

大模型辅助编程

[美]内森·B. 克罗克(Nathan B. Crocker) 著

郭 涛 译

清华大学出版社

北 京

北京市版权局著作权合同登记号 图字：01-2025-0995

Nathan B. Crocker

AI-Powered Developer: Build great software with ChatGPT and Copilot

EISBN: 978-1-63343-761-6

Original English language edition published by Manning Publications, USA © 2024 by Manning Publications Co. Simplified Chinese-language edition copyright © 2025 by Tsinghua University Press Limited. All rights reserved.

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。举报：010-62782989，beiqinquan@tup.tsinghua.edu.cn。

图书在版编目(CIP)数据

大模型辅助编程 / (美) 内森·B.克洛克

(Nathan B. Crocker) 著；郭涛译. -- 北京：清华大学出版社，2025. 9. -- ISBN 978-7-302-70305-1

I. TP18

中国国家版本馆 CIP 数据核字第 2025CJ6080 号

责任编辑：王 军

封面设计：高娟妮

版式设计：思创景点

责任校对：马遥遥

责任印制：杨 艳

出版发行：清华大学出版社

网 址：<https://www.tup.com.cn>，<https://www.wqxuetang.com>

地 址：北京清华大学学研大厦 A 座

邮 编：100084

社 总 机：010-83470000

邮 购：010-62786544

投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：北京联兴盛业印刷股份有限公司

经 销：全国新华书店

开 本：148mm×210mm

印 张：8.125

字 数：231 千字

版 次：2025 年 10 月第 1 版

印 次：2025 年 10 月第 1 次印刷

定 价：59.80 元

产品编号：105677-01

读者感受

★★★★★一本为现代软件工程师打造的卓越、实用指南。

——Colin Kirk

本书见解深刻，示例设计精巧且实用，行文流畅，阅读体验极佳。运用书中所学，我的工作效率得到了极大提升。

★★★★★一本引领你借助生成式 AI 革新软件开发的指南。

——Madhuri Lokesh

刚读完《大模型辅助编程》，本书彻底改变了我对软件开发的认知。作者将复杂的 AI 概念拆解得通俗易懂，让 ChatGPT、GitHub Copilot、CodeWhisperer 等工具真正落地、可上手。

最打动我的是书中紧贴实战的讲解：从设计到编码，每一步都有 AI 辅助的实例，我立刻就能搬进自己的项目。其中第 4 章介绍的用 GitHub Copilot 做领域建模、理解设计模式，让我大开眼界。

读完后，我更深刻地体会到将 AI 引入开发流程时，测试与安全的重要性。书中关于最佳实践的讨论，不仅教我“怎么用”，更教我“怎么负责任地用”。

强烈推荐给所有想在软件开发中借力 AI 的同行——生成式 AI 正在重塑传统开发方式，而本书就是最好的路线图。

总之，它彻底刷新了我对“AI+编码”的思考，带来了启发，我会在工作中继续深挖。

★★★★★解锁 AI 工具在软件开发中的强大潜力。

——Nilay

《大模型辅助编程》是一本极具实用价值的入门指南，全面介绍了如何将 ChatGPT、GitHub Copilot 和 CodeWhisperer 等 AI 工具应用于软件开发。本书以通俗易懂的方式解析复杂概念，即使你是 AI 增强开发领域的新手也能轻松掌握。

书中对提示工程的讲解令我尤为赞赏。详细介绍的“优化模式”帮助调整提示词，获取更佳结果，特别有趣的“角色模式”演示如何使 AI 输出保持一致性。书中对不同工具的适用场景对比十分精辟：ChatGPT 擅长原型设计，而 Copilot 能在明确需求后极大提升编码效率。

总之，若你希望在编码项目中借助 AI 提升生产力，本书提供的清晰实用的指导和即学即用的示例将为你提供立即可操作的解决方案。

★★★★★开发者使用 ChatGPT 的绝佳入门指南。

——Rami

本书为开发者介绍了 ChatGPT 在实际工作中的多种应用场景，内容通俗易懂，编排合理，是学习 AI 辅助开发的优质选择。

谨以此书纪念 Catherine L. Crocker,

她的力量和爱一直指引着我。

虽然她已不在我们身边，但她的精神和智慧永远与我们同在。

我写的每一个字都蕴含了她的思想。

她虽然离开了这个世界，却永远留在我们心中。

译者序

大模型时代催生了以自然语言交互的编程范式。从最早的汇编语言到其后的低级编程语言，再到高级编程语言，需要程序员根据业务需求和拟解决问题动手编写代码、调试代码并运行代码。此外，不同的场景催生不同的高级编程语言。例如，如果是开发操作系统、数据库和嵌入式系统，C++、Rust 是最佳的编程语言；如果是做 Web 开发，Java、Go 和 C# 是最佳的编程语言；如果是数据科学和算法建模，MATLAB、Python 和 R 是最佳的选择。这使得程序员需要掌握 2~3 门编程语言，学习门槛比较高。

大模型的出现彻底改变了这一局面。针对编程需求，目前已经出现了一批基于大模型形成的辅助编程工具，与 IDE 高度集成或者开发出 AI IDE。例如 Cursor IDE、Windsurf AI IDE、GitHub Copilot 和 Google AI Studio 等，这些都具备上下文引擎、多文件编辑、AI 集成和自然语言交互等方面的能力。这些工具不仅改变了传统的编程方式，还为开发效率、代码质量以及创新能力带来了新的可能性。

以大模型(ChatGPT、DeepSeek、Claude 等)作为编程引擎的 AI 辅助编程工具彻底改变了编程范式，由以前的程序员编程，到与 AI 工具结对编程，再到未来的 AI Agentic 编程(GitHub Copilot Workspace)，编程人员通过自然语言人机交互，从头脑风暴、规划、构建、测试和运行代码，很短时间内就可以实现自己的产品。

本书为程序员转型提供了指南。本书的出版正值这一领域发展的关键时期。本书深入探讨了如何将 AI 编程工具集成到软件开发过程中，提供了切实可行的建议和最佳实践案例。这不仅对资深开发者有所帮助，也为编程初学者提供了利用 AI 加速学习的机会。

本书结构清晰，内容丰富，从基础知识到高级应用，从理论介绍到实践案例，涵盖了 GitHub Copilot 和 ChatGPT 编程的方方面面。无论你是希望提高生产力的资深开发者，还是刚刚踏入编程世界的学习者，本书都能为你提供有益的指导。愿本书成为你的 AI 开发旅程中的一盏明灯！

