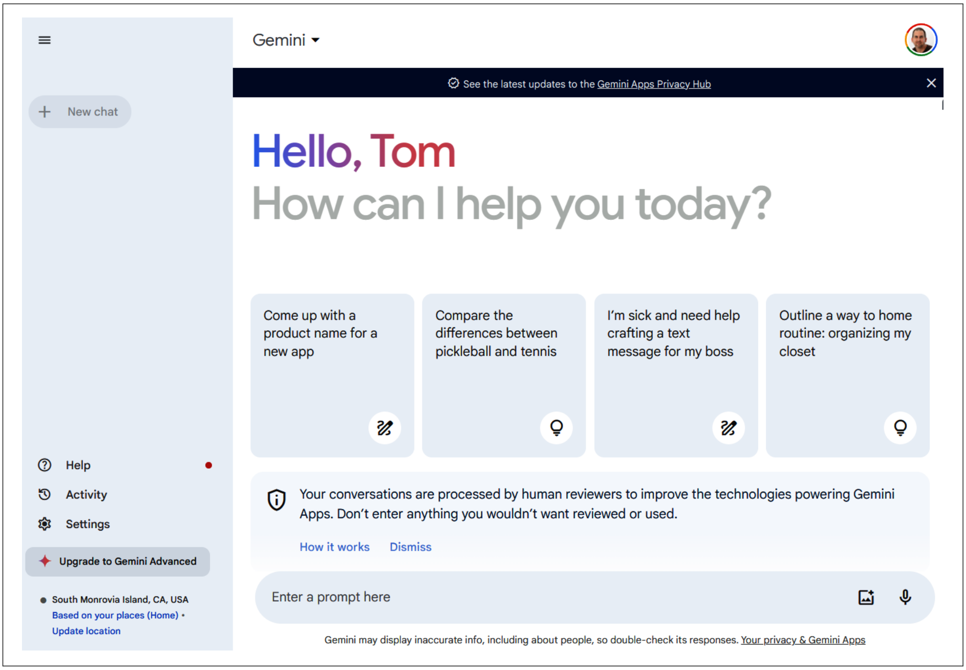
> You can include third-party APIs in your GPT. You do this by selecting Actions.

Gemini

When ChatGPT showed up, it caught Google off guard. Even with all its early moves and big-bucks investment in AI, Google wasn't ready for the generative AI wave. Google was wary about jumping into the new-fangled technology—worried about its accuracy and its effect on the company's lucrative ad business. But then, early in 2023, Google's stock took a dip, and that shook things up. The company decided to make a move and rolled out Bard, its own take on generative AI, kind of a comeback to ChatGPT. Bard had a few bumps at the start, but it got better, finding its way into different Google apps and helping the company's stock bounce back.

In February 2024, Google renamed the Bard platform Gemini. In addition to a free version, there's now also a premium offering, Gemini Advanced, which costs $19.99 per month after two free initial months. It uses Google's most sophisticated LLM, which is called Ultra 1.0. According to Google: "It can help you with more advanced coding scenarios, serving as a sounding board for ideas and helping you evaluate different coding approaches."

Google also launched a mobile version of Gemini for Android and iOS. With the app, you can type, talk, or use images to interact with the LLM. Figure 6-9 shows the interface for Gemini.



See the latest updates to the Gemini Apps Privacy Hub

Hello, Tom

How can I help you today?

Come up with a product name for a new app

Compare the differences between pickleball and tennis

I'm sick and need help crafting a text message for my boss

Outline a way to home routine: organizing my closet

Your conversations are processed by human reviewers to improve the technologies powering Gemini Apps. Don't enter anything you wouldn't want reviewed or used.

Enter a prompt here

South Monrovia Island, CA, USA
Based on your places (Home) •
Update location

Gemini may display inaccurate info, including about people, so double-check its responses. Your privacy & Gemini Apps

Figure 6-9. The user interface for Gemini allows you to manage chat sessions and enter text prompts, images, and speech-to-text input

Just like ChatGPT, on the left side of the screen, you've got your list of chat sessions. If you want to hide the list, just click on the hamburger icon above. For any chat, you can hit the little icon on the right to either change its name or delete it. Then there's this cool feature: you can pin chats you want to keep handy.

If you want to clear out all or just some of your items—based on when they happened—head over to the bottom left and click on Activity. Here, you can also tell Gemini if you want it to hang onto your prompts or just let them go.

In the middle of the screen, you'll find some helpful tips on how to use Gemini, as well as some security heads-ups. Down at the bottom is where you type in your prompts. And over on the right is your spot for uploading images. Gemini can pull out text, spot objects and scenes, answer questions about the pics, and even whip up some creative text formats.

The right side of the input box also has a microphone icon. This allows you to use speech-to-text to generate text for Gemini.

So what about plugins? Does Gemini have its own version? It does, and they are called extensions. Currently, they only are for Google applications like Flights, Hotels, Maps, Workspace, and YouTube; there are no third-party extensions.

Finally, Gemini has real-time access to the internet. Its responses will include citations so you can verify the source.

Applications

Productivity apps like Excel and Google Sheets aren't just for spreadsheets. They've got their own programming languages to amp up what they can do. Take Excel, for instance. It's got Visual Basic for Applications (VBA), which is a beefy programming system that lets you cook up your own scripts and programs. It's great for automating the same old tasks, making special functions, and analyzing your data. Excel also comes loaded with many ready-to-go functions, like PivotTables and Power Query, and it plays nice with Power BI for some top-notch data visuals and analysis.

Google Sheets, on the other hand, rolls with Google Apps Script, which is like JavaScript's cousin. It lets you do tasks similar to those you can do in Excel. But in addition, there are smooth integrations to Google apps like Drive and Gmail. Google Sheets is also packed with its own set of ready-to-use functions, and you can boost it with different add-ons from the Google Workspace Marketplace. Being entirely cloud based makes it great for teaming up with others, and it's a solid pick for web apps and group projects.

Then can Gemini help out? Definitely. For example, you can use it to explain a formula. No doubt, formulas can get quite complicated. Here's an example:

> *Prompt:* Explain the following Excel formula: =SUM(OFFSET(A1,1,0,COUNT(A:A), 1))

For Gemini, making sense of this is no problem.

Another use case is to create VBA scripts. Here are some sample prompts:

> *Prompt:* How can I create a custom form in Excel using VBA for entering user login details?
>
> *Prompt:* Can you help me write a VBA script for Excel to protect specific data with a password?
>
> *Prompt:* I need VBA code to validate email addresses and date formats in my Excel user login sheet. Can you assist?
>
> *Prompt:* Is it possible to send automated email notifications from Excel using VBA to get updates about user login information?

While all this is great, when it comes to working with spreadsheets, it's important to understand the relationships among the data, formulas, and scripts. Knowing how they dance together helps you build models that are powerful and don't trip up on errors. After all, if you tweak something in your data, the change can ripple through your formulas and mess with what your scripts output, and before you know it, your final results have done a total 180.

But Gemini can load and analyze Excel spreadsheets. You can then ask questions about them, such as to get a step-by-step understanding or to highlight a certain area of your data or logic.

Let's take an example. Suppose you have an Excel spreadsheet with data on book sales. Here are some example prompts:

> *Prompt:* Can you show me how to find the highest royalty amount earned by an author in Excel?
>
> *Prompt:* I want to calculate the inventory turnover rate for each book in Excel. What formula should I use?
>
> *Prompt:* How do I create a stock shortage alert in Excel, indicating when to order more based on the reorder level?
>
> *Prompt:* How do I calculate the percentage of positive reviews (ratings 4 or higher) for books in Excel?

Gemini for Coding

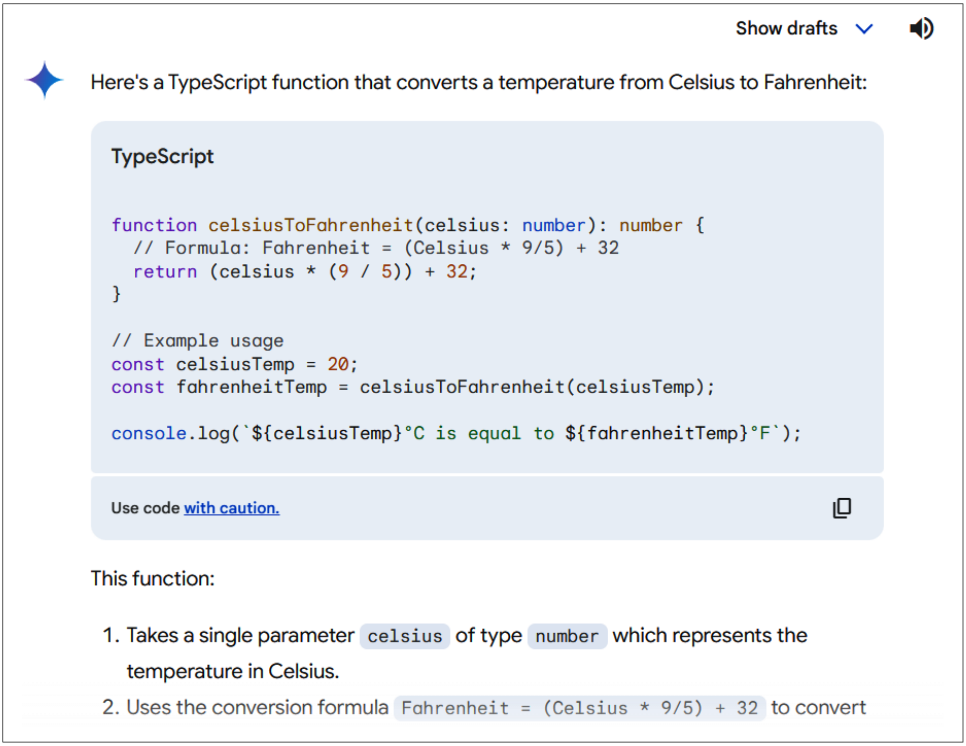
Gemini understands over 20 computer programming languages. So let's try out the system with this prompt:

> *Prompt:* Generate a TypeScript function that converts a temperature from Celsius to Fahrenheit.

Figure 6-10 shows the response from Gemini.

At the top right, there is a Show drafts option. If you click on this, you will find three versions of Gemini's response. This can be a way to get better ideas for your coding tasks.

As with ChatGPT, the response includes a code listing, which you can copy. Gemini also provides an explanation.



Here's a TypeScript function that converts a temperature from Celsius to Fahrenheit:

```
function celsiusToFahrenheit(celsius: number): number {
  // Formula: Fahrenheit = (Celsius * 9/5) + 32
  return (celsius * (9 / 5)) + 32;
}

// Example usage
const celsiusTemp = 20;
const fahrenheitTemp = celsiusToFahrenheit(celsiusTemp);

console.log(`${celsiusTemp}°C is equal to ${fahrenheitTemp}°F`);
```

This function:

1. Takes a single parameter `celsius` of type `number` which represents the temperature in Celsius.
2. Uses the conversion formula `Fahrenheit = (Celsius * 9/5) + 32` to convert

Figure 6-10. Gemini responds to a request to create a Typescript function

Figure 6-11 shows several ways to work with the response.

Output:

```
77
212
```

Figure 6-11. Click the icons on the bottom for different options to take actions on Gemini's response

Your options include:

Evaluation
: You can click thumbs-up or thumbs-down to provide feedback about the response.

Modify this response
: You can click the filter icon to access choices to make the response shorter, longer, simpler, more casual, or more professional.

Share
: You can share the response as a public link. You can also export it to Google Docs or for a Gmail draft.

Double-check the response
: When you click the Google logo, Gemini will review the response for accuracy. It may provide links for support.

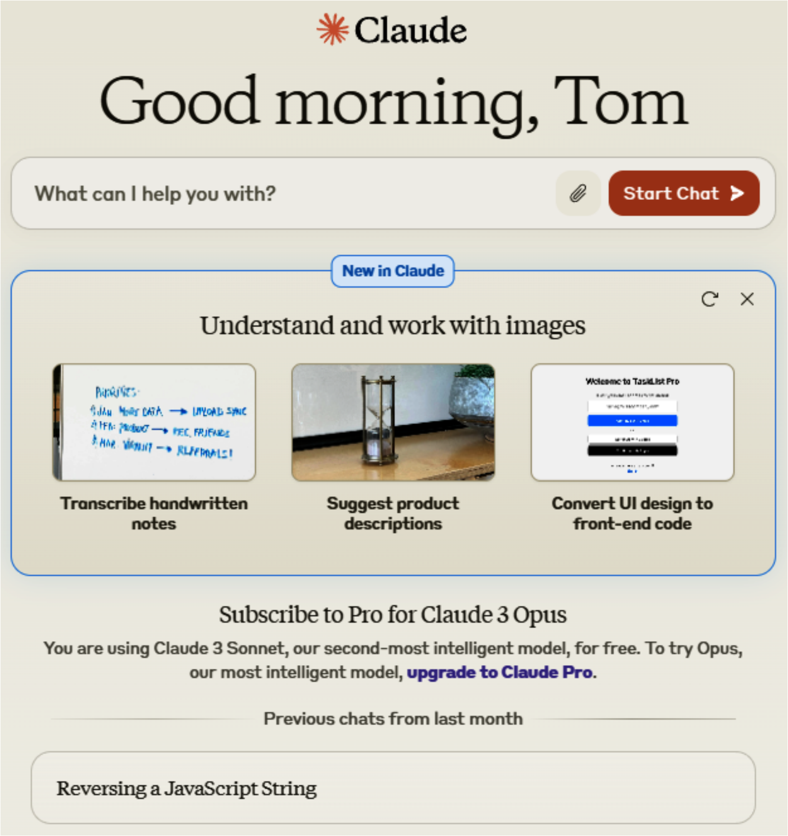
Claude

Back in 2021, the brother–sister team of Dario and Daniela Amodei kicked off their own startup called Anthropic. They used to be big shots at OpenAI, but they had the idea of creating a different kind of generative AI platform. Their vision was to focus on making AI safer, such as by minimizing bias. This became known as *constitutional AI*.

The Anthropic LLM has strong coding skills. It has scored 71.2% on the Codex HumanEval, which is a coding test for Python.

A major benefit of Claude is its context window, which is at 100,000 tokens. This means it can handle large code listings. You can also upload files like PDFs.

Figure 6-12 shows the initial screen for Claude.



Good morning, Tom

Understand and work with images

Subscribe to Pro for Claude 3 Opus

You are using Claude 3 Sonnet, our second-most intelligent model, for free. To try Opus, our most intelligent model, upgrade to Claude Pro.

Figure 6-12. The main screen for the Claude chatbot lists your prior activities and allows you to enter a prompt and upload files

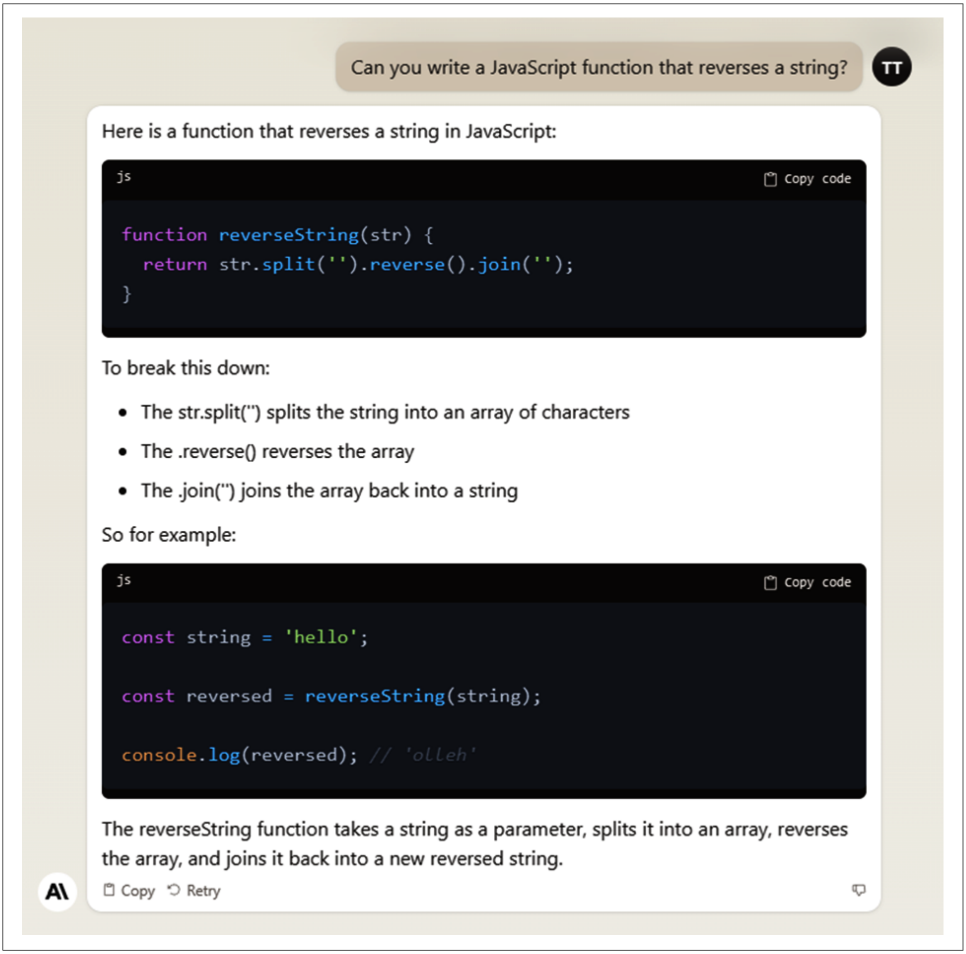
At the top, you can enter your prompt as well as upload up to five files. Below is a list of your prior activities, grouped by how many days ago they occurred.

Suppose we enter this:

> *Prompt:* Can you write a JavaScript function that reverses a string?

Figure 6-13 shows Claude's response.

Like ChatGPT, Claude offers a free tier. But you can also pay $20 per month for a premium version. With the upgrade, you get five times more usage capacity, priority access during high-traffic periods, and early access to new features.



Can you write a JavaScript function that reverses a string?

Here is a function that reverses a string in JavaScript:

```js
function reverseString(str) {
  return str.split('').reverse().join('');
}
```

To break this down:

- The str.split('') splits the string into an array of characters
- The .reverse() reverses the array
- The .join('') joins the array back into a string

So for example:

```js
const string = 'hello';

const reversed = reverseString(string);

console.log(reversed); // 'olleh'
```

The reverseString function takes a string as a parameter, splits it into an array, reverses the array, and joins it back into a new reversed string.