对于开发者而言，合理选择工具至关重要。例如，GitHub Copilot 适合需要进行高效开发的团队和个人，而 Tabnine 更适合注重隐私保护的企业用户。同时，国内开发者应关注国产工具的发展，特别是在满足中文场景需求和合规性要求方面，充分利用华为 MindSpore Studio 和百度文心 Code 的功能。

本书附赠“Trac AI 编程工具入门与实例精讲”文档(可扫下面的二维码下载)，供读者参考和学习。



Trac AI 编程工具入门与实例精讲

在本书翻译过程中，西南交通大学外国语学院王艺锦参与本书的审校工作，感谢她为本书所作的一切努力。最后，感谢清华大学出版社的编辑，他们做了大量的编辑与校对工作，保证了本书的质量，使得本书符合出版要求，在此深表谢意。

由于本书涉及的内容广且深，加上译者翻译水平有限，在翻译过程中难免有不足之处，欢迎批评指正。

作者简介



Nathan B. Crocker 是 Checker 的联合创始人兼 CEO，Checker 以 API 为主提供解决方案，将传统资本市场基础设施与区块链生态系统连接起来。凭借其在构建数字资产基础设施方面的专业知识，Nathan 目前负责 Checker 的技术愿景和开发工作，构建其核心基础设施，从而在区块链上运行新的金融应用程序。

致 谢

撰写本书绝非易事，需要花费并投入大量时间，需要一丝不苟的努力。这是一条充满挑战的道路，但每走一步都是颇具价值的经历，让我更接近 AI 辅助编程这一广阔而迷人的世界。这段旅程如果没有杰出人士的支持和贡献，我无法开始，更无法完成。

衷心感谢我的编辑 Katie Sposato Johnson，她在本书的写作过程中起到了重要作用。她深刻的评论、批判性的见解和建设性的反馈帮助我提炼思想，并将其转化为条理清晰且引人入胜的文字。她坚定不移的付出和充满激情的投入对这个项目来说非常宝贵。

特别感谢我的技术编辑 Nicolai Nielsen，他是 SymphonyAI 的首席 AI 工程师，同时是一名程序员和内容创作者，在 YouTube 上制作 AI 和计算机视觉教育视频及课程，在帮助人们成长的同时也扩大他的品牌影响。Nicolai 的专业知识和对细节的敏锐眼光时刻警醒着我，不断提醒我在这一广阔的领域中还有许多知识要学。他的意见不仅具有教育意义，也让我感到谦卑，加深了我对该领域的理解并使我能够脚踏实地。

衷心感谢 Manning 出版社的所有人在这段旅程中给予我的不懈支持。他们的职业精神、合作态度和追求卓越的决心让我备受鼓舞。他们在本书的出版过程中发挥了重要作用，对此我深怀感激。

感谢所有审稿人：Carmelo San Giovanni、Chad Yantorno、Christopher Forbes、Dan McCreary、Dewang Mehta、Greg MacLean、Håvard Wall、Jeff Smith、Jim Matlock、Jonathan Boiser、Louis Aloia、Luke Kupka、Mariano Junge、Maxim Volgin、Maxime Boillot、Mike Piscatello、Milorad Imbra、Peter Dickten、Philip Patterson、Pierre-Michel

Ansel、Rambabu Posa、Rebecca Wagaman、Riccardo Marotti、Roy Wilsker、Stefano Pri-ola、Thomas Jaensch、Thomas Joseph Heiman、Tiago Boldt Sousa、Tony Holdroyd 和 Walter Alexander Mata López，你们的建议让这本书变得更好。

我最感谢的是我的家人，你们是我的精神支柱。感谢我的妻子 Jenn，谢谢你成为我的依靠，感谢你一直以来的耐心、理解 and 爱。感谢我的女儿 Maeve 和 Orla，你们是我的灵感源泉，你们的快乐、好奇心和无尽的热情激励着我前行。感谢所有支持我的家人，谢谢你们。

这本书是无数个小时的努力、投入和团队合作的结晶。我衷心感谢所有为本书做出贡献的人。谢谢大家。

关于封面插图

本书封面上的人物标题为 *Junger kroatischer Gebirgsbauer*，意为“年轻的克罗地亚山区农民”，取自 1912 年出版的历史和民俗服装插图集。该图集的每幅插图都由手工精细绘制和上色而成。

在那个时代，仅凭人们的着装就能轻易辨认出其居住地、职业或社会地位。**Manning** 出版社以几世纪前丰富多彩的区域文化为基础，并通过这类插图集中的图片重现往昔人们的精神风貌，以此彰显计算机行业具有的创造性和主动性。

序 言

欢迎阅读本书，这是探索编程与 AI 之间共生关系的门户。本书不仅讲述 AI 及其在软件开发中的应用，而且介绍通过前沿 AI 模型(如 ChatGPT 和 GitHub Copilot)的辅助探索编程的未知领域。翻开本书，读者将踏上探索与发现的旅程，以全新的视角发掘 AI 如何重塑和提升编程领域。

本书的精髓在于其非常规的指导方法。与大多数技术文献不同，它并不要求读者遵循固定的脚本或模式。因为本书涉及 LLM 在软件开发中的应用，在这个领域中，即使输入相同，输出也可能千差万别。与其说这是一张规划预定路线的地图，不如将其视为一个指南针，指引读者在充满可能性的迷人风景中前行。

本书鼓励做实验、提问题，最重要的是，对意外结果持开放态度。本书会引发好奇心，激发创造力，并促进读者提升解决问题的能力。像 ChatGPT 和 Copilot 这样的 LLM 领域不仅提供辅助编程，还提供了一个具有变革性的框架，有可能从根本上革新软件开发。

这本书本质上扮演了导师的角色，是推动超越传统编程熟悉边界的催化剂，鼓励探索 AI 与编程之间的复杂交互。它激发对生成式 AI 模型未知潜力的兴趣。通过大量真实案例、实践练习和深入了解，不仅能学会如何使用这些 AI 工具，还能更深入地了解其工作原理、潜力和局限性。

然而，与跟随其他导师一样，这段旅程的收获与激情、好奇心和投入成正比。通过深入研究、提问和质疑假设，读者不仅会获得技术技能，还将从更广阔的视角了解在 AI 时代成为开发人员意味着什么。

在软件开发领域，当今正处于令人振奋的时代。AI 和机器学习正在颠覆传统范式，提供新的工具和方法，这可以显著提高生产力、创造力和效率。将 AI 融入开发流程可以解决更复杂的问题，简化工作流程，并从根本上改变编写代码的方式。

本书不仅是一本书，还是通往新世界——一个将编程逻辑与 AI 的能力和灵活性相结合的世界——的大门。无论是经验丰富的开发人员还是充满热情的初学者，均可通过本书学习工具、技术和知识，以此在环境的不断变化中开辟自己的道路。

请记住，千里之行，始于足下。阅读本书，就已经迈出了这第一步。现在，让我们一起走进智能编程的精彩世界，享受这段旅程吧！

前言

本书是掌握 LLM(如 ChatGPT 和 CoPilot)如何与软件开发过程相集成的必备指南。本书提供了实用建议，展示了最佳实践，指导利用人工智能(AI)提升项目质量。从使用 AI 的注意事项到真实案例，读者将获得所需的见解和工具，来提升开发技能，并在不断变化的技术领域保持领先。

本书读者对象

无论是专业开发人员还是业余爱好者都能从本书中受益。虽然这本书主要面向有经验的开发人员，但大模型(Large Language Model, LLM)¹有助于快速学习，因为这些工具可以提供解释、代码示例以及编程概念的指导。有经验的开发人员可以利用这些工具提高效率，简化编程流程，更高效地应对复杂的编程挑战。这些工具还可以协助生成代码片段、调试，并提供关于最佳实践的见解。

本书内容

本书主要分为 4 个部分，并配备 3 个实用的附录，用于指导读者设置 3 个 AI 工具。

¹ 译者注：Large Language Model 中文翻译为“大规模语言模型”“大型语言模型”“大模型”，本书中文统一为“大模型”，英文统一为 LLM。

第 I 部分：基础

第 1 章介绍 LLM，追溯其历史并阐释了生成式 AI 的概念。该章还就如何恰当和谨慎地使用这些技术提出了建议。

第 2 章介绍如何开始使用 LLM，比较了 ChatGPT、GitHub Copilot 和 CodeWhisperer，并详细说明了使用它们的初步步骤。

第 II 部分：输入

第 3 章通过信息技术资产管理(ITAM)系统项目示例，讲解如何在 ChatGPT 的辅助下设计软件。

第 4 章着重介绍如何使用 GitHub Copilot 构建软件，涵盖了诸如领域建模、不可变性和设计模式等基本概念。

第 5 章深入探讨如何使用 GitHub Copilot 和 Copilot Chat 管理数据，包括使用 Kafka 进行实时资产监控以及使用 Apache Spark 进行数据分析。

第 III 部分：反馈

第 6 章讨论使用 LLM 开发的软件的测试、质量评估和解释过程，包括漏洞查找和代码转换。

第 IV 部分：走向世界

第 7 章涵盖从构建 Docker 镜像到使用 GitHub Actions 设置持续集成/持续部署(CI/CD)管道的基础设施编程和管理部署知识。

第 8 章讨论如何使用 ChatGPT 开发安全应用程序，包括威胁建模和安全最佳实践应用。

第 9 章探讨“随时随地使用 GPT”的概念，包括托管自己的 LLM 以及通过 GPT-4All 实现访问民主化。

附录提供设置 ChatGPT、Copilot 和 CodeWhisperer 的简单指导，确保具备开始 AI 辅助编程所需的操作知识。

除最后一章外，本书应按顺序阅读，因为每一章都是在前一章的基础上构建的。

关于代码

读者可扫描下面的二维码，下载本书示例的完整代码。



目 录

第 I 部分 基础

第 1 章 了解 LLM	3
1.1 加速开发	4
1.2 LLM 介绍	10
1.3 何时使用或避免使用生成式 AI	11
1.4 本章小结	13
第 2 章 使用 LLM	15
2.1 ChatGPT	16
2.1.1 使用 GPT-4 处理细微差别	16
2.1.2 使用 GPT-3.5 绘制路径	22
2.1.3 在 AI 海洋中航行：从 GPT-3.5 “海岸” 到 GPT-4 “天际线”	25
2.2 Copilot	26
2.3 CodeWhisperer	29
2.4 比较 ChatGPT、Copilot 和 CodeWhisperer	31
2.5 本章小结	33

第 II 部分 输入

第 3 章 使用 ChatGPT 设计软件	37
3.1 项目介绍：ITAM 系统	38
3.2 请 ChatGPT 协助系统设计	38

3.3	记录架构	43
3.4	本章小结	62
第 4 章	使用 GitHub Copilot 构建软件	63
4.1	奠定基础	64
4.1.1	表达领域模型	64
4.1.2	优先采用不可变性	66
4.1.3	修饰最喜欢的类	68
4.1.4	调整折旧策略	73
4.2	编织模式	75
4.2.1	访问部门	76
4.2.2	使用工厂(模式)创建对象	77
4.2.3	指导系统构建	82
4.2.4	观察变化	87
4.3	插入端口和适配器	90
4.3.1	六边形架构回顾	91
4.3.2	驱动应用程序	92
4.3.3	访问数据和持久化更改	100
4.3.4	集中(和外部化)数据访问	104
4.4	本章小结	109
第 5 章	使用 GitHub Copilot 和 Copilot Chat 管理数据	111
5.1	构建数据集	112
5.2	使用 Kafka 实时监控资产	123
5.3	使用 Apache Spark 进行分析、学习和跟踪	133
5.4	本章小结	138
第 III 部分 反馈		
第 6 章	基于 LLM 的测试、评估和解释	143
6.1	3 种测试类型	144

6.1.1 单元测试	144
6.1.2 集成测试	151
6.1.3 行为测试	152
6.2 评估质量	157
6.3 寻找错误	160
6.4 代码覆盖	162
6.5 代码转译——从代码到描述	164
6.6 从一种语言翻译到另一种语言	166
6.7 本章小结	173