Figure 6-13. Claude responds to a request for a JavaScript function

Conclusion

In this chapter, we took a tour through some of the mega LLMs like ChatGPT, Gemini, and Claude. We checked out their cool features—stuff like code generation, plugins, surfing the web in real time, and handling big chunks of information. Sure, they might not be on par with specialized coding tools, but they still have a lot to offer when you want to speed up your software development. And as these LLMs keep getting better, they're gearing up to bring even more awesome AI-powered programming breakthroughs.

CHAPTER 7

Ideas, Planning, and Requirements

In this chapter, we'll dig into the important early stages of making software. This is where all the cool ideas come alive and where we plan them out so they're ready to go. We're going to use ChatGPT to brainstorm effectively and plan things down to the last detail. We'll look at market research, keep an eye on trends, and check out the competition. We'll also talk about whiteboarding, which is a fun, visual way to throw around ideas and make them better. Plus, we'll dive into how to plan your project, looking at methods like Agile and Scrum, and how to tweak them to fit your software project.

Brainstorming

Thousands of software products pop up every day, so it's no walk in the park to make something that really shines and stands out from the crowd. But with an LLM like ChatGPT, you can do some brainstorming.

First of all, you can have it provide different approaches. Here's a prompt:

> *Prompt:* What are techniques for brainstorming ideas for new products? Use the internet to research this question.

ChatGPT dished out 10 helpful approaches—here's a peek at a few of them:

> *ChatGPT:*
>
> - 5 Whys: You start with a problem and keep asking "why" it happened. It's handy for looking back and figuring out what went wrong.
> - Bad Idea: Think up a bunch of terrible ideas on purpose. Chatting about why they're so bad can actually spark clever solutions.
> - Starbursting: This is about questions, not answers. You use a star-shaped diagram and hit it with questions like: who, what, when, where, why and how.

Suppose you want to use the starbursting approach. You can learn more about this with the following prompt:

> *Prompt:* How would you use starbursting as a way to come up with a new idea for a software product?

ChatGPT gave a straightforward breakdown of the starbursting exercise as applied to this topic:

> *ChatGPT:*
>
> - Who: Think about who's going to use your software.
> - What: Get into what problem your software is solving. How is it different from what's already out there?
> - When: Figure out the best times to use your software, launch it, and do updates or fixes.
> - Where: What platforms or devices can people use the software on?
> - Why: Dive into why your software is a must-have. Why should people pick it over the others?
> - How: Think about how your software will work, how you'll build and sell it, and how you'll take care of your users when they need help.

Going through this process is a great way to get a clearer picture of your new software product idea.

Now, let's say you want to come up with a list of ideas. It's best to zero in on a specific category, know who you're targeting, and understand what problem you want to tackle. Here are a few prompts to get you started:

> *Prompt:* I want to develop a health-tracking app for seniors. The problem is the difficulty they face in monitoring chronic conditions and medication schedules. What features can be included to make the app senior-friendly and assist in managing their health more effectively?
>
> *Prompt:* The target audience is college students struggling with managing their finances. What features should a budgeting app include to help them track expenses, save money, and understand financial planning easily?
>
> *Prompt:* I want to create a language learning app for busy professionals who struggle to find time for regular classes. Can you suggest features that cater to their tight schedules and offer quick, effective language-learning sessions?

How good are the ideas that ChatGPT creates? Well, some Wharton School professors put it to the test in their MBA innovation class. They had the students come up with a dozen product or service ideas. Then, they did the same thing with ChatGPT, tapping into the GPT-4 model. Among the ideas tossed out were things like a dorm room–friendly chef kit and a comfy cushion for those hard classroom seats.

To test these ideas, the professors used an online purchase-intent survey. The question was: “How likely would you be to purchase based on this concept if it were available to you?”

On average, around 40% of the student ideas were fairly solid, and ChatGPT scored slightly higher at 49%. But the professors didn’t stop there. They focused on the top 10% of ideas. These are the ones with real potential to be game-changers. And guess what? Out of the 40 top-notch ideas, only 5 were from the students.

Market Research

Before diving into a software project, it’s smart to ask a few key questions: Is there actually a need for this tool? Are customers willing to shell out cash for it? And just how big is the market for it?

This is all about doing your market research. Sure, it’s not a crystal ball, but it definitely helps lower the risk of pouring your time and energy into something that might not take off.

The truth is, a lot of new products don’t make it. Clayton Christensen, a Harvard professor and best-selling author, did a lot of research on this and found out that about 80% of new products flop.

And, when it comes to software projects, it seems that number may be even higher. Table 7-1 shows some of the most notable examples.

Table 7-1. Failed software applications

Product/Service	Launch year	Failure reasons
Friendster	2002	Technical issues, scalability problems, overtaken by Facebook
Microsoft Zune	2006	Failed to compete with iPod, changing music market dynamics
Windows Vista	2007	High system requirements, compatibility issues, security prompts
Google Wave	2009	Complex interface, unclear purpose, lack of user adoption
Google+	2011	Numerous errors, inaccurate data, poor navigation capabilities
Vine	2013	Overshadowed by competitors like Instagram and Snapchat
Amazon Fire Phone	2014	High price, limited app selection, unappealing features
Quibi	2020	Short-form content format not embraced, high competition, COVID-19 impact

One of the most epic fails is the video game for Steven Spielberg’s 1982 movie *E.T. the Extra-Terrestrial*. Atari shelled out a whopping $21 million for the rights from Universal Pictures and dropped another $5 million on promoting it.

The game really flopped. It was super confusing and not much fun either. So, what happened? Atari only managed to sell about 1.5 million units. The remaining 2.5 million ended up in a landfill in New Mexico.

Let’s dive into how you can use ChatGPT for market research. First of all, we need to figure out the target audience. Think about who’s going to get the most out of this software. Let’s start brainstorming about who these users might be. Consider this prompt:

> *Prompt:* What are potential target demographics for a project management software program?

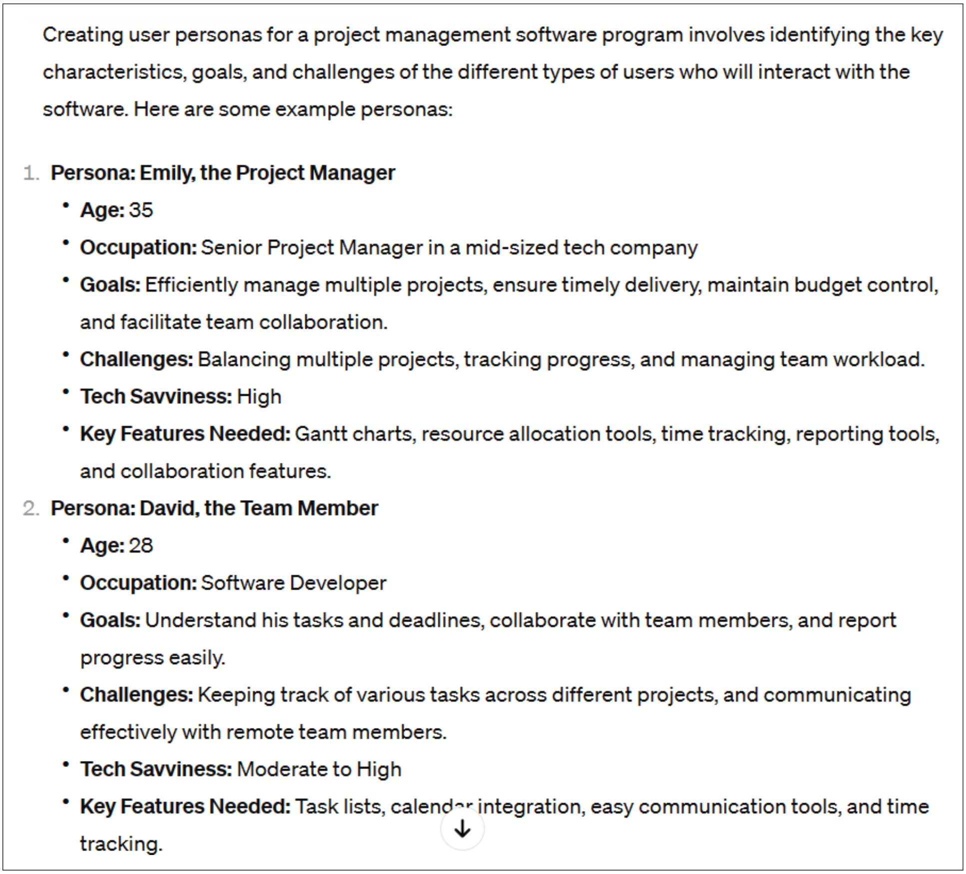
You can then have ChatGPT take on a task that’s important for getting to know your marketing: creating detailed user personas. Think of these personas as characters that give you snapshots of the different kinds of people who might use your product. Details often include how old they are, their interests, their jobs, and their lifestyles. By mapping out these personas, you start to understand who your customers are and what they’re looking for.

I asked ChatGPT for some user personas for management software, and you can see some of its responses in Figure 7-1.

ChatGPT can also be instrumental in drafting surveys and questionnaires to gather specific data.

> *Prompt:* Could you create a survey focused on understanding what buyers look for in project management software? The survey should include questions that help identify key features, usability preferences, specific industry needs, and any particular challenges buyers want to address with this type of software. Additionally, please include questions that gauge the importance of factors such as cost, support, scalability, and integration capabilities.

Let’s take a look at some of the other factors we need to consider when evaluating the market for software.



Creating user personas for a project management software program involves identifying the key characteristics, goals, and challenges of the different types of users who will interact with the software. Here are some example personas:

1. **Persona: Emily, the Project Manager**
    - **Age:** 35
    - **Occupation:** Senior Project Manager in a mid-sized tech company
    - **Goals:** Efficiently manage multiple projects, ensure timely delivery, maintain budget control, and facilitate team collaboration.
    - **Challenges:** Balancing multiple projects, tracking progress, and managing team workload.
    - **Tech Savviness:** High
    - **Key Features Needed:** Gantt charts, resource allocation tools, time tracking, reporting tools, and collaboration features.
2. **Persona: David, the Team Member**
    - **Age:** 28
    - **Occupation:** Software Developer
    - **Goals:** Understand his tasks and deadlines, collaborate with team members, and report progress easily.
    - **Challenges:** Keeping track of various tasks across different projects, and communicating effectively with remote team members.
    - **Tech Savviness:** Moderate to High
    - **Key Features Needed:** Task lists, calendar integration, easy communication tools, and time tracking.

Figure 7-1. ChatGPT created user personas for management software

Market Trends

When you're rolling out new software, digging into market research can make a big difference. It's like getting insider information on what users are really after, and it also gives you the scoop on what your rivals are up to. Think of it as your secret playbook. You get to spot trends before they blow up and opportunities that others might miss. Plus, it's a great safety net. It cuts down on the guesswork and helps you make smart choices that are more likely to hit the mark. And let's not forget about marketing. Knowing your audience inside out means you can create ads and promos that resonate with them, instead of just throwing stuff out there and hoping it sticks.

ChatGPT can be a good sidekick for your market research. Of course, the model is trained only up to a certain date, but there's a workaround: you can tell it to "research the internet" in your prompts.

Here are some examples:

> *Prompt:* Research the internet to find current trends in <*specific software category, e.g., mobile gaming, educational technology*> as of <*year*>.
>
> *Prompt:* Research the internet to provide an overview of emerging technologies in the software industry that are gaining popularity in <*year*>.
>
> *Prompt:* Research the internet to find the consumer behavior trends that are currently influencing the software market, particularly in <*target demographic or application type*>.
>
> *Prompt:* Research the internet to find the latest trends in user interface and user experience (UI/UX) design for software applications in <*year*>.
>
> *Prompt:* Research the internet to come up with a list of the top software applications in <*a specific field or category*> and explain what makes them successful in the current market.

Total Addressable Market

The *total addressable market*, or TAM for short, is an estimate of the maximum amount of money you could make if everyone who could use your product actually did. This helps you figure out where to focus your energy, how to position your product in the market, and what your growth game plan should be. It's a reality check that helps you set achievable goals for how much of the market you can grab and the cash you can bring in.

Venture capitalists (VCs) weight TAM heavily when they're figuring out where to put their money, since it shows how much growth potential there is. They're always on the hunt for places where their investment could take off, and a hefty TAM means there's a huge market just waiting to be tapped.

Usually, VCs look for a TAM that's over a billion dollars—and ideally much more than that. After all, even if a company only grabs a small piece of such a huge market, the profits can be impressive.

In response to a prompt like "What are the different ways to estimate the size of the TAM?" ChatGPT will respond similar to this:

> *ChatGPT:*
>
> - Top-down approach: You gather industry data and market research to get a sense of the whole market size. Then, you zoom in a bit, such as by focusing on the slice of the market you're aiming for.
> - Bottom-up approach: You figure out how many people or businesses might buy your product, guess how many of them will actually go through with it (your conversion rate), and then think about how much each customer will bring in. This approach takes more work, but it can give you a clearer, more tailored picture.

Let's take an example. Suppose you have a startup in the medical technology category:

> *Prompt:* SilverConnect Health is a health tech startup focusing on telemedicine services. The target market is in the United States, with a focus on elderly populations in urban areas. What is the current size of the telemedicine services market for SilverConnect Health? What are the projected growth rates for this market segment over the next 5–10 years?

ChatGPT searches the internet and comes up with some useful facts:

> *ChatGPT:*
>
> - Market size in 2022: The U.S. telemedicine market was valued at more than USD 35 billion in 2022. Another estimate puts the telemedicine market size at USD 91.2 billion in the same year.
> - Projected growth rates: One estimate suggests a compound annual growth rate (CAGR) of 12.5% from 2023 to 2032. Another source projects a CAGR of 22.9% from 2023 to 2030.

Competition

There are several key benefits of competitive analysis. First of all, it gives you a sneak peek into the market. You get to see what's already out there, what's working, and what's not. This helps you spot any gaps that your software can fill, making sure you bring something fresh to the table.

Then, there's learning from your competitors—both from their wins and their oops moments. This can really up your game.

Also, when you know what makes your software different, you can get specific with your marketing, hitting the right people more effectively.

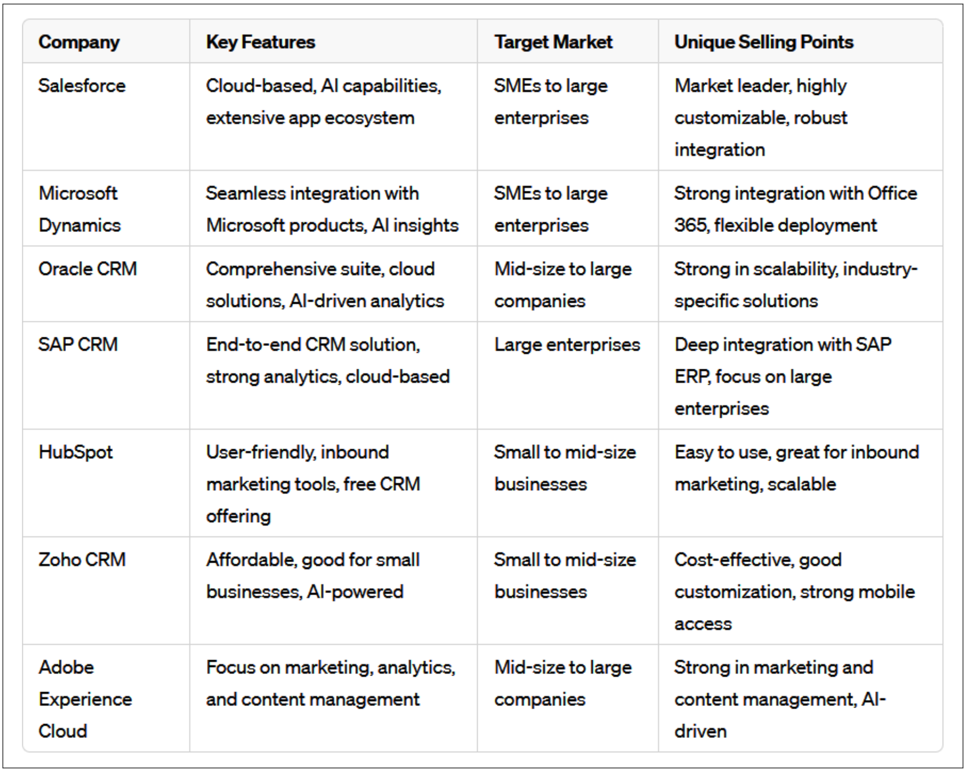
As for ChatGPT, it can be helpful with competitive analysis. Here are some prompts:

> *Prompt:* List the main competitors for <*your software type*> in the <*specific industry or market*>.
>
> *Prompt:* Compare the features of <*your software*> with its top three competitors.
>
> *Prompt:* What are the pricing models used by competitors of <*your software*> in the market?
>
> *Prompt:* How do the leading competitors position themselves in the market for <*type of software*>?
>
> *Prompt:* Identify the strengths and weaknesses of <*competitor's name*> software.
>
> *Prompt:* Summarize the customer reviews for <*competitor's software name*> focusing on user satisfaction and pain points.

You can also have ChatGPT put the analysis in a table, which can make it clearer:

> *Prompt*: Can you provide a summary in bullet points of the key players in the customer relationship management (CRM) software market as of 2023? Put the results in a table.

Figure 7-2 shows the response.



Company	Key Features	Target Market	Unique Selling Points
Salesforce	Cloud-based, AI capabilities, extensive app ecosystem	SMEs to large enterprises	Market leader, highly customizable, robust integration
Microsoft Dynamics	Seamless integration with Microsoft products, AI insights	SMEs to large enterprises	Strong integration with Office 365, flexible deployment
Oracle CRM	Comprehensive suite, cloud solutions, AI-driven analytics	Mid-size to large companies	Strong in scalability, industry-specific solutions
SAP CRM	End-to-end CRM solution, strong analytics, cloud-based	Large enterprises	Deep integration with SAP ERP, focus on large enterprises
HubSpot	User-friendly, inbound marketing tools, free CRM offering	Small to mid-size businesses	Easy to use, great for inbound marketing, scalable
Zoho CRM	Affordable, good for small businesses, AI-powered	Small to mid-size businesses	Cost-effective, good customization, strong mobile access
Adobe Experience Cloud	Focus on marketing, analytics, and content management	Mid-size to large companies	Strong in marketing and content management, AI-driven

Figure 7-2. ChatGPT provides a competitive analysis for the CRM market and formats it in a table

ChatGPT can even do a SWOT (strengths, weaknesses, opportunities, and threats) analysis. This helps you play up your strong points, fix or work around weak spots, grab chances that come by, and dodge any curveballs. Businesses of all types use it, and it's even useful for personal career moves.

Requirements

Requirements documents are critical for software development. Here are just a handful of their benefits:

Give you direction
: These documents lay out exactly what the software should do. Everyone, from developers to clients, gets what the plan is, which keeps things on track.