第IV部分 走向世界

第7章 编写基础设施代码和管理部署	177
7.1 构建 Docker 镜像并“部署”到本地	179
7.2 使用 GitHub Copilot 协助 Terraform 构建基础设施	182
7.3 移动 Docker 镜像(困难模式)	186
7.4 移动 Docker 镜像(简单模式)	186
7.5 将应用程序部署到 AWS EKS	188
7.6 在 GitHub Actions 中设置 CI/CD 管道	191
7.7 本章小结	194
第8章 使用 ChatGPT 开发安全应用程序	197
8.1 使用 ChatGPT 进行威胁建模	199
8.1.1 威胁建模在当今开发环境中至关重要的原因	199
8.1.2 ChatGPT 如何辅助威胁建模	200
8.1.3 案例研究：使用 ChatGPT 模拟威胁建模	203
8.2 审查应用程序设计并识别潜在漏洞	207
8.2.1 评估设计问题	207
8.2.2 识别常见漏洞	208
8.3 应用安全最佳实践	209

8.3.1	建立安全意识	209
8.3.2	持续安全测试	210
8.4	静态数据和传输中数据的加密	213
8.4.1	数据加密的重要性	213
8.4.2	静态数据加密	214
8.4.3	传输中数据的加密	218
8.5	本章小结	220
第 9 章	随时随地使用 GPT	221
9.1	动机理论	221
9.2	本地托管 LLM	222
9.2.1	使用 ChatGPT 进行基准测试	223
9.2.2	要求 Llama 2 输出答案	224
9.2.3	用 GPT-4All 实现答案的民主化	232
9.3	本章小结	235
附录		237

第 I 部分

基 础

第 I 部分全面介绍 LLM 及其在现代软件开发中的重要性。该部分追溯生成式 AI 的历史演变，为这些强大的技术提供坚实的概念框架；同时强调负责任并谨慎使用的重要性，引导读者了解将 AI 融入工作流程中的基本原则和潜在隐患。此外，第 I 部分提供实用的建议，帮助读者着手使用 LLM，比较 ChatGPT、GitHub Copilot 和 CodeWhisperer 等流行工具，并详细介绍有效利用其功能的初始步骤。

第 1 章

了解LLM

本章内容：

- 介绍生成式 AI——LLM
- 探索生成式 AI 的优势
- 确定何时使用生成式 AI

无论你是否意识到，也无论是否愿意承认，有了生成式 AI，你便已经悄然晋升。每一位专业软件工程师都如此。几乎在一夜之间，从员工变成了经理。现在，你团队中拥有世界上最聪明、最有才华的初级开发人员——生成式 AI 成了你新的编程伙伴。因此，指导、培训和进行代码审查应该成为你日常工作的一部分。本章将介绍生成式 AI 的一个子集 LLM，特别是 ChatGPT、GitHub Copilot 和 AWS CodeWhisperer。

注意：这不是一本传统的编程书。不能像使用脚本一样使用它。与 LLM 进行对话时，就像其他任何对话一样，内容和方向会根据模

型和先前的语境而变化。输出很可能与书中的内容不同。你不应该因此而气馁；相反，应该去探索。旅程本身和目的地一样令人愉悦。你可能会因为无法跟上进度而感到沮丧。无论如何，要有耐心。如果有纪律性并且稍微有点冒险精神，则可以让 GPT 与本书的主题和目标相契合：学习如何使用生成式 AI 让自己成为一名更好的程序员。

1.1 加速开发

在软件开发的新时代中，开发团队增加了一位非常有才华的工程师。生成式 AI 不仅是一个工具，还是一名团队成员，能够将编程质量提升到新的高度。设想一下设计复杂的系统、前所未有的编程速度，以及难以想象的强大测试——这一切都离不开从最佳实践中学习的智能。本书将探讨生成式 AI 如何辅助完成日常编程任务，以及帮助实现以前无法企及的目标，确保获得更快的开发速度、更高的质量以及前所未有的创新能力。

LLM 和生成式 AI 可以极大地加速软件开发过程。将烦琐的样板代码创建自动化有助于专注于创造性的问题求解和增值任务。例如，要想让 LLM 创建一个 Flask 应用程序，可以使用以下提示。

NC 生成一个简化的 Flask 应用程序。

LLM 可能会输出如代码清单 1.1 所示的 Python 代码。

代码清单 1.1 由 LLM 生成的用于构建 Flask 应用程序框架的代码

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, world!"
```



```
if __name__ == '__main__':  
    app.run(debug=True)
```

LLM 是深度学习模型，擅长对自然语言提示作出自然语言响应。可以想象，只需要用简单的英语描述需求，就能收到可集成的代码。

NC 创建一个用于管理用户会话的 Python 类。

代码清单 1.2 是 LLM 可能作出的响应。

代码清单 1.2 创建会话管理类的代码

```
import uuid  
from datetime import datetime, timedelta  
  
class SessionManager:  
    def __init__(self):  
        self.sessions = {}  
  
    def create_session(self, user_id):  
        """Create a new session for a user with a unique session  
ID and expiration time."""  
        session_id = str(uuid.uuid4())  
        # 会话在 1 小时后过期  
        expiration = datetime.now() + timedelta(hours=1)  
        self.sessions[session_id] = {'user_id':  
user_id, 'expiration': expiration}  
        return session_id  
  
    def get_user_id(self, session_id):  
        """Retrieve a user ID from a session ID  
        if the session is still valid."""  
        session_info = self.sessions.get(session_id)  
        if session_info and session_info['expiration'] >  
datetime.now():  
            return session_info['user_id']  
        return None # 会话已过期或不存在
```

```
def delete_session(self, session_id):
    """Delete a session by its ID."""
    if session_id in self.sessions:
        del self.sessions[session_id]

def is_session_valid(self, session_id):
    """Check if a session is still valid."""
    return session_id in self.sessions and self.sessions[session_id]['expiration'] > datetime.now()
```

这些工具还能通过识别模式和建议修复来提高调试效率。假设想要 AI 工具分析一段代码并标记潜在的内存泄漏，它可能会生成代码清单 1.3。

代码清单 1.3 使用 LLM 在 Python 中查找潜在的内存泄漏

```
def process_data():
    large_data = [x for x in range(1000000)] # 大型数字列表
    result = sum(large_data)
    print("Result of processing:", result)
    # large_data 未清除或重新分配,
    # 可能导致记忆保留

process_data()
```

根据 ChatGPT 提供的解释，这段 Python 代码展示了一个潜在的内存泄漏问题，原因是一个大型列表在使用后未清除。如果引用管理不当，它在内存中保留内容的时间会超过实际需要。

如代码清单 1.4 所示，在改进的代码中，AI 工具建议在使用列表后，特别是在垃圾回收不积极的环境中，显式清空列表或将其重新赋值为 None，以便更快释放内存。

代码清单 1.4 用“改进”的 LLM 代码解决潜在的内存泄漏问题