Keep everyone on the same page
: They make sure everyone involved, like your team and your clients, understands what's expected. This way, there's less chance of mix-ups or wrong turns.

Help with planning
: Need to figure out how long things will take or how much they'll cost? These documents are your go-to for planning out the work, time, and money needed.

Improve communication
: They're great for clear communication, especially in big teams or when people are geographically spread out.

Reduce risks
: Spotting potential problems early on? That's what these documents help you do, so you can dodge delays and keep costs under control.

Of course, coming up with software requirements isn't easy, and a big reason why is that software development can get complicated. Then there are the challenges of coordinating the work and thinking of many team members, who may be in other countries and time zones. Each group has its own ideas and needs, and sometimes these don't exactly match up or aren't clear.

Another headache is that what people want from the software can keep changing. This is especially true when you're working in a fast-paced industry. So, you're often trying to hit a moving target with your requirements, which makes it tough to settle on something solid. In the meantime, you've got to make sure the requirements are doable and that you can verify that they're being met.

When requirements are off base, the results can be disastrous. Here's a look at a few examples:

Denver International Airport's automated baggage system (1995)
: The requirements were mind-numbingly complex, and the goals were overambitious. This led to a 16-month delay and skyrocketing costs. In the end, the airport decided to ditch the whole system.

FBI's virtual case file system (2000–2005)

The FBI had only a vague idea of what it wanted. So yes, the system wound up not being of much use. In the end, the agency poured over $170 million into the project only to throw in the towel. The next project, called Sentinel, had much more specific requirements and, of course, was more successful.

HealthCare.gov launch (2013)

The federal government didn't nail down what it needed and didn't test it enough, so when it launched, it just couldn't handle the traffic and wasn't working right. The launch was a flop, and a lot of people had to put in a lot more work and money to fix it and get it running smoothly.

In the world of software development, there's an array of different requirements, each serving a unique purpose. You've got functional requirements that lay out what the software should do, like the tasks it should perform or the features it needs. Then there are non-functional requirements, which are about how the software should operate, such as with performance speed, security, and usability.

To explore how generative AI can help, we'll focus on two types of important documents: the *product requirements document* (PRD) and the *software requirements specification* (SRS).

Product Requirements Document

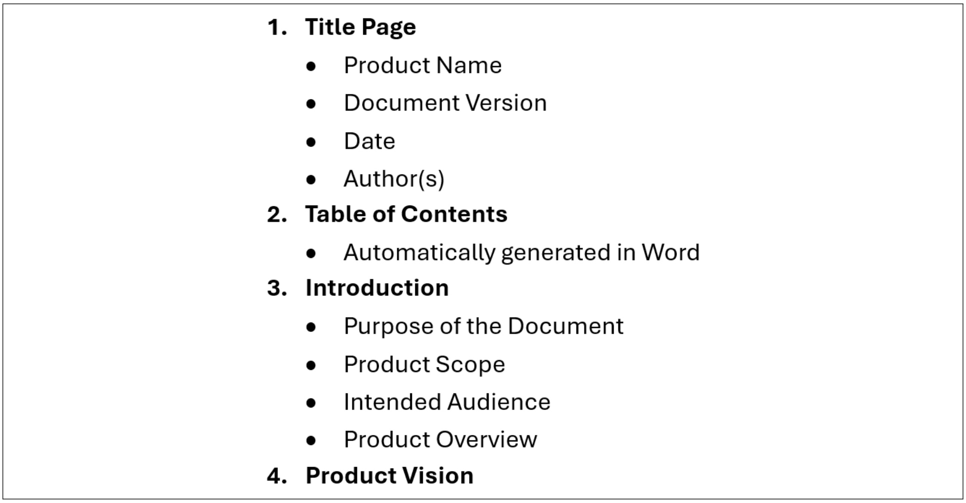
A PRD sets forth what your software will be and do. Think of it as a story, outlining what the application should look like, how it should behave, who's going to use it, and what problems it's going to solve. It is often detailed, covering everything from the must-have features to the user experience.

The audience for the PRD is diverse. It is for developers, designers, project managers, and the quality assurance team. It's also crucial for executives or other stakeholders who need to understand the product's vision and goals. Moreover, it can be useful for marketing teams to get an understanding of how to position the product. Essentially, it's for anyone involved in bringing the product to market and making sure it hits the mark with its intended users.

If you haven't seen a PRD, ChatGPT can help out:

> *Prompt:* What are the main parts of a PRD? Create a Word document as a template.

Figure 7-3 shows the document ChatGPT produced.



1. **Title Page**
    - Product Name
    - Document Version
    - Date
    - Author(s)
2. **Table of Contents**
    - Automatically generated in Word
3. **Introduction**
    - Purpose of the Document
    - Product Scope
    - Intended Audience
    - Product Overview
4. **Product Vision**

Figure 7-3. ChatGPT created a sample outline of a PRD

Software Requirements Specification

How is the SRS different from the PRD? The SRS is like the nitty-gritty technical manual for a software project. Unlike the PRD, which takes on the perspective of end users or customers and focuses on what they need and why, the SRS goes deep into the specifics of what the software needs to do and how it should do it, focusing on the technical aspects.

If you have a simple app, the SRS could be just a few pages. But with more complex projects—say for an enterprise software implementation—it can easily run hundreds of pages.

Let's take a look at some prompts:

> *Prompt:* What are the essential elements and best practices to consider when drafting an effective SRS?
>
> *Prompt:* Outline the process for developing an SRS, emphasizing the steps involved in gathering and analyzing requirements, stakeholder collaboration, and documentation methodologies.
>
> *Prompt:* Create a detailed SRS for a software project, including sections like Project Overview, Stakeholder Analysis, Functional and Non-functional Requirements, Assumptions, and Constraints. <*Add the details you want for each.*>

Interviews

Voice recognition software has been around for several decades, but for the most part, it was more of a novelty than a practical tool. Usually, it was impractical for everyday use or professional settings, as it couldn't reliably understand natural, conversational speech.

However, during the past few years, voice recognition has taken a leap forward, driven by the advances in AI. Modern systems are powered by sophisticated AI algorithms that are not only better at understanding a wide range of accents and dialects but also can grasp the context and nuances of spoken language. These systems learn and adapt over time, constantly improving their accuracy.

One way this can be helpful is during the interviews you often conduct to put together requirements documents.

For example, with voice recognition software, you can improve the following:

Accuracy and playback
While transcription accuracy is usually pretty good, you can use playback for corrections.

Speaker identification
Voice recognition can differentiate speakers in a meeting. This can help bring clarity to the requirements.

Key topics identification
Voice recognition software can highlight frequent keywords in the transcript, making it easier to identify and focus on main topics or requirements.

Summaries of key points
Voice recognition software can produce summaries of key points and suggest action items, aiding developers in pinpointing critical requirements and converting them into specific project tasks.

Some of the top voice recognition software systems include:

- Otter AI
- Fathom
- Rev
- Sonix
- Notta

Whiteboarding

Whiteboarding is a common practice when putting together requirements for software projects. It can help with the brainstorming process and allow everyone to chip in. It allows for collecting lots of different ideas and viewpoints, which is key for nailing down what the software should do. It's also hands-on—you can sketch out flowcharts or diagrams during the discussion, making complex stuff easier to get.

Well, ChatGPT can help with this. How? It can decipher images that you upload to the system.

For example, suppose you sketch out a login system, as shown in Figure 7-4.

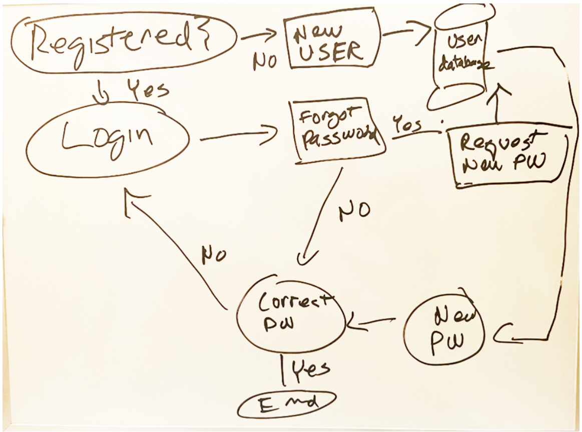


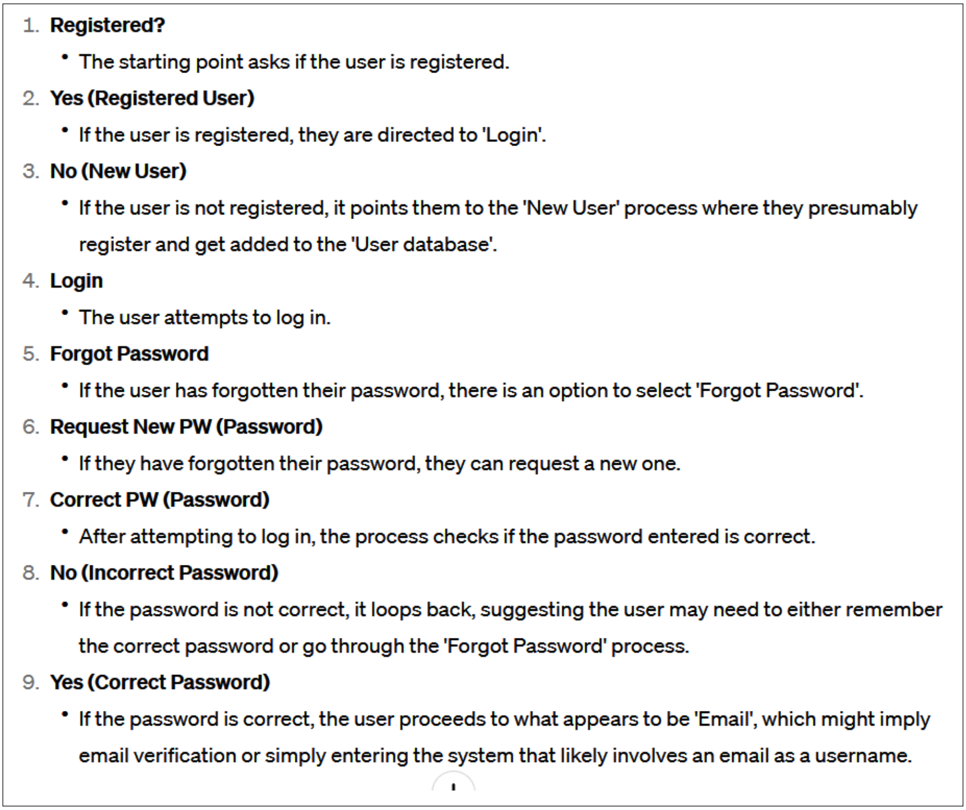
Figure 7-4. A sketch of a login system can be input to ChatGPT

As you can see, my handwriting is kind of messy. I also use abbreviations, such as "pw" for "password."

Let's see if ChatGPT can figure it out. I used this prompt:

> *Prompt:* Analyze the diagram.
>
> *ChatGPT:* The diagram you've uploaded appears to be a flowchart related to user account management, specifically for a login process on a website or application.

It then set forth the whole process (Figure 7-5).



1. **Registered?**
   - The starting point asks if the user is registered.
2. **Yes (Registered User)**
   - If the user is registered, they are directed to 'Login'.
3. **No (New User)**
   - If the user is not registered, it points them to the 'New User' process where they presumably register and get added to the 'User database'.
4. **Login**
   - The user attempts to log in.
5. **Forgot Password**
   - If the user has forgotten their password, there is an option to select 'Forgot Password'.
6. **Request New PW (Password)**
   - If they have forgotten their password, they can request a new one.
7. **Correct PW (Password)**
   - After attempting to log in, the process checks if the password entered is correct.
8. **No (Incorrect Password)**
   - If the password is not correct, it loops back, suggesting the user may need to either remember the correct password or go through the 'Forgot Password' process.
9. **Yes (Correct Password)**
   - If the password is correct, the user proceeds to what appears to be 'Email', which might imply email verification or simply entering the system that likely involves an email as a username.

Figure 7-5. ChatGPT analyzed an image of a diagram for a login system, translating it into an easy-to-read breakdown of the workflow

Tone

There's an art to writing effective PRDs and SRSs. It's important to strike the right tone. These documents should be clear, concise, and professional.

Let's break down why the writing style matters:

Making it crystal clear
: These documents are like the holy grail for your project. You want everything to be spot-on so that everyone gets what's going on. This way, you dodge those pesky miscommunications and development blunders.

Sticking to the facts
: PRDs and SRSs aren't the place for personal or stylistic flair. You need to be like a news reporter here—just the facts, please. Keeping it straightforward and neutral ensures that you're focusing on what the product really needs.

Consistency is key

You want to keep the same tone from start to finish. Think of it as following a recipe. If you start switching things up halfway, you're going to end up with a mess. Consistency makes these often hefty documents easier to digest.

Keeping the focus on the product

Remember, the star of the show here is the product and what it's supposed to do. Keeping a direct tone ensures you don't stray from the main plot—that is, the functionality and requirements.

When it comes to an LLM like ChatGPT, writing is one of its strengths. It can really up your game. And yes, you can instruct the LLM to hit the right tone, as with this prompt:

> *Prompt:* Please draft a comprehensive list of technical requirements for a mobile banking app, ensuring the writing is clear, well structured, and professional.

There are tons of AI writing tools on the market, each with its own special features to suit whatever kind of writing you're into. Take Jasper and Writer, for example. They're great for churning out content and keeping your brand voice consistent. Then you've got Grammarly, which is the go-to for nailing grammar and sprucing up your writing style. AI21 and Copy.ai are also in the mix, making content creation a breeze.

Consider Jigyasa Grover, who uses Grammarly for her projects. An expert in machine learning, she wrote a book on the topic, *Sculpting Data for ML* (2021). According to her:

> Grammarly is another tool that aids me in drafting technical design documents. I leverage it for proofreading, grammar, punctuation, and style suggestions. It integrates seamlessly with many web browsers and word-processing software. The language fabrication is clear and concise and even customizes suggestions based on the target audience and purpose of the document.

Approaches to Project Planning

After you have put together the requirements, the next step is to plan how you'll approach the project. This is about laying out the game plan for how your team will tackle the software creation journey. It involves setting clear goals and figuring out what you need to do, who's doing what, when it's happening, and how. The main objective is to keep things running smoothly, make sure everyone's productive, and, in the end, have a product that does what it's supposed to do and makes everyone, especially the customers, happy.

Now, the world of software development is certainly complex, so there are a bunch of different ways that teams can plan their projects. They each have varying strategies, each with their own rules, and steps to help people handle their tasks, deal with any changes, keep the quality up, and deliver something awesome to their clients. So whether your team needs everything spelled out in detail or prefers a more go-with-the-flow kind of plan that can change on the fly, there's a method out there for you.

Some common project-planning approaches are waterfall, Agile, Scrum, Kanban, and Extreme Programming (XP). Picking the right one is important because it can make or break your project. Your choice depends on many factors like how big and complex your project is, what your client needs, how your team works together, and how certain or uncertain things are in your project world. Some teams might stick to one method, but others might mix and match to find the perfect fit for their project.

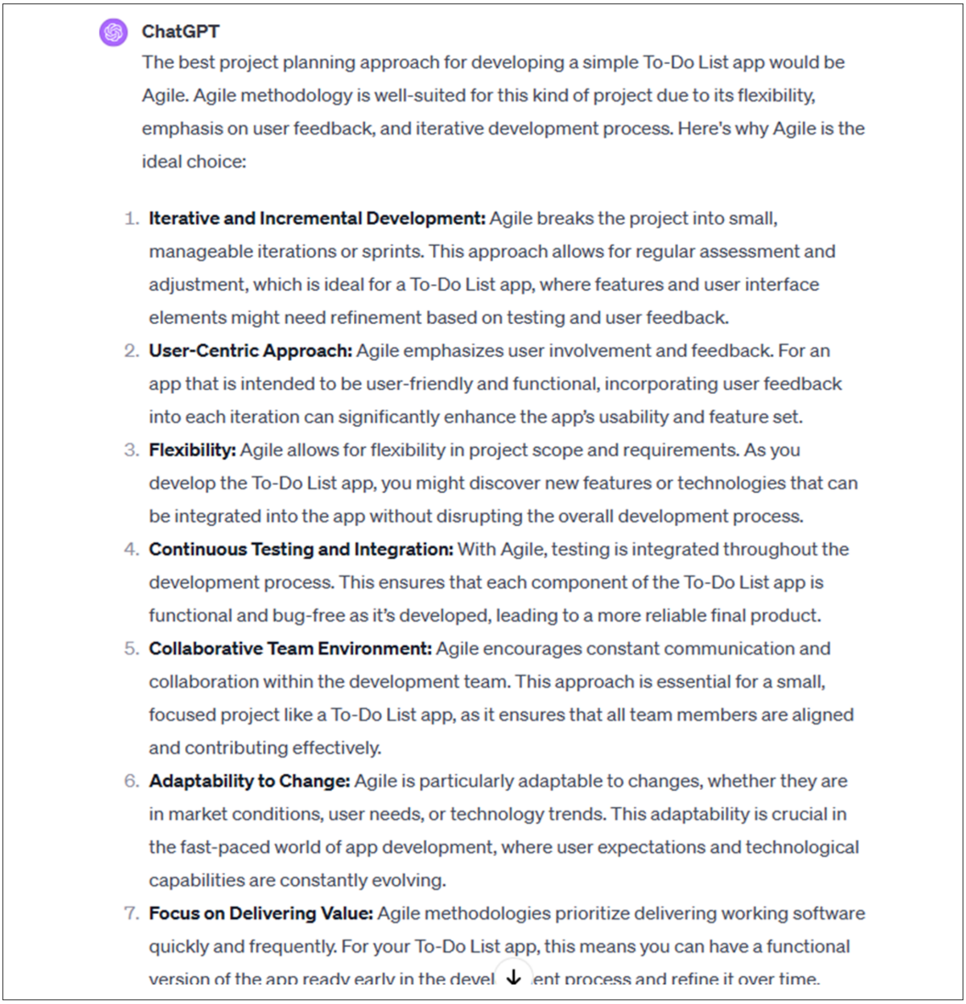
As for prompts, let's take an example. Suppose you are building a "to-do list" app. You have created a one-page requirements document for it, and now you ask ChatGPT:

> *Prompt:* What is the best project planning approach for this app?

ChatGPT suggests the Agile method as the best option and provides reasons (Figure 7-6).

Here's a look at some other prompts you might use:

> *Prompt:* How can teams effectively select and tailor a project-planning approach to fit their specific software project?
>
> *Prompt:* Discuss the importance of aligning a project-planning approach with the goals and complexity of a software project.
>
> *Prompt:* Explain the sequential phases of the waterfall model in software project management.
>
> *Prompt:* Describe the core principles of Agile methodology in software development.
>
> *Prompt:* How does Agile methodology facilitate flexibility and customer involvement in a project?
>
> *Prompt:* Outline the roles and ceremonies involved in the Scrum framework.
>
> *Prompt:* Compare and contrast Scrum and traditional project management approaches.
>
> *Prompt:* List the key practices of Extreme Programming and how they contribute to customer satisfaction.
>
> *Prompt:* Discuss the benefits and challenges of implementing test-driven development as part of XP.



ChatGPT

The best project planning approach for developing a simple To-Do List app would be Agile. Agile methodology is well-suited for this kind of project due to its flexibility, emphasis on user feedback, and iterative development process. Here's why Agile is the ideal choice:

1. **Iterative and Incremental Development:** Agile breaks the project into small, manageable iterations or sprints. This approach allows for regular assessment and adjustment, which is ideal for a To-Do List app, where features and user interface elements might need refinement based on testing and user feedback.
2. **User-Centric Approach:** Agile emphasizes user involvement and feedback. For an app that is intended to be user-friendly and functional, incorporating user feedback into each iteration can significantly enhance the app's usability and feature set.
3. **Flexibility:** Agile allows for flexibility in project scope and requirements. As you develop the To-Do List app, you might discover new features or technologies that can be integrated into the app without disrupting the overall development process.
4. **Continuous Testing and Integration:** With Agile, testing is integrated throughout the development process. This ensures that each component of the To-Do List app is functional and bug-free as it's developed, leading to a more reliable final product.
5. **Collaborative Team Environment:** Agile encourages constant communication and collaboration within the development team. This approach is essential for a small, focused project like a To-Do List app, as it ensures that all team members are aligned and contributing effectively.
6. **Adaptability to Change:** Agile is particularly adaptable to changes, whether they are in market conditions, user needs, or technology trends. This adaptability is crucial in the fast-paced world of app development, where user expectations and technological capabilities are constantly evolving.
7. **Focus on Delivering Value:** Agile methodologies prioritize delivering working software quickly and frequently. For your To-Do List app, this means you can have a functional version of the app ready early in the devel ↓ ent process and refine it over time.

Figure 7-6. ChatGPT explains why Agile is the best approach for development of a certain app

Test-Driven Development (TDD)

One of the smartest things you can do before you start coding is to map out your test cases. Remember the old saying "Measure twice, cut once"? It's good advice. You want to get ready before you dive in. A common approach is *test-driven development* (TDD).

Here's the deal: when you sit down and think about all the tests your code needs to pass, you get a much better understanding of what you need to build. You're taking a good, hard look at the blueprint before you start building. This way, you know exactly what your code should do, which saves you from a lot of head-scratching later on.

What's more, planning your tests first means you're thinking about the people who'll use your software right from the start. You're not just lost in code. You're focused on making something that works for users. It's better to spot potential headaches early on than to tear your hair out fixing bugs later.

In TDD, this approach is at the heart and soul of the whole thing. You write your tests before you code, which means you're always focused on what the code needs to achieve. This approach leads to cleaner, more straightforward code because you're always coding with a purpose.

With TDD, there's a toolkit for the different ways to write our tests:

Given-When-Then (GWT)

The Given is setting the stage, such as where everything's at before the action starts. Then you hit the When stage, the main event that gets things rolling. And you wrap up with Then, where you spell out what you expect to happen after the dust settles. It's like telling a story of what your code will carry out.

Arrange-Act-Assert (AAA)

This is similar to GWT. But GWT tends to be more narrative and user-centric, making it ideal for collaboration among developers, testers, and nontechnical stakeholders. AAA, on the other hand, is more about the technical execution of the test itself.

Setup-Exercise-Verify-Teardown (SEVT)

This is your go-to when you're in the trenches of integration and systems testing. You start with Setup, where you prep your testing ground. Then, you jump into Exercise, where you run your system through its paces. After this, you go to Verify, which is where you play detective to confirm everything worked out. And don't forget the Teardown. This is the cleanup step that's essential to keeping everything neat for next time.

Here's an example of a prompt for TDD, using GWT:

> *Prompt:* Formulate a testing approach for critical features using the Given-When-Then methodology. The specific requirements include:
>
> - Develop an online booking system for a chain of boutique hotels.
> - The system must pull room availability data from each hotel's internal management software.
> - It should allow users to filter rooms by date, price, and amenities.

- The interface needs to be intuitive and compatible with both desktop and mobile browsers.
- The system should be built using cloud-based solutions to ensure reliability and scalability.

TDD also often focuses on specific scenarios or use cases that need to be tested. Here are several types of prompts that can be useful:

> *Prompt:* Write tests for a feature that allows users to reset their password via email verification.
>
> *Prompt:* Create tests to verify the system's response when the database connection fails.
>
> *Prompt:* Develop tests to ensure that the search functionality returns results within 2 seconds under normal load conditions.
>
> *Prompt:* Write tests to check that all user data is encrypted when stored.
>
> *Prompt:* Design tests to verify the integration between the payment gateway and the order-processing system.

Planning Web Design

Planning a website or web app can be complex and involve numerous steps. First, you need to understand the goals for the site and the target users. Then, you will sketch out the website's blueprint and plan how users will move around. This is often known as the *prototype* or *wireframe*.

After this, it's about making it look good by picking colors, fonts, and snazzy graphics that show off what the brand's all about. Of course, you need some catchy words and cool pictures or videos to tell your story. Then comes the geeky part, where you turn all these ideas into a real, working website using languages like HTML, CSS, and JavaScript.

Of course, a tool like ChatGPT can be helpful when you're going through the planning stages. Here are some prompts:

> *Prompt:* I'm developing a website for <*a certain category*>. Suggest some objectives or goals. Who would be the target audience?
>
> *Prompt:* What are the pages I need for the basic layout of an online store?
>
> *Prompt:* I'm working on a wireframe for a blog's main page. Could you suggest some important elements to include?
>
> *Prompt:* I'm drafting a wireframe for a landing page of a mobile app. What essential sections should I make sure to incorporate?
>
> *Prompt:* For a website about <*a certain category*>, what would be good ideas for content, say blog posts, videos, and infographics?
>
> *Prompt:* How will you structure the navigation of your site to make it intuitive and user-friendly for your audience?

> *Prompt:* List the key SEO strategies you will implement to improve your website's visibility in search engines.
>
> *Prompt:* What color palette do you envision for my website about <*a particular category*>?
>
> *Prompt:* Select font styles for your website about <*a particular category*>. Consider readability and brand alignment and how the fonts contribute to the overall aesthetic of the site.

There are also various tools that use AI to help create wireframes. One is Uizard. Similar to a system like Figma, it has design functions, such as the ability to drag-and-drop elements like buttons and forms. It can also be used by teams to collaborate.

The AI features are located on the toolbar on the left of the dashboard. The button is called Magic. Click it, and you will see a list of AI features.

To create a wireframe, select Autodesigner and this will bring up a wizard, which you can see in Figure 7-7.

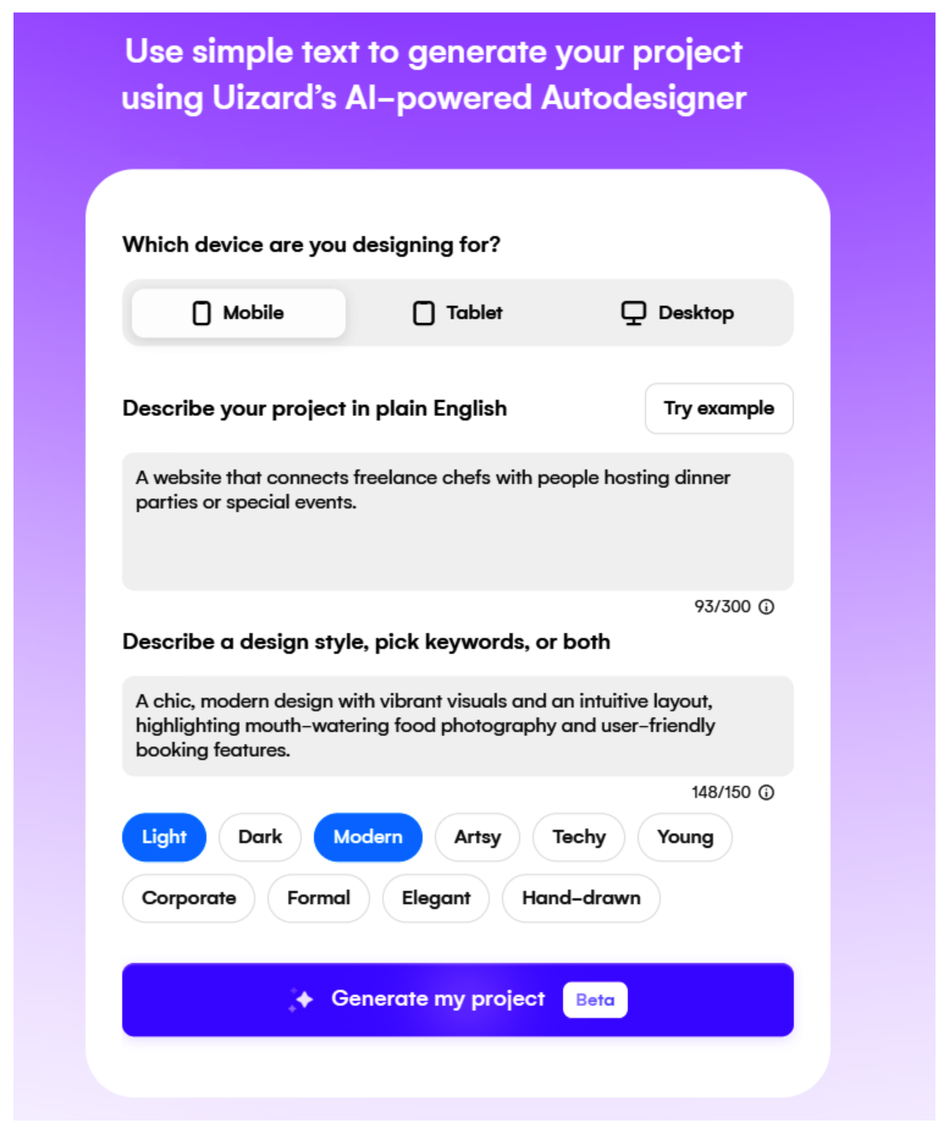
First, you will select the device to design for: mobile, tablet, or web. Then, you will describe the project. You may provide just a sentence or two. Here is an example:

> *Prompt:* A website that connects freelance chefs with people hosting dinner parties or special events.

Next, you'll come up with the design style, such as:

> *Prompt:* A chic, modern design with vibrant visuals and an intuitive layout, highlighting mouth-watering food photography and user-friendly booking features.

Figure 7-8 shows the wireframe.



Use simple text to generate your project using Uizard's AI-powered Autodesigner

Which device are you designing for?

Describe your project in plain English

A website that connects freelance chefs with people hosting dinner parties or special events.

93/300

Describe a design style, pick keywords, or both

A chic, modern design with vibrant visuals and an intuitive layout, highlighting mouth-watering food photography and user-friendly booking features.

148/150

Figure 7-7. Uizard has an AI wizard to help create a wireframe for a mobile app or website

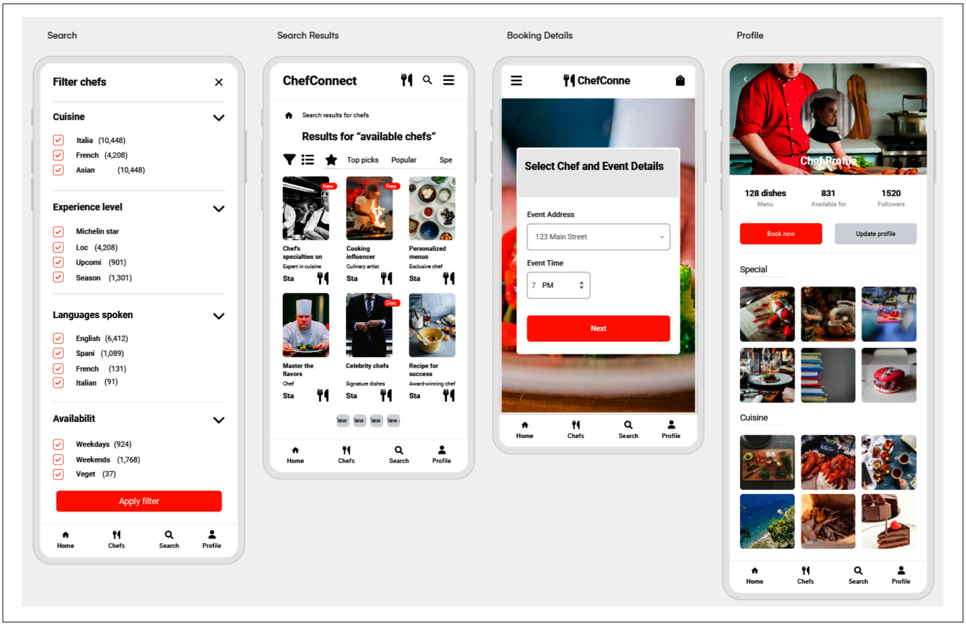


Figure 7-8. Based on the given prompts, Uizard generated this wireframe for a website

Conclusion

In this chapter, we explored how to use ChatGPT to start a software development project. We kicked things off with brainstorming, followed by digging into market research. Then, we dived into the nitty-gritty of drafting requirements, shining a spotlight on how PRDs and SRSs help get your project goals and technology details on point. We checked out different planning styles, from the go-with-the-flow Agile to the structured waterfall. We also looked at whiteboarding and TDD techniques. By mixing a powerful tool like ChatGPT with tried-and-true methods, you can build a strong base for a project.

CHAPTER 8

Coding

In this chapter, we'll dive into some handy coding techniques. We'll start with the bread and butter of coding—playing around with functions and classes. Then, we'll get our hands dirty with a bit of refactoring, tidying up messy code and making it run smoother. We'll also show how to work with and create data. Then we'll check out some AI tools that can help create fancy frontends without breaking a sweat.

Reality Check

AI-assisted programming tools are pretty cool, right? But let's not kid ourselves. They're not the magic wand that can solve all our coding problems. Why? Well, these tools are like those students in class who only learn from public notes. They're trained on a mishmash of code from all over the internet, and let's be honest, not all of it is top-notch. What does this mean for you? Sometimes, you might get code that's more bloated than a Thanksgiving turkey or as maintainable as a house of cards. And sometimes the code is just plain wrong, doesn't do what you need, or, even worse, leaves your front door wide open for hackers or the hogging of your network resources.

Let's look at an example where ChatGPT produced messy code. Figure 8-1 shows code for merging two sorted lists into a single sorted list.

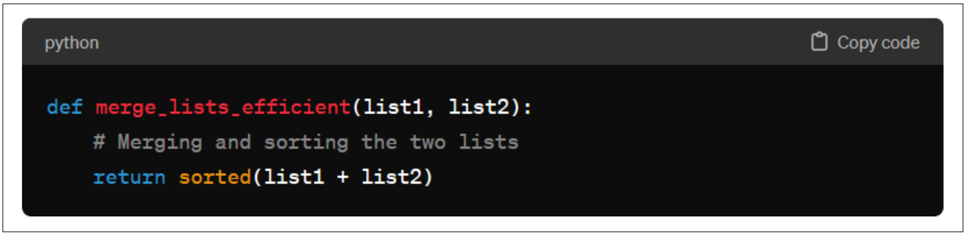
```python
def merge_lists(list1, list2):
    if not list1:
        return list2
    if not list2:
        return list1

    merged_list = []
    index1, index2 = 0, 0

    while index1 < len(list1) and index2 < len(list2):
        if list1[index1] < list2[index2]:
            merged_list.append(list1[index1])
            index1 += 1
        else:
            merged_list.append(list2[index2])
            index2 += 1

    while index1 < len(list1):
        merged_list.append(list1[index1])
        index1 += 1

    while index2 < len(list2):
        merged_list.append(list2[index2])
        index2 += 1

    return merged_list

# Example usage
print(merge_lists([1, 3, 5], [2, 4, 6]))
```

Figure 8-1. When asked to merge two sorted lists, ChatGPT created some messy code

This AI-generated code does correctly merge two lists, but it takes a verbose approach. It separately handles the cases of empty lists and uses two additional `while` loops to append the remaining elements from each list.

This is where your coding smarts really come into play. You've got to know the fundamentals. You need to know what questions to ask and where to poke around when something isn't quite right.

Figure 8-2 shows a better approach.



```python
def merge_lists_efficient(list1, list2):
    # Merging and sorting the two lists
    return sorted(list1 + list2)
```

Figure 8-2. With some human help, we've achieved a better approach to the merging of the sorted lists

So don't worry about all the doom and gloom you may hear about robots taking over developer jobs. Granted, these AI models are getting better all the time. They're learning and growing. Yet they are still far from being the be-all and end-all of programming. Programmers are still very much in the game. AI-assisted programming tools are extremely helpful, but they're not about to replace the savvy and know-how of real, live engineers.

When you're jamming with these systems, remember: it's a collaboration, not a handover. Stay sharp, stay curious, and don't forget to double-check everything these AI tools serve up. They're helpers, not heroes.

Judgment Calls

Sometimes, it's simpler to just do the coding yourself instead of using an AI-assisted programming tool. After all, it can take a bunch of tries to get the AI to catch on to what you need. But as you keep using these AI tools, you'll get the hang of figuring out when they're useful or when you're better off flying solo.

Take the experience of Dmitrii Ivashchenko, a lead software engineer at MY.GAMES, an Amsterdam-based game company with over 1 billion registered users worldwide. According to him:

> An example would be the prompt "Write a method in Python to add a default timezone to the datetime object." However, you should be prepared that many aspects will be omitted and you will either have to finalize the handling of corner cases yourself or have a long correspondence with ChatGPT pointing out its errors. The main thing here is to initially gauge the time it takes to explain a task versus what it will take for you to implement it yourself.

Learning

Using something like ChatGPT to pick up a new language, framework, or library can be a big help. It may be faster than the old-school ways like taking a course, endlessly Googling, or watching a bunch of YouTube videos.

You can use ChatGPT to get the lowdown on all the basics and syntax of the code. Plus, if you're more of a learn-by-doing type, you can ask for code examples to see how everything clicks together. If you already know one programming language, ChatGPT can make learning a new one easier by comparing the two. Suppose you're good with Python but just diving into JavaScript; it can explain the main differences in Python terms. If you need more resources, ChatGPT can scour the internet and recommend some solid learning materials for you.