```
def process_data():
    large_data = [x for x in range(1000000)] # 大型数字列表
```

```
result = sum(large_data)
print("Result of processing:", result)
large_data = None # 清除引用以释放内存

process_data()
```

此外，在涉及重构时，AI 会建议采取优化措施，使代码更简洁、更高效，如代码清单 1.5 和代码清单 1.6 所示。

代码清单 1.5 建议重构之前的冗长代码

```
class DataProcessor:
    def __init__(self, data):
        self.data = data

    def process_data(self):
        if self.data is not None:
            if len(self.data) > 0:
                processed_data = []
                for d in self.data:
                    if d is not None:
                        if d % 2 == 0:
                            processed_data.append(d)
                return processed_data
            else:
                return []
        else:
            return []

processor = DataProcessor([1, 2, 3, 4, None, 6])
result = processor.process_data()
print("Processed Data:", result)
```

重构后，代码更易读、更易于维护且更符合惯用法。

代码清单 1.6 更简洁的 LLM 重构代码

```
class DataProcessor:
    def __init__(self, data):
```

```
self.data = data or []

def process_data(self):
    return [d for d in self.data if d is not None and d % 2 == 0]

processor = DataProcessor([1, 2, 3, 4, None, 6])
result = processor.process_data()
print("Processed Data:", result)
```

LLM 不仅可以生成代码，还能够协助设计软件架构。这一能力有助于开发人员更富有创意和战略性地与该模型交互。例如，开发人员不再仅请求特定的代码片段，而是可以描述系统的总体目标或功能需求。LLM 可以提出不同的架构设计、建议设计模式或概述整个系统的结构。此种方法不仅节省大量时间，还充分利用 AI 的广泛训练来创新和优化解决方案，并且可能引入人类开发人员最初未考虑到的功效或想法。这种灵活性使 LLM 成为软件开发的创造性和迭代过程中不可或缺的合作伙伴。第 3 章将对此进行探讨。

此外，通过提高从代码到文档等各种交付物的质量和安全性，这些工具能够确保输出达到最高标准。例如，在集成新库时，AI 可以自动生成安全、高效的实现示例，帮助避免常见的安全陷阱。

最后，学习新的编程语言或框架变得更加容易。AI 可以提供实时、语境感知的指导和文档，不仅有助于概念的理解还能促进实际应用。例如，当转向像 Dash 这样的新框架时，AI 助手可以立即生成符合当前项目场景的示例代码片段和详细解释(如代码清单 1.7 所示)。

代码清单 1.7 LLM 生成的演示如何使用程序库的示例代码

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import pandas as pd
import plotly.express as px
```

```
# 样本数据创建
dates = pd.date_range(start='1/1/2020', periods=100)
prices = pd.Series(range(100)) + pd.Series(range(100))/2
# 只是一个简单的序列,用于模拟股票价格
data = pd.DataFrame({'Date': dates, 'Price': prices})
# 初始化 Dash 应用 (通常在主模块中)
app = dash.Dash(__name__)

# 定义应用程序的布局
app.layout = html.Div([
    html.H1("Stock Prices Dashboard"),
    dcc.DatePickerRange(
        id='date-picker-range',
        start_date=data['Date'].min(),
        end_date=data['Date'].max(),
        display_format='MMM D, YYYY',
        start_date_placeholder_text='Start Period',
        end_date_placeholder_text='End Period'
    ),
    dcc.Graph(id='price-graph'),
])

# 回调以根据日期范围选择器的输入更新图表
@app.callback(
    Output('price-graph', 'figure'),
    Input('date-picker-range', 'start_date'),
    Input('date-picker-range', 'end_date')
)
def update_graph(start_date, end_date):
    filtered_data = data[(data['Date'] >=
        start_date) & (data['Date'] <= end_date)]
    figure = px.line(filtered_data, x='Date',
        y='Price', title='Stock Prices Over Time')
    return figure

#运行应用程序
if __name__ == '__main__':
    app.run_server(debug=True)
```

可以在图 1.1 中看到这段代码的输出,这是运行中的 Dash 代码。

LLM 的真正威力在于其在开发环境中的集成。像微软开发的 GitHub Copilot 这样的工具,利用了 LLM 的能力,在集成开发环境(如 Visual Studio Code)中提供实时编程辅助。第 4 章将对此进行介绍。

本书不仅会解释这些概念,还会通过大量示例进行演示,展示如何使用 LLM 显著提高效率和代码质量。读者将学会如何在日常开发中充分利用这些智能工具来设置环境及应对复杂的编程挑战。

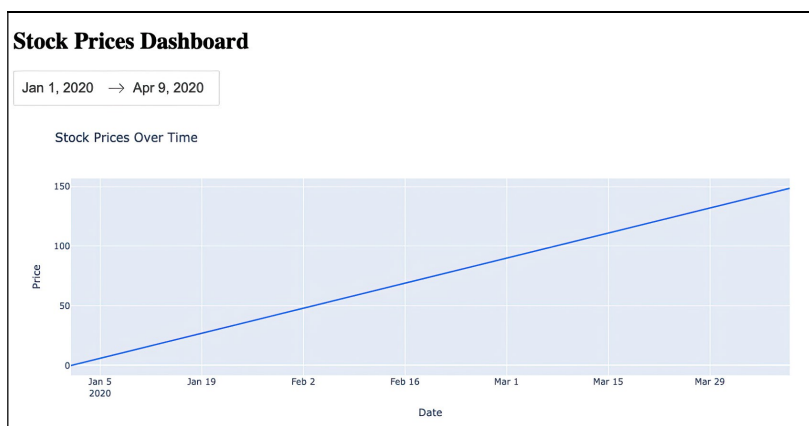


图 1.1 ChatGPT 根据提示“使用 Dash 创建一个示例仪表板”生成的股票价格仪表板

1.2 LLM 介绍

本书主要是实践指南,因此理论部分较少,但以下章节将提供最相关的内容,尽可能帮助“新队友”发挥其作用。

相关拓展

如果有兴趣深入了解 LLM、神经网络以及生成式 AI 背后的理论,可以参考以下两本书: Sebastian Raschka 出版的 *Build a Large Language Model (From Scratch)*(Manning, 2024)和 David Clinton 所著的