Here are some prompts to consider:

> *Prompt:* For someone who does not know Python, what should I learn first?
>
> *Prompt:* What are some beginner-friendly projects I can try to improve my skills in React?
>
> *Prompt:* Can you suggest resources for learning advanced SQL queries?
>
> *Prompt:* How would you implement a basic "to-do list" app in both Python and JavaScript to highlight their differences?
>
> *Prompt:* I know how to use loops in C++. Can you show me how loops work in Python for comparison?
>
> *Prompt:* Find 5 useful YouTube videos that show how to learn Rust.

Let's now see how Ivashchenko approaches this:

> If you want to quickly master a new language or framework, you can immediately start making a new project on that technology. Let's say you want to learn Django. Let's choose a project that we will implement on this technology—let it be an online store. To begin with, just ask ChatGPT how to start developing a new project on Django. It will tell you what dependencies you need to install, how to set up the initial configuration of the project, and what related technologies you will need to deal with. Then we move on to more specific requests such as "How is a model organized in Django?" "How to add a new section in the admin panel?" and "How to migrate the database?" By gradually gaining new knowledge and asking more and more detailed questions, you can realize almost any project on any previously unfamiliar technology. This is how I mastered Django literally through a weekend of fruitful work.

You can have ChatGPT create useful study guides as well. Here's a sample prompt:

> *Prompt:* Create a study guide for JavaScript. Focus on a beginner level.

You can then follow this up with:

> *Prompt:* Please create a study schedule for me. What topics should I study? What about practice exercises or quizzes? Coding problems? Links to resources?

ChatGPT can even help you to remember new concepts. One way is to use a mnemonic device, such as acronyms. For example, suppose you want to remember the key elements of DevOps, which include continuous integration, continuous delivery, microservices, and infrastructure as code. This is what ChatGPT suggests:

> *ChatGPT*: CI-CD-MI (pronounced "Ci-Ci-D-Mi")

Finally, you can use ChatGPT to get help with LeetCode. Picture it: you're sitting in the interview room, maybe with a whiteboard or a laptop in front of you, and then they hit you with this brain-teasing algorithm or data structure question. It's more than just coding. They're sizing up how quickly you think, how well you explain your thoughts, and your problem-solving style.

If you want to get a leg up, think about signing up for LeetCode. It's $35 a month. But of course, ChatGPT can lend a hand whenever you need it. Here's a simple prompt:

> *Prompt:* Suggest 3 common LeetCode interview questions.

ChatGPT then came up with challenges like two sum, longest substring without repeating characters, and valid parentheses. You can then dive deeper, such as by getting longer explanations or examples.

Comments

When you're racing against a deadline, it's easy to skimp on code comments or ditch them altogether. But AI-assisted tools can have your back. You'll often find code suggestions with comments already included.

Funny enough, because of AI technologies, having comments may feel somewhat retro. If code's got you scratching your head, you could just ask the tool to break it down for you, right? This is certainly true.

Something else: writing prompts is becoming the new way to comment. After all, most tools include the prompts in the comment lines.

Of course, if you want to add comments to some code, you can use a simple prompt like:

> *Prompt:* Add comments that are clear and according to best coding practices.

Whether or not to comment is your call. There's no one-size-fits-all rule here. It's all about what works for you and your team. But one thing's for sure—slapping comments onto your code is a breeze with AI-assisted tools.

Modular Programming

Modular programming is a cornerstone of efficient and effective software development. With modular programming, coding is more organized, easier to understand, and easier to keep up with. It also makes teaming up with other coders a lot less of a headache because everyone's not tripping over each other. Plus, modules are reusable; reusability is a massive time-saver, keeps things consistent, and makes it less likely you'll goof up.

The value of modular programming certainly applies to how you work with AI-assisted tools. They will not whip up an advanced application from a simple prompt. They're not wizards. But if you break your task down into clear, specific pieces, these tools shine. Otherwise, you might end up with a code jumble that's way off track.

According to Titus Capilnean, the cofounder and chief product officer at Private Market Labs:

> After I started using AI tools, I could focus on the problem and my approach to get a solution, not the minutiae of the actual code I need to write. When I have a technical problem, I start by breaking it down into smaller pieces, where the inputs and outputs are clear. The reason for this is that the context window of the AI tool I'm using might not be sufficient to come up with a good solution in one shot. I found that it's easier to debug and build if I ask the model to deliver code that uses a simple input, does one single job and provides an output I can verify. If I'm not happy with the intermediate output, I describe my issue to ChatGPT and attempt to run the updated code it delivers. Once I'm happy with the solution, I can add extra processing steps, either inside the existing function or that takes the output of the previous step and refines it further, closer to my desired solution.
>
> AI tools also improved my productivity by allowing me to check whether a snippet of code I built is actually correct before even sending it to a compiler or deploying a cloud function and consuming resources on an error. I just paste the part of the code in the ChatGPT window and ask whether it is correct, assuming I initialized/provided the correct context for the snippet to function (e.g., imports, variables, constants, custom functions being called, etc.). This allows me to produce solutions that run well and faster.

Starting a Project

At the start of a coding project, you may hit the "cold start problem" or "blank page problem."

Here's a scenario: you're staring at an empty screen, no code, no data, not even a clear path forward. No doubt, it's as overwhelming as it sounds. The first big headache is selecting your project's architecture, design patterns, and what technology to use. These decisions are huge because you'll be living with them for the long haul, so you want to nail them from the get-go as much as possible.

And let's not forget the people factor. Getting your team on the same page, figuring out how to communicate effectively, and setting up a workflow from scratch—these can be just as challenging as the technology. It's not just about slapping down those first lines of code. It's about laying a solid foundation for everything that comes next. Tackling this phase needs a mix of smart planning, technical know-how, and solid teamwork.

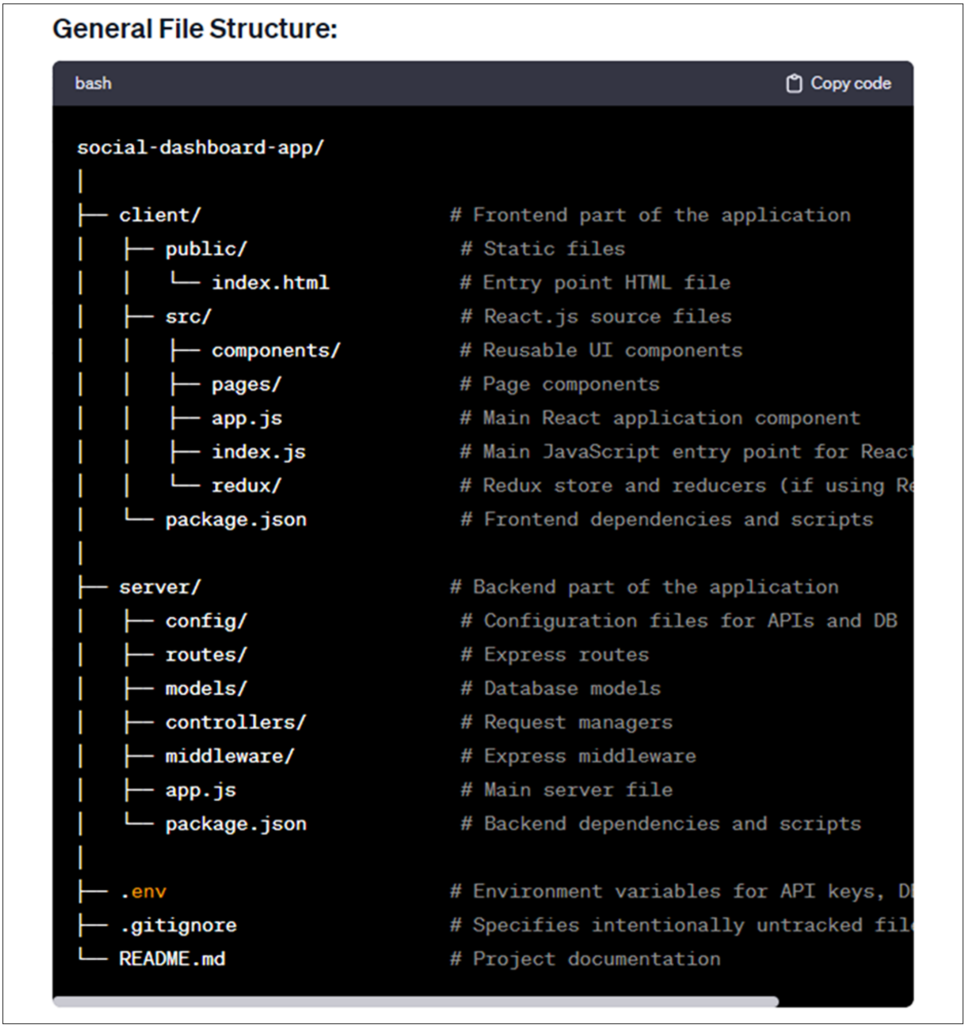
AI-assisted programming tools can be a big help. You can use them to set up the basic scaffolding of your app. You'll figure out a customized starting point that aligns with your vision. The tools save you from the nitty-gritty of initial setup and let you jump straight into the more interesting parts of your project.

You can come up with a particular task and then prompt the LLM to generate boilerplate or starter code.

This is a sample prompt:

> *Prompt:* Generate starter code for a web app that aggregates a user's social media feeds from multiple platforms into a single dashboard interface. What do you suggest I use for the language and frameworks? What about the general file structure?

With this, ChatGPT starts off by suggesting React for creating a smooth, single-page application (SPA). It also suggests using Redux to maintain strongly styled and reusable components. It then says to use Node.js to run the application and Express.js for the API endpoints. It goes on to recommend Passport.js for the logins and MongoDB and Mongoose for the database and model building. As for the file structure, Figure 8-3 shows the layout.



General File Structure:

```bash
social-dashboard-app/
|
├── client/                  # Frontend part of the application
|   ├── public/              # Static files
|   |   └── index.html       # Entry point HTML file
|   ├── src/                 # React.js source files
|   |   ├── components/      # Reusable UI components
|   |   ├── pages/           # Page components
|   |   ├── app.js           # Main React application component
|   |   ├── index.js         # Main JavaScript entry point for Reac
|   |   └── redux/           # Redux store and reducers (if using R
|   └── package.json         # Frontend dependencies and scripts
|
├── server/                  # Backend part of the application
|   ├── config/              # Configuration files for APIs and DB
|   ├── routes/              # Express routes
|   ├── models/              # Database models
|   ├── controllers/         # Request managers
|   ├── middleware/          # Express middleware
|   ├── app.js               # Main server file
|   └── package.json         # Backend dependencies and scripts
|
├── .env                     # Environment variables for API keys, D
├── .gitignore               # Specifies intentionally untracked fil
└── README.md                # Project documentation
```

Figure 8-3. ChatGPT suggests a file structure for a new web app

Autofill

You know when you're deep in the coding zone, setting up those constants for time units, and it's just line after repetitive line? Here's where GitHub Copilot jumps in. You can have it provide autofill.

Let's take an example. Suppose you are creating an app that you want to make responsive. This means you will need to have constants for the breakpoints for a CSS-in-JS styled-components library.

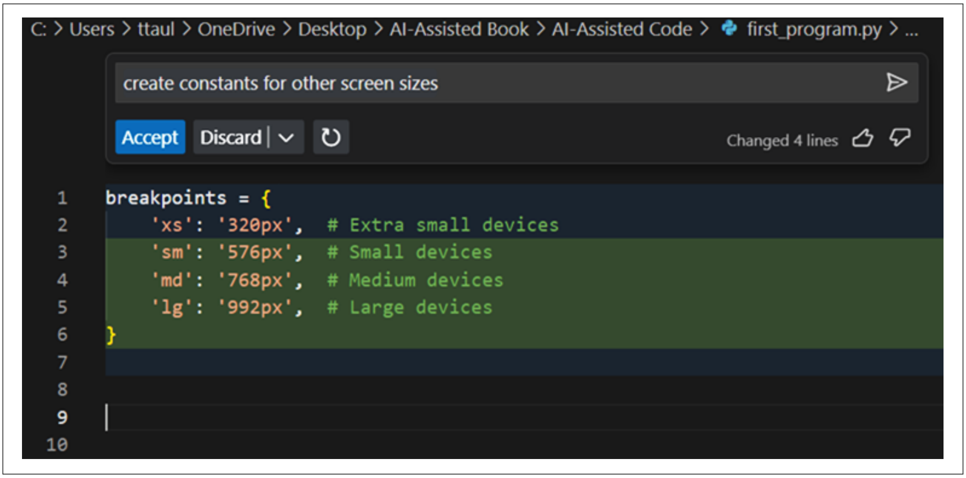
You can first write this:

```
breakpoints = {
    'xs': '320px',  # Extra small devices
```

Then in the inline chat, you can use this prompt:

> *Prompt:* Create constants for other screen sizes.

Figure 8-4 shows the result. It provides other screen sizes and variables that have a similar style.



C: > Users > ttaul > OneDrive > Desktop > AI-Assisted Book > AI-Assisted Code > first_program.py > ...

create constants for other screen sizes

```
breakpoints = {
    'xs': '320px',  # Extra small devices
    'sm': '576px',  # Small devices
    'md': '768px',  # Medium devices
    'lg': '992px',  # Large devices
}
```

Figure 8-4. This is output from Copilot that suggests different constants for screen sizes

You can go further with this. Copilot can help with autofill by looking at open files in your project.

According to Capilnean:

> I found that Copilot is especially useful if you have parts of the code that are similar when it comes to data structures. If you have defined an object in another file that's part of the code repository you're working with, then when you are trying to define a similar object, it can reliably pre-fill the code for you as you type. For example, an object for a common API call, like a Sendgrid email, works well here.
>
> However, this autocomplete can hallucinate if you're not careful. On imports, for example, it won't always get the folder structure right, especially if you designed your own code shape, or if the framework you're using is fairly new and not well documented yet. In that case, I have to go in and manually check where the component is coming from and whether the directory that Copilot suggested was correct.

Refactoring

Refactoring is like spring cleaning for your code. It's all about tidying up, reorganizing, and sometimes sprucing things up. It's not about adding new features or fixing bugs.

Refactoring keeps the codebase healthy, less cluttered, and more intuitive. This means when you or someone else jumps back in later, it's easier to understand what's going on, which cuts down on headaches and, well, swearing at the screen. Plus, cleaner code is usually more efficient and less buggy, so it's a win-win for everyone involved.

Refactoring is one of those areas where AI-assisted tools shine. With a prompt or two, you can slice through the complexity of your code, trim the fat, and reorganize your code into something that's not just functional but elegant. With AI in your corner, refactoring becomes less of a chore and more of a secret weapon for staying ahead of the curve.

In the next few sections of this chapter, we'll take a look at some examples of refactoring.

Ninja Code

Think of ninja code as one of those over-the-top action movie stunts. It's flashy and complex, and it leaves you thinking, "Whoa, how'd they do that?" But here's the catch: it's a beast to figure out once the awe wears off. Imagine code so sneaky and tangled that even the person who wrote it can't make heads or tails of it after their "ninja" phase passes. Sure, pulling off that ninja move feels epic in the moment, but when you or someone else needs to jump back in and make changes, it's less "hi-ya!" and more "uh-oh." The truth is, while ninja code might show off some serious programming chops, it often goes against the grain of good coding practices, which are all about keeping things clean, simple, and understandable.

Regardless, ninja code is common. But refactoring with an AI-assisted tool can help sort things out. Here's an example:

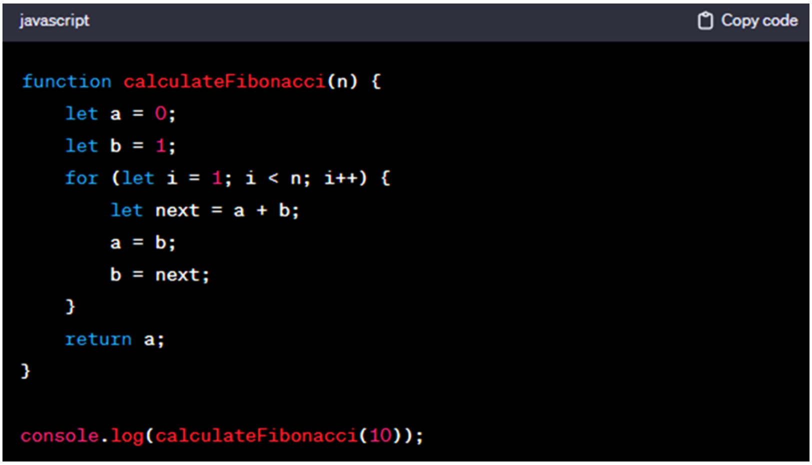
```
console.log((function(n, a = 0, b = 1) { while (--n) [a, b] = [b, a + b];
return a; })(10));
```

Make any sense? Well, you'd probably need to take quite a bit of time to figure this out. Instead, make ChatGPT do the work:

> *Prompt:* Can you explain this code in a step-by-step process? Also, can you write this in a simpler way that is more maintainable?

ChatGPT determines that this is a function to calculate the *n*th Fibonacci number. It goes on to describe it in six steps and then provides a simplified version, as seen in Figure 8-5:

ChatGPT: In this version, the calculateFibonacci function clearly shows the process of iterating through the Fibonacci sequence. It's more verbose but much easier to understand and maintain, especially for other developers who might read this code in the future.



```javascript
function calculateFibonacci(n) {
    let a = 0;
    let b = 1;
    for (let i = 1; i < n; i++) {
        let next = a + b;
        a = b;
        b = next;
    }
    return a;
}

console.log(calculateFibonacci(10));
```

Figure 8-5. ChatGPT has provided a simpler version of what was ninja code

Extract Method

You apply the extract method when you have a long method or function. You'll pick out a chunk of the code that's focused on a certain task—say data checking or a specific calculation—and you'll turn this into a new method.

This approach isn't just about making things look neat and tidy. It's practical, too. It helps you keep your code easy to read and understand. When you need to do that same task again, you can call up your new method instead of rewriting the code. And if there's ever a glitch, it's easier to sort out when your code is nicely split up into these focused, bite-sized pieces. It's about making your life easier and keeping your code clean and organized.

Here are some prompts for the extract method:

> *Prompt:* Are there any common pitfalls to avoid when extracting methods in functional programming languages?
>
> *Prompt:* I've attached a piece of my C# code. Could you suggest which sections would be good candidates for extract method refactoring?
>
> *Prompt:* Can you compare my original function and the refactored version with extracted methods? Which is more efficient?

Decomposing Conditionals

Decomposing conditionals is about breaking down big, gnarly if-then-else statements into something a lot more digestible. You know the kind—those lengthy, twisty conditions that make you squint at your screen trying to figure out what's going on.

Let's say you have an `if` statement with a complex condition that checks multiple variables and perhaps calls other functions. Instead of trying to decipher this every time, you can extract this condition into a method with a name that clearly describes what the condition is checking. For instance, a condition like `if (user.isActive() && user.hasSubscription() && !user.isBlocked())` can be refactored into a method named `canUserAccessContent()`. This not only makes your main method cleaner but also instantly makes the code self-explanatory.

Similarly, the code within the `then` and `else` blocks can be extracted into distinct methods. This makes the main flow of your program much more readable. Instead of wading through lines of detailed logic, a reader can now understand the flow at a high level: if this condition is true, do this; otherwise, do that. Each part of the logic lives in its own neatly named method, making it easier to test and modify in isolation.

Here are some prompts:

> *Prompt:* Can you explain how to decompose conditionals in Java code for better readability?
>
> *Prompt:* How can I break down complex if-then-else statements using the decomposing conditional technique?

Renaming

Renaming functions, variables, and classes might seem like a small change, but it can have a significant impact on the readability and maintainability of code. This is particularly useful in situations where code has evolved over time and original names no longer accurately describe what the code does. For instance, a method initially named `processData` might become more specialized. Renaming it to something more descriptive, like `filterInvalidEntries`, can instantly clarify its functionality.

Writing a prompt for renaming is fairly simple:

> *Prompt:* What would be a good name for a variable that holds the total number of users in a database in my SQL script?
>
> *Prompt:* Here's a snippet of my JavaScript code. Can you review the variable and method names and suggest improvements?
>
> *Prompt:* I'm not sure if the names in my Java class are clear enough. Can you propose better names for clarity?

But you need to be cautious, such as when you're using a tool like Copilot. If you change a name, this can break parts of the code that are still using the old name.

Dead Code

As the name implies, dead code is not being used for anything. These are the forgotten lines of code from features that got scrapped or updates that made some parts redundant.

Scrubbing this dead code from your project makes everything neater and more manageable. It also makes it less confusing for anyone new jumping into your project. They don't have to scratch their heads over why something's there if it doesn't seem to be doing anything.

Here are some helpful prompts:

> *Prompt:* Can you help me identify potential dead code in this JavaScript snippet?
>
> *Prompt:* Here's a piece of my Python project. Could you point out any code that seems unused or redundant?
>
> *Prompt:* Could you take a look at these SQL procedures and confirm if any of them are safe to delete?

Note that using an LLM for this can be risky. Sometimes what looks like an old, dusty corner of code might actually be important for those rare, just-in-case scenarios. Then there's the domino effect: removing one piece might mess up something else that was relying on it, especially if it's part of complex logic or setup. So just be cautious.

The other problem is that the generative AI may not truly understand the relationships. So until AI systems get more sophisticated, it's probably best to avoid using them for rooting out dead code.

When it comes to dealing with dead code, alternative tools may be a better option. An example is a linter. Think of this as your code's tidy-up crew. If you're working with JavaScript, there's ESLint. Or for Python fans, there's Pylint, and the Ruby folks can turn to RuboCop. These tools are like the grammar checkers of coding. They're awesome at picking up on those pesky syntax mistakes, potential bugs, and, of course, even those sneaky bits of code that aren't doing anything.

You've also got the heavy-duty inspectors: the static code analysis tools. Top providers are SonarQube, Code Climate, and Coverity. These tools are like detectives. They dig through your code without even running it, sniffing out complicated patterns that might be troublemakers down the line, including dead code.

Functions

Functions are the bread and butter of coding, playing a huge role in any kind of software program, no matter what programming language is used. They're chunks of code that do wonders for keeping your programs tidy and easy to read since they let you easily reuse code—a lifesaver for any developer. They can also take a big task and break it down into smaller, more manageable bits. This makes dealing with complex software a whole lot easier, especially when you need to fix bugs, make updates, or just try to wrap your head around what the code is doing.

It's key to nail functions right from the get-go. Making them work is one thing, but you also need to make sure they play nice with the rest of your code. You want your functions to be clear, easy to keep up with, and efficient. You'll need to think about what to call each function so that it makes sense, how you're going to set it up, how it will deal with the data coming in and going out, and how it handles any hiccups.

To help out, here are some guidelines to keep in mind:

Think of the single responsibility principle
: Your function needs to be a specialist in one job—that's it. This makes it a lot simpler to figure out what it's doing, check if it's working right, and fix it if it's not.

Name it clearly
: Give your function a name that tells you exactly what it does. If it calculates the total price, call it `calculateTotalPrice`. This makes your code much more readable.

Keep it short and sweet
: A good rule of thumb is that you should be able to see the whole function on your screen without scrolling. Short functions are easier to handle and less likely to have bugs.

Parameters are key
: Use parameters for inputs and return values for outputs. This makes your functions predictable and self-contained.

Stay consistent
: Follow the coding conventions and style guidelines of your language or project. This helps keep your code uniform and easy for others to read.

When you keep these tips in mind, you can whip up some really effective prompts for functions, and ChatGPT can help. Check out these example prompts:

> *Prompt:* Write a Python function named calculate_area that takes two integers as parameters, length and width, and returns the area of a rectangle. Include a docstring

explaining the function's purpose and ensure the function handles non-integer inputs by raising a TypeError.

Prompt: I need a JavaScript function called filterAndTransform. It should take an array of objects as input. Each object has properties name (string) and age (number). The function should return a new array containing the names of people who are 18 years or older, converted to uppercase. Include comments explaining the logic.

Prompt: Create a C++ function named efficientSort that sorts an array of integers in ascending order. The function should be optimized for time complexity. Provide comments within the function explaining the choice of sorting algorithm and its time complexity.

Prompt: Can you generate a Java function called safeDivide that takes two double parameters, numerator and denominator, and returns their division? The function should handle division by zero by returning a custom error message. Include Javadoc comments explaining the function and its error handling.

Object-Oriented Programing

Object-oriented programming, or OOP for short, is a way of writing computer programs using the idea of "objects" to represent data and methods. Think of it like creating a bunch of small, self-contained boxes, each with its own set of tools and information. These boxes, called *classes*, are like blueprints for creating different objects. A class defines the structure and behaviors of objects—similar to a template. Then from this class, you can create individual *objects*, each with its own specific details but following the same basic structure.

Diving into the world of OOP can feel as if you're stepping into a maze of complex concepts like abstraction, inheritance, encapsulation, and polymorphism. They can feel like they're written in an alien language.

This is where ChatGPT can be your translator. It breaks down these complex ideas into bite-sized, easy-to-digest explanations. Struggling with what *encapsulation* really means? Just ask, and you'll get an answer that actually makes sense, minus the tech jargon.

Here are some prompts:

Prompt: Can you create a simple class in <*your preferred programming language*> that demonstrates encapsulation?

Prompt: What are some real-world examples of encapsulation in programming?

Prompt: Explain abstraction in OOP with an analogy from everyday life.

Prompt: Can you show me an example of inheritance in a programming scenario?

Prompt: How does inheritance promote code reuse in OOP?

Prompt: How does polymorphism enhance flexibility in a program?

Prompt: In what scenarios is polymorphism particularly useful, and can you provide examples?

AI-assisted programming tools can also be helpful in coming up with the initial structures for classes. Here are some example prompts:

> *Prompt:* Design an Employee class with properties like employeeName, employeeID, and department. Implement a method that displays the employee's details. Also, include a constructor to set these properties.
>
> *Prompt:* I need a BankAccount class in Java. It should have private properties like balance and accountNumber. Can you add methods for deposit(), withdraw(), and checkBalance() that modify or access these properties safely?
>
> *Prompt:* Could you show me how to create a Vehicle class in C# and then a Truck class that inherits from it? Make sure to include properties like wheels and fuelType and demonstrate the use of different access modifiers.
>
> *Prompt:* In C++, how would I write a FileHandler class that opens a file in its constructor and closes it in its destructor? Also, include methods for writeToFile() and readFromFile().

Frameworks and Libraries

Diving into software development without frameworks and libraries is like trying to bake a fancy cake from scratch without a recipe or premixed ingredients. It's possible, but it's a whole lot harder and takes more time. Frameworks and libraries are the secret sauce that makes a developer's life easier. Instead of reinventing the wheel every time you need to make a web request or manipulate a DOM element, you just tap into what's already there.

AI-assisted programming tools can certainly help. First of all, they can be useful for learning the basic features and workflows. They can also tell you when it's best to use a framework or library.

But take their help with a grain of salt. Here's what Capilnean has to say:

> Given the large number of updates React gets—as well as other frameworks and libraries—and the fact that we use specific versions of frameworks, I had to provide these as system prompts to my ChatGPT instance in order to optimize my results. Sometimes, GPT provided solutions that were more academic than production-grade, so I tend to rely on working with our senior developers to get the job done for more complex issues. For React, I find myself asking GPT to check my syntax and my method of processing a specific data type more than actually build a full feature for me.
>
> The same goes for NodeJS. I have to take into consideration our internal APIs and methods of working with our data before I can go to ChatGPT and ask for a code snippet for a feature. Once I am able to describe the output of one of our functions as the input for the feature I am building, it usually is able to provide me with stable code. I use the same process here if the output isn't good or it errors out. I provide it with the issue and ask for a code update in the right direction.

Data

Data is the lifeblood of every application, truly. It's what keeps an app alive and kicking. Just like blood carries oxygen and nutrients to keep our bodies functioning, data flows through an application, giving it the information and insights it needs to work its magic.

But creating sample data can be a slog. Picture this: you're excited to test your shiny new app, but first, you need a bunch of data to see how it actually works in the real world. You start typing in rows and rows of data—names, dates, numbers, whatever it takes. But then, it just keeps going. And going.

What's more, you have to be careful about making it realistic enough so your tests are valid, but not so detailed that you're writing a novel for each entry. And if you need a large dataset? Forget it: you're basically signing up for a marathon of copy-pasting, tweaking, and double-checking.

An AI-assisted programming tool can be handy for this. But first, you need to select a database and spin up the schema and tables. The tool will also need to be given the relationships, say, among the tables. Then you will need to do the configuration and setup. But then the AI can begin to help.

Here are some prompts to get help evaluating databases:

> *Prompt:* What type of database would be best suited for handling <*specific data types or functions, e.g., user interactions, product inventory, etc.*>?
>
> *Prompt:* For an app expecting <*high/low*> traffic with <*type of data, e.g., images, text, real-time data*>, which database should I consider?
>
> *Prompt:* I'm on a tight budget. Can you suggest a cost-effective database solution for a small <*type of app, e.g., local delivery service app*>?
>
> *Prompt:* I'm relatively new to database management. Which databases are user-friendly and easy to maintain for a beginner?

Here are some prompts to ask for help with a database schema:

> *Prompt:* Can you help me design a basic database schema for a <*type of application, e.g., online store, blog*>? I need to know what tables I should create and the primary relationships between them.
>
> *Prompt:* What would be an efficient table structure for managing <*specific type of data, e.g., customer orders, inventory*> in a relational database? What fields and data types should I include?
>
> *Prompt:* How should I define the relationships between tables in a relational database for an application that deals with <*describe the application's functionality, e.g., event management, course enrollment*>? Specifically, I need help with understanding foreign keys and join tables.

Prompt: I'm working on a relational database schema for <*describe the project*>. Could you guide me on setting up primary and foreign keys effectively for data integrity?

Prompt: What normalization strategy would you recommend for a database handling <*type of data or application function*>? How can I avoid data redundancy and ensure data integrity?

Prompt: I'm using a NoSQL database for a <*type of project, e.g., social media app*>. How should I design the document structures to store <*specific data types, e.g., user profiles, posts, comments*>?

Prompt: In designing my database schema, what indexing strategies should I consider for optimizing query performance, especially for <*type of queries or operations, e.g., full-text search, frequent updates*>?

Prompt: I need to migrate an existing database to a new schema. What are the key considerations and steps for redesigning the database structure without losing data integrity?

And here are some prompts for setting up the initial database environment:

Prompt: Can you provide step-by-step instructions for installing <*a specific database server, e.g., MySQL, PostgreSQL*> on <*a specific operating system, e.g., Windows, Linux*>?

Prompt: After installing <*name of database*>, what are the essential configuration settings I should initially set up for optimal performance?

Prompt: What are the best practices for securing a <*specific database*> server? I'm particularly interested in user authentication and protecting sensitive data.

Prompt: How can I optimize the performance of <*a specific database*> for an application that will handle <*describe the nature of the data and expected load, e.g., large datasets, high transaction volumes*>?

Now, let's look at how we can use AI to create sample data. Here are some example prompts:

Prompt: Create demo data for 100 IDs and email data and save this to a CSV file.

Prompt: Create demo data for 50 products, including product ID, name, price, and category.

Prompt: Create a demo dataset of 150 order records, each with an order ID, customer ID, order date, and total amount.

Prompt: Generate sample data for 100 employees, including employee ID, full name, department, and email address.

Prompt: Create sample data for 80 customer feedback entries, including feedback ID, customer ID, comments.

With the data, you can then create SQL statements for it. You can use something like this for the customer feedback entries:

Prompt: Generate a SQL insert statement to populate the Feedback table with the data.

ChatGPT can be a lifesaver when it comes to the nitty-gritty task of data conversion. If you're a developer, you know that converting data between different formats like XML, SQL, JSON, CSV, and TOML is common. But let's be honest, it can be a tedious and sometimes error-prone process. That's where ChatGPT comes in handy.

Here are some sample prompts:

> *Prompt:* Here's a CSV row: 'John Doe, 35, New York'. Can you convert this into an XML format for me?
>
> *Prompt:* I have a JSON array like this: [{'name': 'Alice', 'job': 'Engineer'}, {'name': 'Bob', 'job': 'Designer'}]. How would I represent this in a SQL table format?
>
> *Prompt:* I need to convert this TOML configuration: 'title = "My Project" owner = "Sara"' into an equivalent YAML format. How would you convert this?

Frontend Development

Frontend development involves making websites look great and navigate smoothly. At its core, it uses HTML to set up the basic structure of web pages, CSS to make things look nice and lay out everything, and JavaScript to make it interactive. Nowadays, frontend developers often use frameworks like React and Vue. These frameworks offer features like reusable components and ways to make data update in real time, which allows for building websites and apps that are dynamic and engaging.

Frontend development can be challenging, even for experienced developers. First of all, there is the unpredictable nature of web browsers and user interactions. You need to deal with different browsers, devices, screen sizes, and user preferences. It's like trying to make a one-size-fits-all T-shirt that looks good on everyone. Next, the frontend world changes fast. Framework updates pop up frequently.

Another thing about frontend development is that it's not just about coding. It's also a lot about good user experience (UX) and user interface (UI) design. This is a unique skill set that many programmers don't have. UX and UI design involves understanding how people interact with technology, what makes a design visually appealing, and how to create a smooth, logical flow in an app or website. It's like being part artist, part psychologist. For a lot of traditional programmers, who might be wizards at things like algorithms and data structures, the world of colors, layouts, and user journeys can be unfamiliar territory.

Regardless, there are ways that AI-assisted programming tools can help out. We'll look at some of these in the next few sections.

CSS

Writing CSS for websites can be tricky, especially when you're working on big, complex sites. It's challenging to make sure everything looks good across different browsers and devices. You've got to know all the weird browser-specific quirks. It's also easy to accidentally mess things up so that one part of your style steps on the toes of another, making things look wonky. Plus, traditional CSS doesn't let you use variables or functions, which means you end up repeating yourself a lot in your code (though tools like SASS and LESS help with that). Even something as simple as centering a `<div>` can be an ordeal.

Here are some prompts for common CSS tasks:

> *Prompt:* Can you provide me with a CSS snippet for a responsive navigation bar that collapses into a hamburger menu on mobile devices?
>
> *Prompt:* I'm having trouble with a flexbox layout in CSS where items are not aligning properly. Can you suggest what might be going wrong?
>
> *Prompt:* I need to add a hover effect to buttons on my website. Can you show me a CSS example to make the button change color when hovered over?
>
> *Prompt:* I'd like to add a fade-in animation to my website's homepage. Could you provide a CSS code snippet for this?

Creating Graphics

Creating professional graphics for websites or apps requires an extensive background in design, along with understanding sophisticated tools like Photoshop. But there are many powerful text-to-image generative AI systems that can help you create standout images. Some of these systems include:

- Canva
- Stable Diffusion
- DALL·E 3 (which is built into ChatGPT)
- Adobe's FireFly
- Midjourney

With these systems, you can create many types of assets, including hero images, buttons, and logos.

Let's look at an example. We'll have ChatGPT create a logo:

> *Prompt:* I'm creating a website for my home bakery business named "Sweet Whisk." I want a logo that's warm and inviting. The main products are cakes and cookies, so maybe those could be incorporated into the design. I like pastel colors, especially light pink and mint green. The style should be simple and modern, with a touch of playfulness.

Figure 8-6 shows what ChatGPT created.

Figure 8-6. ChatGPT created a logo when given some guidelines

If you want to make changes to the output, you can continue to prompt ChatGPT. Other AI image tools, like Firefly, have more features than ChatGPT for designing images, but ChatGPT can still create compelling images—without much work.

AI Tools

There are some excellent AI tools that can whip up websites just from a prompt or a picture of what you want the frontend to look like. They'll handle all the coding for you. This even includes making conversions to frameworks like React, Angular, and Vue. What's more, an AI tool will also usually be able to import a wireframe, say from Figma.

After the code is created, you can jump in and tweak things to get it just right. Using AI is an effective shortcut for speeding up the whole website-building process.

Here are some of the AI tools available:

- TeleportHQ
- Anima
- Locofy
- v0 by Vercel

Let's take a closer look at v0 by Vercel, which is easy to use. In fact, the interface is similar to that of ChatGPT, as you can see in Figure 8-7.

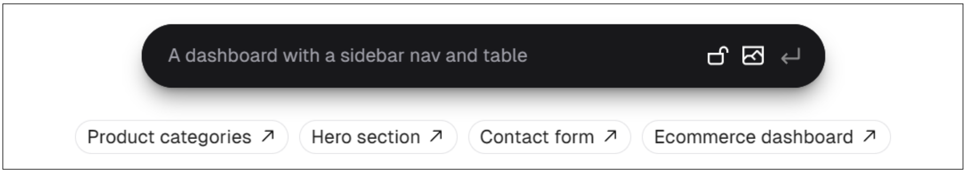
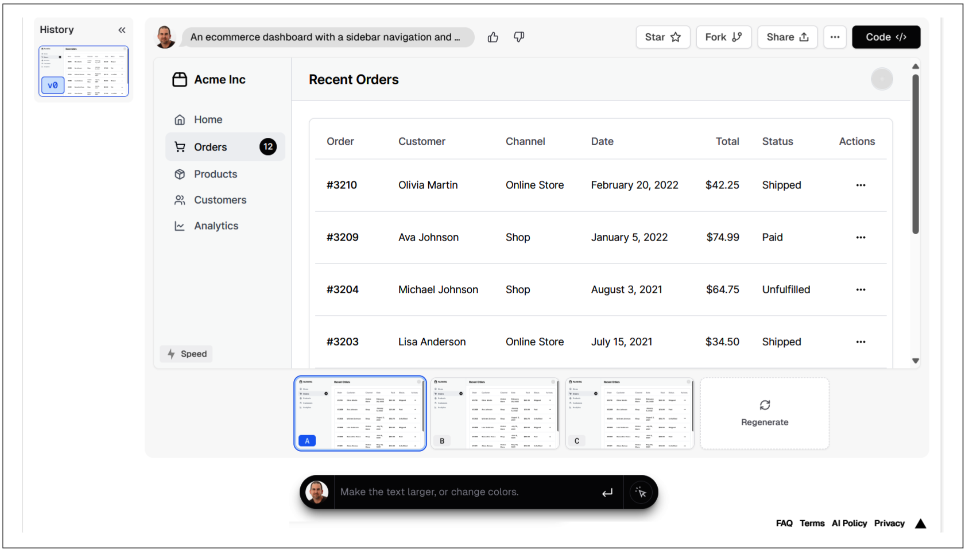


Figure 8-7. This is the interface for v0 by Vercel

You can either enter a prompt for the type of interface you want the system to create or upload an image. For this example, I will use a prompt:

> *Prompt:* An ecommerce dashboard with a sidebar navigation and a table of recent orders.