The Complete Obsolete Guide to Generative AI(Manning, 2024)。

首先介绍 LLM 的定义及其功能，以便于你将其更好地推荐给老板和同事。LLM 是一种 AI 模型，基于训练数据来处理、理解和生成类似人类的文本。这些模型是深度学习的子集在 NLP 等领域特别受欢迎。

顾名思义，这些模型的“庞大”不仅体现在训练数据的物理大小上，还体现在复杂性和参数数量上。现代 LLM(如 OpenAI 的 GPT-4)拥有高达数千亿个参数。

LLM 通过大量的文本数据进行训练。这种训练涉及阅读和分析大量互联网文本、书籍、文章和其他形式的书面表达，以学习人类语言的结构、细微差别和复杂性。

大多数 LLM 使用 Transformer 架构，这是一种依赖自注意力机制的深度学习模型，可以忽略位置的影响来衡量句子中不同单词的重要性。这使得 LLM 能够生成更具语境¹相关性的文本。一个 Transformer 模型通常由编码器和解码器组成，两者均为多层结构。

了解 LLM 的架构有助于更有效地利用其功能，并在实际应用中解决其局限性。模型不断演化有助于将来为开发人员提供更精细的工具来增强应用程序。

1.3 何时使用或避免使用生成式 AI

生成式 AI(包含 LLM)并不是一劳永逸的解决方案。了解何时使用该技术，以及识别其可能效果不佳甚至存在问题的情况，对于最大限度发挥其优势并减少潜在缺点至关重要。首先将介绍何时适合使用 LLM。

¹ context 在本书有两种翻译方式，在大模型场景中，翻译为“语境”；在软件设计模型中，翻译为“上下文”。

- 提升效率。
 - 示例：使用 AI 自动生成模板代码、生成文档或在 IDE 中提供编程建议。
 - 第 3 章和第 4 章探讨像 GitHub Copilot 这样的工具如何提高编程效率。
- 学习与探索
 - 示例：利用 AI 学习新的编程语言或框架(通过生成示例代码和解释)。
 - 第 5 章研究 AI 如何加速学习过程并介绍新技术。
- 处理重复任务。
 - 示例：使用 AI 处理重复的软件测试或数据录入任务，腾出时间解决更复杂的问题。
 - 第 7 章讨论自动化在测试和维护任务中的应用。

然而，某些情况(主要涉及数据安全和隐私保护)下，应避免使用 LLM 和生成式 AI 工具，如 ChatGPT 和 GitHub Copilot。在敏感或专有数据环境中使用 AI 可能会导致意外的数据泄露。原因之一是部分或全部代码作为语境发送给模型，这意味着至少部分专有代码可能会流出防火墙。还存在一个问题，即这些代码是否会被包含在下一轮训练的数据中。但请放心，第 9 章将探讨几种解决方法。

另一种需要限制使用的情况是：需要精确性和专业性。鉴于 LLM 的一个特点是能够在输出中加入随机性(有时被称为幻觉)，其输出可能会与正确答案存在细微差异。因此，在将输出纳入代码库之前，应始终对其进行验证。

尽管生成式 AI 有许多优势，但必须谨慎使用，同时考虑其使用背景和项目的具体需求。了解何时使用这些强大的工具以及何时应谨慎使用，开发人员可以最大限度发挥其效率，并确保技术高效使用且合乎道德。

1.4 本章小结

- 生成式 AI 既是渐进的，也是革命性的。它是渐进的，因为开发人员每天使用的工具只是其又一个新版软件；它是革命性的，因为它将改变我们的工作方式。
- 未来，开发将涉及如何管理生成式 AI。即使是 10 个传奇的开发人员也无法与拥有 AI 搭档的开发人员相匹敌；借助 AI 的开发人员将以更快的速度、更低的成本产出更高质量的代码。与不使用 AI 编程相比，使用 AI 需要花费更多的时间训练 AI 搭档来按需工作。
- 要信任但也要验证 LLM 的输出。

第 2 章

使用LLM

本章内容：

- 与 ChatGPT 交互
- 了解使用 Copilot 的基础知识
- 了解使用 CodeWhisperer 的基础知识
- 探索提示工程模式
- 对比上述 3 种生成式 AI 的差异

本章将利用 3 种开创性的工具(ChatGPT、GitHub Copilot 和 AWS CodeWhisperer)实践探索生成式 AI 领域。在深入研究该技术的同时还会模拟顶尖科技巨头提出的严格面试问题，将其应用于一系列具有挑战性的场景。无论是经验丰富的开发人员还是充满好奇的技术爱好者，都要准备好解锁创新策略——这可能会为下一次技术面试增加优势；同时准备好将抽象概念转化为切实可行的解决方案，站在 AI 技术招聘发展的最前沿。

下面将首先使用 ChatGPT 的两个现有模型：GPT-4 和 GPT-3.5。这样做有两个目的：一是有助于了解 ChatGPT 的交互模式；二是推动构建基准，以便与其他两个模型进行对比。使用这两个模型还将帮助我们理解模型版本之间的代际变化。最后，本章将使用一些常见的提示工程模式。

2.1 ChatGPT

语境是使用 ChatGPT 时涉及的最重要的方面之一。之前的提示可能会极大地改变当前提示结果。在像 ChatGPT 这样的语言模型中，提示指的是提供给模型以生成响应的输入。提示可以是一句话、一段文字，甚至是一篇较长的文章。它充当对模型发出的指令或查询，引导模型作出响应。考虑到提示的质量以及模型响应的语境，要始终注意在当前会话中发出的提示。因此，建议每次开始新项目时都从新会话开始。附录 A 将指导如何设置账户、登录 ChatGPT 并编写提示。