This will take you to a design studio, which will look like Figure 8-8.



Recent Orders

Order	Customer	Channel	Date	Total	Status	Actions
#3210	Olivia Martin	Online Store	February 20, 2022	$42.25	Shipped	···
#3209	Ava Johnson	Shop	January 5, 2022	$74.99	Paid	···
#3204	Michael Johnson	Shop	August 3, 2021	$64.75	Unfulfilled	···
#3203	Lisa Anderson	Online Store	July 15, 2021	$34.50	Shipped	···

Figure 8-8. The design studio that v0 by Vercel comes up with when you enter a prompt

The AI will create three versions of the dashboard. You can create additional ones, too. Then, with a prompt, you can make changes, such as to the text size, colors, and so on.

When you are finished, you can click Code at the upper right. You'll see a listing of clean React code that's based on the shadcn/ui and Tailwind CSS.

There are also several open source systems that use AI to generate frontends. Consider Screenshot to Code. The creator of this project is Abi Raja, who is the CEO and founder of Pico. His prior startup was acquired by Yahoo! in 2013.

Raja spent six months creating the initial codebase for Screenshot to Code. “Frontend developers often convert designs and mock-ups into code,” said Raja. “Much of this work is repetitive. But my software can help automate this process, achieving about 90% of the task.”

Currently, Screenshot to Code supports exports to React, Bootstrap, and HTML/Tailwind CSS. The project has over 31,000 stars on GitHub.

With ChatGPT, you can also convert an image to code. Suppose you want to create a calculator app and you want it to look like the version on the iPhone. Figure 8-9 shows the image.

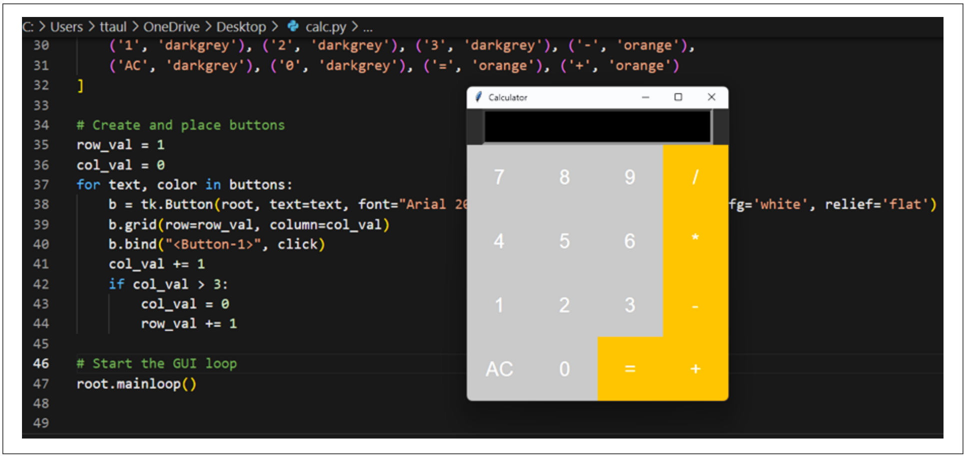
Figure 8-9. You can input an image of the iPhone calculator app to ChatGPT to generate code

First, we’ll ask ChatGPT to identify the image, which it does correctly. Then we can use this prompt:

> *Prompt:* Suggest code for this image

Figure 8-10 shows the calculator.

ChatGPT created code for Python. True, the styling is off, but this is still fairly good. Of course, if you want it to look more like the iPhone version, you can be more specific with the prompt, such as by telling ChatGPT to use something like React.



C: > Users > ttaul > OneDrive > Desktop > calc.py > ...

```
    ('1', 'darkgrey'), ('2', 'darkgrey'), ('3', 'darkgrey'), ('-', 'orange'),
    ('AC', 'darkgrey'), ('0', 'darkgrey'), ('=', 'orange'), ('+', 'orange')
]

# Create and place buttons
row_val = 1
col_val = 0
for text, color in buttons:
    b = tk.Button(root, text=text, font="Arial 2                                  fg='white', relief='flat')
    b.grid(row=row_val, column=col_val)
    b.bind("<Button-1>", click)
    col_val += 1
    if col_val > 3:
        col_val = 0
        row_val += 1

# Start the GUI loop
root.mainloop()
```

Figure 8-10. ChatGPT created code for the calculator based on an image of Apple's iPhone app

APIs

Getting the hang of a new API can be a challenge. First off, you've got to wade through the documentation, which may be dense and technical. And it's hit or miss with these docs—some are clear and easy to follow, while others...not so much. Then there's the deal with logging in and getting access, like using API keys or OAuth tokens. Plus, trying to figure out the data structures and response formats the API spits back at you can be overwhelming. Then there is error handling as well as having to keep up with any updates or changes in the API.

Fortunately, using ChatGPT can make tackling these challenges a bit easier.

To see how, let's take a simple example. Suppose you want to use an API to get information about the weather. You can ask ChatGPT for API suggestions:

> *Prompt:* What are good weather APIs?

One that ChatGPT recommends is OpenWeather API. Next, ask the following:

> *Prompt:* How do I start using the OpenWeather API?

ChatGPT describes the main steps.

You can then ask it to show how to make a request:

> *Prompt:* How do I create a request using the OpenWeather API? How do I do this using Python?

ChatGPT will go through the process, providing details on these steps:

- Import the requests library.
- Define the API key.
- Create the API request URL and use the current weather data endpoint.
- Execute the request and handle the response.
- Check to see if the request was successful.

Conclusion

In this chapter, we dug into modular programming, functions, and object-oriented programming. We also explored how to use AI to learn new languages and frameworks on the fly as well as how to use it to work more efficiently with data. Then we saw how to create compelling frontends.

But it's important to repeat: AI-assisted programming tools are far from perfect. That's why it's important to keep your eyes peeled and double-check everything.

CHAPTER 9

Debugging, Testing, and Deployment

This chapter is about the parts of software development that often do not get enough love: debugging, testing, and deployment. It's a guide to everything from spotting different bugs, to writing docs that people can actually understand, to making sure your code does what it's supposed to. We'll also dive into how to merge changes without a hitch, make your software run smoothly and safely, take in what users say, and even how to make a splash when you launch.

Debugging

When creating code, a developer will spend around 35% to 50% of their time on debugging. This is not just a time sink. It also eats up a big chunk of the budget in a software project.

There are two main types of bugs. First, there are *syntax errors*. These pop up when your code doesn't play by the rules of the structure of a language. It can be as simple as forgetting to add a colon at the end of a `for` loop. A modern IDE like VS Code should detect and fix many of these types of errors.

Next, there are *logical errors*. These are trickier because they happen when something in your logic is off. For example, suppose you have created a program to filter out adult users from a list based on their age. The only problem is, instead of excluding everyone aged 18 and above, your code mistakenly excludes those under 18. So, you end up with a list full of adults instead of teens. This mix-up is a prime example of a logical error. Your code is doing the exact opposite of what you intended. Pinpointing why it's flagging the under-18 crowd, instead of those 18 and above, can be quite the puzzle. The fix can range from making a quick adjustment in the age condition to having to rethink the whole logic.

Regardless, using an AI-assisted programming tool is not something you should start off with when debugging. Often, traditional approaches should be fine. VS Code provides powerful debugging features that can detect and resolve problems. After all, you can easily set up breakpoints to inspect variable values, walk through the code line by line, and see what's exactly happening. These are a lifesaver, especially in large programs.

But of course, some bugs can be real head-scratchers. Modern software is often a complex puzzle with many layers and parts that have to work together. Sometimes a bug pops up because these parts interact in unexpected ways, and figuring out what's actually going wrong can be a real challenge. Then there's the issue of documentation—or the lack of it. When software doesn't come with clear instructions or explanations, it's tough to understand how it's supposed to work, which makes finding and fixing bugs even harder. Your software will usually rely on external libraries or services, and if something goes wrong there, the bugs in your software can be maddening.

So this is where you can turn to AI. For example, one scenario is deciphering cryptic or long error messages. These may be stack traces, for instance, which are snapshots of what the program was doing when everything went haywire. You might also get detailed information about the interactions among various frameworks and libraries.

What you can do is copy-and-paste the error message into a prompt and include something simple like:

> *Prompt:* What does this mean? {error message}

Or, suppose you have code and there is a logic error. The program simply will not do what you want it to do. In this case, you can copy-and-paste the code into the prompt. Then include this:

> *Prompt:* This program is supposed to allow users to upload photos and display them in a gallery. However, when it is executed, the photos are not appearing. What is the problem with this program? {code}

If ChatGPT finds the problem, it will often suggest a solution. But if it does not, you can add this instruction to the prompt.

Documentation

Documentation is the glue that keeps everything from falling apart, but sadly, it's usually shoved to the back burner. We've all seen it—working with code without a guide is like wandering in a maze, and it's a pain, especially for new folks or anyone trying to figure things out.

Good documentation helps to prevent guessing games and keeps everyone on the same page. A survey from Stack Overflow says that 68% of developers bump into these knowledge roadblocks every week. Plus, GitHub's report from 2021 showed that sharing information in documentation can make teams up to 55% more productive.

But it's not just about making developers' lives easier. Good documentation is the backbone of a smooth-running development process. It's akin to a map that shows you where to go and what to watch out for. Without it, you're often flying blind, and that's when costly mistakes happen.

With ChatGPT, you can create any type of documentation, such as:

- User manuals
- README files
- API documentation
- FAQs
- Troubleshooting guides

When developing prompts for creating effective documentation, here are some factors to keep in mind:

Know your audience

Think about who's going to use your docs. Are they newbies looking for a guide to get started quickly, or are they tech wizards needing the nitty-gritty details like API docs and code snippets? Getting a bead on what your readers need is key to hitting the mark with your content.

Keep it simple

Nobody likes to wade through jargon or tech-speak that needs a decoder ring. Keep your words straightforward and to the point. If your grandparent couldn't get it, you're probably not there yet.

Stick to the plan

Keep your docs looking familiar. Use the same style, headings, fonts...you name it. Consistency is your friend. It's like having a good rhythm in a song. It just flows better.

Show, don't just tell

People understand things faster when they see it in action. So, throw in real-life examples or scenarios. If you're talking to coders, some code examples are gold. For others, screenshots or step-by-step walkthroughs can be helpful.

A picture is worth a thousand words

Sometimes, text just doesn't cut it. Use diagrams, screenshots, or even videos to get your point across, especially when you're dealing with complex stuff.

The why matters as much as the how

Don't just lay out the steps. Tell your readers why they're doing what they're doing.

With all this in mind, you could use a prompt like the following:

> *Prompt:* Can you suggest how I might explain the concept of version control in Git to a non-technical audience? I need to keep it simple and avoid technical jargon. Also, explain the importance of version control and suggest any visuals or diagrams.

ChatGPT has deep capabilities for language translation. You can certainly leverage this with your documentation or any other content.

Microsoft has been cooking up a system called GitHub Copilot for Docs, which is for the enterprise edition. It's not your average, run-of-the-mill tool for digging through docs. For example, the search results and responses are based on a user's coding background and experience. It is also updated with the latest on GitHub's repositories. It even gives you the ability to add private documentation. Essentially, this is a highly sophisticated knowledge base that can greatly boost your coding.

Code Review

Think of a code review as your code's test drive before it gets a pull request in the codebase. You make sure everything's running smoothly, fits in just right, and won't go kaput down the road.

But the process is about more than just looking for clunky bits or glitches. It's helpful for everyone to huddle around, bat ideas back and forth, and learn from each other. You'll see different ways to tackle a problem and get a better grip on the whole project.

In the meantime, a code review can help enforce an organization's coding style and guidelines. Then there's the security check. You see, automated tools don't always catch everything. Sometimes, it takes a human eye to spot those sly security risks.

As for ChatGPT, it can be a key part of this process. Here's an example prompt:

> *Prompt:* Write a code review for the code below. Keep a focus on the maintainability of the code, potential security issues, and performance flaws. {code}

I intentionally gave ChatGPT a poorly written function, yet ChatGPT did a good job with its review. It suggested numerous areas for improvement, such as that the function could benefit from not having hardcoded database connections. ChatGPT also detected the potential for a SQL injection due to the direct concatenation of the user ID in the SQL query and a lack of user input validation. It then found a performance issue with the construction of the SQL.

Unit Tests

In a way, unit tests are mini evaluations for parts of your code, say for a few functions or methods. Developers often do this testing themselves using cool tools like JUnit for Java, NUnit for .NET, or pytest for Python. These tools help write and run tests, and tell you the results. They usually play nice with other software tools you're using.

Doing unit testing is helpful because it makes your software better, cuts down on pesky bugs, and makes it easier to tweak and fix problems later. Each test focuses on just one thing, so if something goes wrong, you know exactly where to look. These tests are usually automated, which means they can be run fast and often. This is important for keeping everything smooth and up-to-date.

Unit tests are usually straightforward to write. Since they focus on small parts of the code, they shouldn't be too complicated. Plus, they're like a guide to your software. By checking out the unit tests, other developers can get how certain parts are supposed to work. If you change your code, unit tests are great for making sure you haven't messed up something that was working fine before.

Let's take a look at an example. Suppose you have created a tip calculator program like this one:

```
def tip_calculator(bill_amount, tip_percentage):
    tip_amount = bill_amount * (tip_percentage / 100)
    total_amount = bill_amount + tip_amount
    return total_amount

bill_amount = float(input("Enter the bill amount: "))
tip_percentage = float(input("Enter the tip percentage (e.g., 20 for 20%): "))
total_amount = tip_calculator(bill_amount, tip_percentage)
print(f"Total amount including tip: {total_amount:.2f}")
```

This has a function that calculates the total bill amount including a tip and takes two parameters, `bill_amount` and `tip_percentage`. The function will find the value for `total_amount`.

For code like this, unit tests check all sorts of scenarios. For instance, they can help verify that the function correctly calculates the total amount, including tips, for a range of inputs. This is important because even a small error can lead to significant discrepancies. Unit tests can cover typical cases, such as standard bill amounts and tip percentages, as well as edge cases, like a zero or negative bill amount or unusually high tip percentages. They also help to ensure that the function handles wonky inputs, such as non-numeric or null values, gracefully.

Here's a sample prompt:

> *Prompt:* For this program, suggest unit tests to correctly calculate the total amount, check the typical and edge cases, such as zero and negative bill amounts and high tip values. Also check for invalid inputs. For the unit tests, you can have console logs.

Figure 9-1 shows the code that ChatGPT created for the unit tests.

If you want to use a testing framework, to allow for a more structured and comprehensive approach, you can ask ChatGPT for this:

> *Prompt:* Create unit tests that use a testing framework.

ChatGPT suggests using unittest. It shows how to set it up, develops the tests, and demonstrates how to run the unit tests.

Or, if you already have a file with unit tests, you can evaluate them. Here's a prompt:

> *Prompt:* Below is the file for unit tests for a program that <*explain what it does or point to the code*>. Any other tests I should have? What is missing here?

While ChatGPT or an AI-assisted programming tool can be useful with creating these, the tools are far from foolproof. For more advanced use cases or larger codebases, the results can be off.

Consider the following advice from David Lee, who is a founder, senior engineer, and AWS-certified solutions architect:

> However, when dealing with tests involving real database interactions and Docker, the dynamics change significantly. It becomes another level of sophistication that ChatGPT 4.0 may not be able to comprehend to some extent, and you probably need to write a few tests manually first, especially the database connection part so that it can learn how to write others.

```python
def run_tests():
    # Typical case
    assert tip_calculator(50, 20) == 60, "Typical case failed"

    # Zero bill amount
    assert tip_calculator(0, 15) == 0, "Zero bill amount case failed"

    # Negative bill amount
    # Depending on your function's intended behavior, update the expect
    assert tip_calculator(-50, 20) == 0, "Negative bill amount case fai

    # High tip percentage
    assert tip_calculator(50, 100) == 100, "High tip percentage case fa

    # Invalid input
    # This requires exception handling in the tip_calculator function
    try:
        tip_calculator("fifty", 20)
        print("Invalid input case failed")
    except ValueError:
        print("Invalid input case passed")

    # Boundary cases
    assert tip_calculator(0.01, 0.01) == 0.01, "Boundary case failed"
    assert tip_calculator(100000, 50) == 150000, "Large amount boundary

    print("All tests passed!")

# Uncomment below to run the tests
# run_tests()
```

Figure 9-1. ChatGPT created code for unit tests on a tip calculator program

Pull Requests

A *pull request*, or PR for short, is like a golden ticket in the coding world, especially when you're working with others. You wrap up your code, push it to somewhere like GitHub or GitLab, and send out a PR. It's not just a "please add my code" request, though. It's also a nudge to your teammates to check out what you've done, give it a thumbs up, or maybe throw in some pointers to make it even better. It's all about making sure that when your code joins the rest of the project, it's the best it can be. What's more, PRs are a neat way to keep track of who did what and when, which is certainly helpful in big projects.

No doubt, writing a solid PR description really makes a difference. You want to lay out the what, why, and how of your changes. Start with a quick rundown of the problem you're tackling. Then dive into how your changes fix this. Don't forget to throw in details like which files got a makeover or any tests you ran. If there's something specific you're unsure about or need a second opinion on, mention that too.