2.1.1 使用 GPT-4 处理细微差别

本节将致力于解决以下问题：如何在 Python 中反转一个单链表？

什么是单链表

单链表是计算机科学中的一种基本数据结构，由一系列元素组成，每个元素存储在一个节点中。通常，单链表由节点构成，每个节点包含数据和指向下一个节点的引用。

单链表只能单向遍历。常见的单链表操作包括插入(添加新节点)、删除(移除节点)、查找(定位节点)和遍历(依次访问每个节点)。

下面是一个简单的提示。

NC

今后，当我提问时，请尝试提出一个更好的问题。作为一名计算机科学专业的实习生，如何用伪代码定义单链表？

当然，这可能不是一个简单的提示。首先，我们已经指示 ChatGPT 根据其训练数据来改进问题，因此会得到更好的提示。更好的提示会产生更好的输出。你可能会问，怎样才能得到更好的提示？非常好的问题！一般的提示会产生一般的结果。具体的提示会产生具体的结果。与 LLM(包括 ChatGPT)交互时，将从一般到具体逐步细化输出。这在提示工程中被称为细化模式：反复优化或改进提示，以获得更准确、相关或精细的响应。

提示的第二部分介绍了将在本书中反复出现的提示模式：角色模式。在提示工程(主要与 AI 和 LLM 相关)中，角色模式指的是设计提示的一种策略，旨在为 AI 设立一个特定的角色或身份。这种方法可以引导模型作出一致并符合语境的响应。使用角色模式的关键好处之一便是保持响应的一致性。遵循定义好的角色可以使 AI 的响应更具可预测性，并且与用户的期望一致。在本例中，要求 ChatGPT 假设一个具有计算机科学背景的实习生角色。为什么选择实习生？因为需要一个能够完成该工作的角色。

需要重申的是，在与 LLM 交互时，即使应用了角色模式，相同的输入也不一定会产生相同的输出。因此，输出可能不会与以下内容完全一致，但希望它是相似的(注意，代码清单 2.1 是伪代码)。

代码清单 2.1 ChatGPT 对反转链表问题的重新表述

作为计算机科学实习生该如何使用伪代码定义单链表的结构和基本操作？该定义应包括列表的基本组件，如节点和链接，并概述诸如创建节点、向列表中添加节点以及遍历列表等基本操作。

当被问到上述优化过的问题时，ChatGPT 将提供代码清单 2.2。

代码清单 2.2 ChatGPT 对如何反转链表的回答

```
#节点结构
class Node
    data
    nextNode

    Constructor(data)
        this.data = data
        this.nextNode = NULL
#单链表结构
class SinglyLinkedList
    head

    Constructor()
        this.head = NULL

    Function AddToFront(data)
        newNode = new Node(data)
        newNode.nextNode = this.head
        this.head = newNode

    Function Append(data)
        newNode = new Node(data)
        if this.head is NULL
            this.head = newNode
            return
        end if

        lastNode = this.head
        while lastNode.nextNode is not NULL
            lastNode = lastNode.nextNode
        end while

        lastNode.nextNode = newNode

    Function Traverse()
        currentNode = this.head
        while currentNode is not NULL
```

```
print currentNode.data
currentNode = currentNode.nextNode
end while
```

使用 ChatGPT 可以提高生活质量。其出色功能之一是，它通常会提供有关代码的文档。图 2.1 是 ChatGPT 提供的文本描述的截图。

- **Node Class:** Represents each element in the list with a `'data'` field and a `'nextNode'` field pointing to the next node in the list.
- **SinglyLinkedList Class:**
 - `'AddToFront'`: Inserts a new node at the beginning of the list.
 - `'Append'`: Adds a new node at the end of the list.
 - `'Traverse'`: Iterates through the list from the head and prints the data of each node.

The pseudocode outlines the basic structure and operations of a singly linked list, suitable for an intern with a computer science background. It covers node creation, list initialization, and basic methods to manipulate the list.

图 2.1 ChatGPT 模型 4 对单链表伪代码定义的解释

现在让 ChatGPT 扮演一个新的角色——计算机科学实习生。

NC

作为一名计算机科学专业的实习生，你将如何使用编程语言 Python 定义单链表的结构和基本操作？该定义应包括链表的基本组件，例如节点和链接，并概述诸如创建节点、向链表中添加节点以及遍历链表等基本操作。

ChatGPT 将提供代码清单 2.3。

代码清单 2.3 ChatGPT 使用 Python 实现的单链表

```
class ListNode:
    def __init__(self, value):
        self.value = value
        self.next = None

class SinglyLinkedList:
```

```
def __init__(self):
    self.head = None # 列表最初没有任何节点

def append(self, value):
    """ Append a new node at the end of the list """
    if not self.head:
        self.head = ListNode(value)
        return
    current = self.head
    while current.next:
        current = current.next
    current.next = ListNode(value)

def prepend(self, value):
    """ Prepend a new node at the beginning of the list """
    new_node = ListNode(value)
    new_node.next = self.head
    self.head = new_node

def traverse(self):
    """ Traverse the list and print each node's value """
    current = self.head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")
```

ChatGPT 可能会包含一个示例测试框架来运行此代码，如代码清单 2.4 所示。

代码清单 2.4 基于 Python 实现和运行单链表的方法

```
# 创建一个新的链表
linked_list = SinglyLinkedList()

# 将节点添加到列表中
linked_list.append(1)
linked_list.append(2)
```



```
# 在列表前添加一个节点
linked_list.prepend(0)

# 遍历和打印列表
linked_list.traverse() # 输出:0->1->2->None
```

同样，可以将角色细化为参加工程类工作面试的应聘者。角色的转换会向 ChatGPT-3.5 模型发出信号，表明其应解释代码和解决问题的方法。

NC

你是一名计算机科学专业的学生，正在一家软件公司面试工程职位。面试官根据之前对单链表(在 Python 中)的定义，问了你一个问题：如何编写方法来反转遍历方法的输出？如何实现这样的方法？

ChatGPT 将提供代码清单 2.5。

代码清单 2.5 ChatGPT 对如何反转单链表的解释

实现反转单链表遍历方法的输出需要反转链表中的实际链接。这个过程涉及将每个节点的下一个引用改为指向之前的节点。在反转链接后，遍历链表通常将产生顺序颠倒的元素。

```
def reverse(self):
    """ Reverse the linked list """
    prev = None
    current = self.head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    self.head = prev
```

ChatGPT 可能会输出类似于图 2.2 所示的文本，解释解决问题的方法和思路。

Explanation of the Reverse Method:

1. **Initializing Pointers:** Two pointers, `prev` and `current`, are used. `prev` starts as `None`, and `current` starts at the head of the list.
2. **Reversing Links:** In each iteration of the while loop:
 - Temporarily store the next node (`next_node`).
 - Set `current.next` to `prev` to reverse the link.
 - Move `prev` to `current`.
 - Advance `current` to `next_node`.
3. **Updating Head:** Once the loop is finished (when `current` is `None`), `prev` is pointing to the new head of the reversed list. We update the list's head to `prev`.