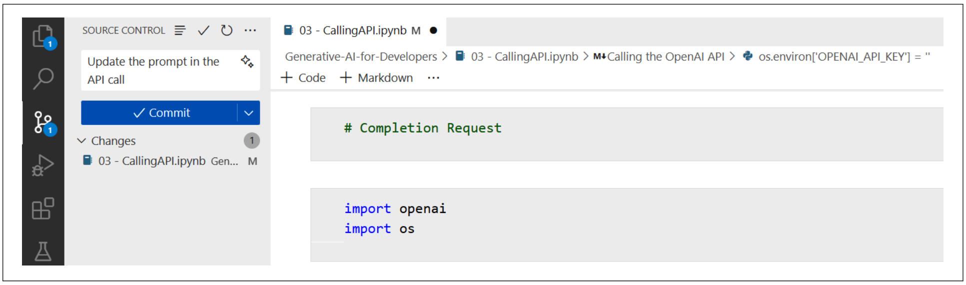
An effective PR description is a lifesaver for your reviewers. It speeds up the whole process and keeps everyone on the same page. Moreover, detailed PR descriptions are like a treasure trove of information for the future.

And yes, ChatGPT can be your sidekick in nailing those PR descriptions. Need to kickstart a draft? Tell ChatGPT what you did, and it will help you structure it into something clear and to the point. If you've already written a description, ChatGPT can check it out for clarity and grammar and suggest ways to make it even better. ChatGPT can also help you figure out the best way to lay out your PR, like starting with a summary and then getting into the nitty-gritty. And if there are technical details that need simplifying, it's got your back in making the PR more understandable for everyone in your team.

Let's take a look at some helpful prompts:

> *Prompt:* I added a new search feature to the application that filters results based on user input. Can you help me write a PR description for this?
>
> *Prompt:* I fixed a bug where the app crashed when a user entered special characters in the text field. How should I describe this in a PR?
>
> *Prompt:* I refactored the authentication module to improve performance and readability. What should I include in the PR description?
>
> *Prompt:* I updated the user interface to make the navigation more intuitive and added new icons. Can you help me draft a PR description?
>
> *Prompt:* What's a good way to phrase this in a PR description?
>
> *Prompt:* I added new unit tests for the payment-processing module. Can you assist me in writing a PR description that highlights these changes?
>
> *Prompt:* I resolved merge conflicts that arose due to recent changes in the main branch. What should I mention in the PR description about this?

Keep in mind that Microsoft has added a feature in Copilot that allows for creating PR descriptions. It's called Generated Commit Message. To use it, you'll need to make sure you have a connection to the repository on GitHub. Then you'll just need to click the sparkle button, as you can see in the lefthand panel in Figure 9-2.



Update the prompt in the API call

```
# Completion Request
```

```
import openai
import os
```

Figure 9-2. Copilot can create PR descriptions based on the repository that is loaded into your project

Copilot will then write up a useful PR description.

Finally, there are numerous startups that are creating their own systems based on LLM technologies. One is called What The Diff. According to the company's cofounder and CEO, Sebastian Schlein:

> There are two main features of What The Diff: 1) Making pull requests easier to review by summarizing them into plain English and giving reviewers and easy-to-understand overviews about the changes within the PR. 2) WTD can also write summaries that are completely non-technical and that get sent to other stakeholders like product managers who don't have GitHub access. That makes it easy for them to see if the change in the pull request actually matches their specifications.

Deployment

You've been grinding away on your software, and now it's go-time—the big launch. It's a thrilling moment. You're about to see real people using your software, telling you what rocks and what could use a tweak or even more. That kind of feedback? Priceless for a developer.

If you're in the business of selling your software, this is the moment you've been waiting for—when the cash starts rolling in. Nailing the launch can be a real game-changer for your bank balance.

But let's be real: deploying software is like holding your breath and hoping for the best. There's always this sneaky feeling that something might not go as planned.

You know how things seem perfect in your test setup, but then in the real world, they get a bit wobbly? It could be different hardware, some funky network systems, or just odd settings that mess things up.

Security is a big deal, too. Once you're live, you must be on your toes to keep the bad guys out and play nice with privacy rules.

Also, your software has to be tough enough to handle the crowd. It's got to stay quick and smooth, no matter how many people jump into it or how big your business gets.

And then there's the whole CI/CD thing. It's all about making deployments smooth and automatic to dodge mistakes. Sounds great, but getting it up and running, and keeping it that way, is a bit of a hustle.

Then what to do? You can definitely check out ChatGPT. The following are just a few of the prompts to consider:

> *Prompt:* Can you guide me in creating a deployment checklist for my team to follow?
>
> *Prompt:* What are some good learning resources for getting started with Docker for deployment purposes?
>
> *Prompt:* Can you provide best practices for zero-downtime deployment in a web application?
>
> *Prompt:* I'm encountering a "server timeout" error during deployment. What are some common causes and solutions for this?
>
> *Prompt:* Could you help me write a bash script for automating the deployment of my Python web app?
>
> *Prompt:* What are the essential configuration settings I should check before deploying an app in a production environment?
>
> *Prompt:* How do I plan a rollback strategy for a failed deployment in a cloud environment?
>
> *Prompt:* What security measures should I consider during the deployment of a financial application?
>
> *Prompt:* How can I optimize the performance of a deployed Node.js application?

While not perfect, ChatGPT can be helpful with complex DevOps. Here are some thoughts from Titus Capilnean, who is the cofounder and chief product officer at Private Market Labs:

> Cloud logs are not the easiest things to work with, especially as I'm not a devops engineer, but we have to deal with them, given that we are running processes on AWS and Google Cloud on a regular basis.
>
> One time, I had to set up tracking for a SQS + Lambda process, based on the output of a large-scale function deployment. My function was essentially printing a status in the logs and I planned to use that status to generate a report of that process—in our case, deal aliveness. I asked ChatGPT to provide an AWS CloudWatch query script and tuned it to the point where I could just run it at the end of each process to get my results. Reading the documentation in detail for this task would have taken probably 5–6 hours, so I saved a ton by going the GPT route.

> Similarly, I had to set up some alerts in Google Cloud, and I worked with ChatGPT to create a query that excluded some system-level errors we weren't actually responsible for and were not user-facing. It saved me hours of reading and work, and it provided me with the custom metric I needed to set up a useful alert system for our team.

User Feedback

User feedback is key to making your software rock. When your users see you're tuning in and making changes based on their thoughts, they're more likely to stick around and be all smiles. It's smarter to iron out the kinks early with their help, rather than try to clean up a big mess later when a lot of (increasingly unhappy) folks are using your app.

Even with all the testing, some bugs are sneaky, only showing their faces when your software hits the real world. Your users are like your own personal detectives, spotting the stuff that might have slipped past you.

Sometimes your users get really inventive with your software, using it in ways you never dreamed of. Their wild ideas can light the spark for new features, or even brand-new products.

Of course, there's a whole toolbox to boost customer service. Just some of the tools include Zendesk, Freshdesk, Drift, and Salesforce. They are handy with everything from live chat that lets you talk to customers in real time to feedback forms that gather insights on what your users think. Automated ticketing systems keep track of customer issues, ensuring that nothing slips through the cracks. And don't forget about customer relationship management (CRM) systems, which keep all customer interactions in one place. These tools can really make a difference in providing efficient, responsive customer service.

Generative AI can definitely add value on top of all this. It is particularly good at processing large amounts of unstructured data like user feedback.

For example, suppose you have a file that includes lots of emails, IMs, and contact form information from users. You can then go to ChatGPT and use this prompt:

> *Prompt:* Identify the common themes and categories, such as usability, performance, features, bugs, and customer service. Also, do sentiment analysis on this file. Based on the frequency and severity of issues mentioned, help prioritize which bugs to fix first or which features to consider adding. With all this, create a report that includes charts.

Another way ChatGPT can help is with response drafting. If you're dealing with customer feedback, it can help you write better responses, such as by creating templates for answers to common questions. This way, you're always on point and professional when you're chatting with users or customers.

Or, you can use ChatGPT for more personalized responses. You can cut-and-paste user email and use a prompt like this:

> *Prompt:* Write a reply to the user email, focusing on a calm and understanding tone. Make sure the response is friendly and helps to ease any concerns without escalating the situation. {email}

In some cases, you can create your own LLM-based application to handle user feedback. This is what Warp did. The company had a developer spend less than a week—at half-time—to create the app. It was built using the OpenAI API.

"The app has made a huge difference," said Noah Zweben, who is a product manager at Warp. "Before, it was difficult to categorize and prioritize the incoming feedback. But the generative AI has been able to do this extremely well."

The Launch

Several years before ChatGPT became a big deal, generative AI was already making an impact on important activities like sales and marketing. The trailblazer was Jasper. The company grew at a staggering rate. Then again, generative AI is ideal for whipping up catchy and creative content—fast.

But you don't need Jasper for your software launch. ChatGPT should work just fine. First of all, you can start mapping out a killer marketing plan with it. Here's a sample prompt:

> *Prompt:* You have created an app to help people plan healthy meals. It creates custom meal plans based on dietary preferences, health goals, and nutritional needs. It can also generate shopping lists, offer recipe suggestions, and track nutritional intake. For this app, put together a marketing plan. The company is an early-stage startup and does not have much resources for a marketing budget.

ChatGPT first recommends that you identify your target audience. It suggests that the primary users are "health-conscious individuals, fitness enthusiasts, people with specific dietary needs (e.g., gluten-free, vegan), busy professionals, etc." It then covers various strategies, such as leveraging social media, content marketing, community engagement, email marketing, and partnerships.

Here are some other helpful ChatGPT prompts:

> *Prompt:* Write an engaging introduction for a blog post announcing the launch of the new health planner app, highlighting its unique features and benefits.
>
> *Prompt:* Craft a series of social media posts for announcing our new health planner app, focusing on its user-friendly interface and how it helps in managing health goals.
>
> *Prompt:* Compose a product announcement email for our health planner app, emphasizing its ability to track and improve users' health routines.

Prompt: Create a persuasive sales email targeting gym instructors and health coaches, promoting our health planner app as a tool for their clients.

Prompt: Generate a list of catchy and relevant names for a new health planner app that conveys a sense of wellness and organization.

Prompt: Develop ad copy for Facebook and Instagram ads promoting our health planner app, highlighting its ease of use and personalization options.

Prompt: Formulate a template to request testimonials from early users of our health planner app, to be used in marketing materials.

Prompt: Create an invitation for a virtual launch event of our health planner app, detailing the agenda and special guests.

Conclusion

This chapter is like a behind-the-scenes look at the things in software development that don't always get the spotlight. Sure, creating new software is thrilling, but it's the less flashy tasks like debugging, testing, and documentation that really make or break your app. We've talked about how AI tools, like ChatGPT, can make these jobs a lot smoother. These AI buddies aren't magic fixes, but they're pretty awesome at sifting through tons of data, giving advice, whipping up content, spotting problems, and speeding up the whole process. This means developers get to focus on the really tricky stuff. By bringing generative AI and some smart strategies into every step, from squashing bugs to launching, developers can build better software that really hits the mark for users—and do it quicker too.

CHAPTER 10
Takeaways

This chapter is about giving you a quick rundown of the main ideas I'd like you to take away from this book.

The Learning Curve Is Steep

AI-assisted programming, while an exciting frontier in technology, presents a significant learning curve that can be challenging for even the most seasoned developers. This challenge is amplified by the rapid pace of advancement in the tech industry, where innovation is constant and new developments are constantly emerging. Staying abreast of these changes can feel overwhelming, akin to trying to catch a swiftly moving bullet train on foot.

One of the key challenges lies in adapting to the nuances of working with large language models, which are far more complex and unpredictable than traditional programming methods. Developers who are accustomed to conventional coding have come to expect a linear process where the code behaves predictably and executes exactly as written. However, with AI models, there's an element of unpredictability and autonomy that can be both baffling and exhilarating. These models often process and respond to coding requests in ways that are not immediately intuitive to human programmers.

This shift requires developers to adopt a new mindset. They must learn to anticipate and interpret the often unexpected outputs of AI models, and this departure from the straightforward, logical processes they are used to can be uncomfortable at first.

There Are Major Benefits

With traditional coding, when you hit a snag, you usually have to stop everything and go on a wild goose chase through Stack Overflow or dig through tons of documentation or search Google. But AI-assisted programming tools are like buddies who can throw you a lifeline. These tools pop up with suggestions and fixes as you type, so you can keep your head in the game and not get lost switching between a bunch of tabs or apps. You can get in the "flow" and stay there.

Some of the tools are smart enough to pick up on your coding style and the names you use for variables, functions, and methods. They get the whole vibe of your code. So, when they make a suggestion, they're not just handing you some random piece of advice. Their wisdom is tailor-made for your project.

Now let's talk about the real value proposition: saving you from the boring stuff. These are the same old routine tasks that make you want to snooze—file handling, data juggling, API calls, UI stuff, regex, starter code, those aggravating bash commands, and messing with GitHub Actions, just to name a few. AI in programming is like, "Don't worry, I got this," and just whips up the code for you.

And it doesn't stop there. The AI is also a whiz at making sense of your code. For one thing, it can automatically document all of it. Instead of you burning the midnight oil, trying to jot down notes and explanations, the AI sorts it all out in plain English. This is certainly important, especially when you're bringing new people into the project or revisiting old code. The AI's got it all laid out.

But There Are Drawbacks

AI-assisted programing has some sticky points, too. First up, there's the whole mess about who owns what when it comes to the code AI tools spit out. These tools learn from a bunch of code from everywhere, and some of it might be copyrighted. If the AI buddy churns out something that looks a bit too much like something that already exists, is that stepping on someone's toes? And if you make something awesome with AI's help, who gets to cash in on it? The whole legal side of things is complicated, and it will likely take time for some consensus to form around it.

Security's another thing to watch out for. Turns out, sometimes the code that comes out of these AI tools can have security gaps. Since we don't really get a peek into how an LLM thinks up its magic—it's pretty much a mystery box—it's hard to be sure about what you're getting. This means you can't just take the code and run with it. Instead, you've got to put it through the wringer with tests and checks before you can actually use it.

Then there are the potential issues with privacy. AI tools might inadvertently learn from private code repositories or proprietary data if they're not carefully managed.

Moreover, the data a developer feeds into these AI models could contain personal information.

The challenge is to ensure that AI programming assistants are trained and used in a way that respects privacy and confidentiality. This means implementing robust data handling and privacy policies and ensuring compliance with regulations like GDPR.

Let's not forget also that sometimes AI-assisted programming tools can get things wrong. Or the code may be far from optimal or be verbose. This is due to factors like the varying quality of the underlying training data. Then there are the unpredictability and complexities of the LLMs. You never quite know what you're going to get.

Prompt Engineering Is an Art and Science

Learning prompt engineering is far from easy. It's an art, since you need a knack for picking the right words to guide the AI. This is often about being creative in how you ask questions or set the scene.

But prompt engineering is a science, too. You need to geek out on how AI models work. You're trying to become more and more precise in your guesses about how the AI will react to your prompts. It's like being a detective, running experiments, checking what happens, and tweaking your prompts to get the answers you want.

Fortunately, there are some guidelines that can help out. For example, the length of the prompt is key. If your prompt is too long, the AI might get confused or miss the point. You also need to be specific. "A prompt needs to be clear and focused," said Ankit Anchlia, a veteran software engineer. "There must be sufficient context. If not, you'll likely not get the response you want."

Beyond Programming

AI-assisted programming tools are not just about coding. They've got a whole bunch of tricks up their sleeve. In this book, we've seen that they are like Swiss Army knives for all sorts of tasks, from brainstorming cool ideas and planning projects to digging into market research and jotting down what you need for a project. For instance, imagine zipping through writing top-notch product requirements documents and software requirements specifications. AI can help you with these tasks, and even do them better.

And when you're ready to roll out your product, guess what? AI's there to help with that too. It's like having a marketing guru in your pocket. You can use AI tools to whip up a marketing plan that gets your product out there and noticed. You can also use AI for analyzing user feedback, as you make your application better and better.

AI Won't Take Your Job

Of course, there's the big worry that AI's going to swoop in and steal everyone's jobs. The world is heading toward some sort of Skynet scenario where the machines take over, right? And all that effort you put into learning to code—is it just going to be for nothing? Well, not exactly.

Here's the deal: AI-assisted programming tools are powerful, but they're not here to replace you. They're more like your sidekicks, there to make you an even better coder. They don't have the smarts or independence of a real developer.

But, and it's a big but, if you're not using these tools, you might find yourself falling behind. More and more, employers are expecting their developers to work with these AI systems. Why? Because the benefits are just too good to pass up. We all need to keep up with the times. Using AI tools is becoming a must-have skill, not to keep them from replacing you (they can't) but because they'll help you do your job way better.

According to James Clift, who is the founder and CEO of Durable:

> Advancements in AI will lead to shifts in the dynamics of the labor market, but it's important to remember that AI and business are better together. One does not replace the other. The key is to not be fearful but to embrace the technology to support further business growth and job creation. AI tools put resources once only available to large companies into the hands of everybody.

Conclusion

We've covered a ton of ground in this book, diving into the ins and outs of AI-assisted programming. But even so, we're just scratching the surface! We're in the early days of this AI journey, and the cool part is, it's only going to get better from here. The technology is evolving at breakneck speed, which means there's going to be even more awesome tools and tricks for developers to get our hands on. We're at the starting line of a really exciting race, and the possibilities are just ramping up.

It's a thrilling time to be in this field. Imagine all the new ways we'll be able to sharpen our skills, streamline our work, and create things we haven't even dreamed of yet. The future of AI-assisted programming is bright and full of potential, so let's keep our eyes peeled and be ready to embrace all the amazing advancements coming our way. Here's to riding the wave of AI innovation and seeing where it takes us as developers.

Index

About the Author

Tom Taulli (@ttaulli) is a consultant to various companies, such as Aisera, a venture-backed generative AI startup. He has written several books like *AI Basics* and *Generative AI*, which cover ChatGPT, GPT-4, and other large language models. Tom has also taught IT courses for O'Reilly, UCLA, and Pluralsight. For these, he has provided lessons in using Python to create deep learning and machine learning models. He has also taught on topics like natural language processing.

Colophon

The animal on the cover of *AI-Assisted Programming* is a reef triggerfish (*Rhinecanthus rectangulus*). The animal is also referred to as *humuhumunukunukuapua'a*, or simply *humuhumu*. Following a popular vote and the approval of the Hawaiian state legislature, the reef triggerfish was selected as the official state fish of Hawaii in 1984.

The reef triggerfish is found in shallow outer reef habitats in the Indo-Pacific and Hawaii. Its diet consists of algae and reef invertebrates, such as sea urchins and snails, which the triggerfish finds by swimming close to the bottom of reefs.

Although the triggerfish tends to keep its distance from onlookers, its distinctive charging, territorial behavior and appearance—a plump mouth, blue top lip, and size of up to 10 inches—make it easier to observe.

There are approximately 40 species of triggerfish, and the reef triggerfish's current conservation status is "Least Concern." Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on an antique engraving from *Oeuvres du Comte De Lacépede*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.