图 2.2 ChatGPT 模型 4 对反转链表的伪代码定义的解释

此方法实现有效地就地反转了单链表，遍历方法将按与原始插入顺序相反的顺序输出元素。

2.1.2 使用 GPT-3.5 绘制路径

本节将探讨如何利用 ChatGPT-3.5 在 Python 中创建单链表。ChatGPT-3.5 擅长根据接收到的输入生成类似人类的文字，这使其成为编程辅助的重要工具，可以提供一步步的指导、提出改进建议，并对复杂的编程概念进行详细解释。

创建单链表需要使用 ChatGPT-3.5 生成必要的 Python 代码。单链表是一种数据结构，由节点组成，每个节点包含一个值和对序列中下一个节点的引用。这种结构特别适用于动态内存分配以及高效的插入和删除操作。使用 ChatGPT-3.5 可以简化单链表的编程过程，确保其实现既高效又易于理解。代码清单 2.6 的示例演示了如何定义 Node 和 LinkedList 类、初始化列表以及执行插入和遍历等基本操作。

代码清单 2.6 由 ChatGPT-3.5 实现的单链表

```
class Node:
    def __init__(self, data):
```

```
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def prepend(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

    def reverse_and_display(self):
        prev = None
        current = self.head
        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        self.head = prev
```

```
current = self.head
while current:
    print(current.data, end=" -> ")
    current = current.next
print("None")

# 示例用法:
if __name__ == "__main__":
    linked_list = LinkedList()

    linked_list.append(1)
    linked_list.append(2)
    linked_list.append(3)

    linked_list.prepend(0)

    linked_list.display()
    linked_list.reverse_and_display()
```

可以双向应用角色模式：让 LLM 像某个角色中的人或事物那样作出响应，或者让 LLM 假设使用者为某个特定的角色。在用简化方式解释代码或尝试理解复杂主题的情况下，这种方法非常有用。例如，可以要求 ChatGPT 用简化方式解释代码。

NC

假设我已经从大学毕业并获得传播学学位，但我没有计算机科学方面的经验。你该如何向我解释这种方法呢？

代码清单 2.7 展示了 ChatGPT 如何向没有计算机科学背景的人解释该方法。

代码清单 2.7 ChatGPT 向非计算机科学家提供的解释

当然！接下来以一种不涉及专业术语或计算机科学概念的方式来解析 `reverse_and_display` 方法。

想象书架上放着一套书，每本书里都有一个书签指向书架上的下一本书。这套书代

表链表，而书签则代表书目之间的下一个连接。

准备阶段：从第一本书（链表的开头）开始，双手做好准备：一只手用来记下刚读过的书（prev），另一只手用来拿当前正在读的书（current）。还有一个空的临时存放处（如一张桌子），帮助你暂时记住下一本要读的书。

改变顺序：现在来改变书的顺序。每一本书（节点）按以下步骤操作：

- a. 使用书签（下一个引用）找到下一本要读的书，并将其放在临时存放处（桌子上）。这就像记下下一本要读的书的位置。
- b. 把当前书中的书签翻转过来，使其指向刚读过的书。这意味着当前这本书现在连接的是刚读过的书，而不是下一本要读的书。这就像翻转书签。
- c. 交换手中的书：把刚读过的书（prev）放在左手（为下一次迭代做准备），把当前正在读的书（current）放在右手（以便处理下一本要读的书）。
- d. 走到放下一本书的地方（桌子上那本书），继续这个过程。

重新开始：不断重复上述步骤，直到没有书可读。此时，右手拿着最后一本书（也就是刚开始的第一本），而左手是空的。现在，把手中拿着的第一本书（原顺序中的最后一本书）放回书架上重新作为第一本书。

倒序阅读：最后，从新的第一本书开始，按照翻转后的书签顺序阅读。每读一本书，大声说出其书名。这样就能知道书名的倒序列表。

因此，`reverse_and_display` 方法实际上改变了书架上书（节点）的顺序，然后按反转后的顺序读取书名（数据）。

反向使用角色模式在提示工程的背景下通常被称为受众角色模式。这指的是为特定应用程序或用例的目标受众预先定义的画像或描述。其有助于调整由 LLM 作出的响应，以便更好地满足特定用户群体的需求和期望。在讨论 GitHub Copilot 之前，先比较一下每个模型的输出质量。

2.1.3 在 AI 海洋中航行：从 GPT-3.5 “海岸” 到 GPT-4 “天际线”

这两个模型输出之间的显著差异在于其输出的复杂性和透明度，特别是在模型与数据结构交互并对其进行修改方面。GPT-3.5 和 GPT-4 所采用方法的不同，凸显了 AI 生成的代码向更清晰和更

可预测性广泛转变。AI 模型愈发先进，其输出便愈发体现优质编程实践的细微差别，这反映了人类程序员技能和敏感性的演变。这一演变对于 AI 成为可靠的软件开发合作伙伴至关重要，在软件开发中，清晰和精确不仅是理想条件，更是必要条件。

使用 GPT-3.5 时，`reverse_and_display` 方法在执行过程中有些不透明。该版本模型改变了链表的底层数据结构，有效反转了节点。然而，它并没有明确地向用户传达这一变化。从开发人员的角度看，这可能会产生意外的副作用。例如，如果调用 `reverse_and_display` 并假设只显示反转后的列表，会发现原始列表结构已被永久改变。这种操作缺乏透明度，容易导致混淆和错误，特别是在复杂的应用程序中，原始数据结构的完整性至关重要。

相比之下，GPT-4 的 `reverse` 方法更精细。该方法明确地反转了链表，任何有经验的程序员都可以从其名称和结构推理出它会修改底层数据结构。GPT-4 的方法更符合代码整洁可维护的原则。该方法体现出每个函数或方法应执行明确定义的任务这一理念。这里体现了关注点分离：链表的反转和显示被视为独立的操作。这增强了代码可读性，并降低了出现意外副作用的可能性，因为开发者完全清楚对数据结构所作的变化。

2.2 Copilot

下面使用 GitHub Copilot 来解决同样的问题。附录 B 将介绍如何创建账户并将插件安装到最喜欢的 IDE(假设为 VS Code 或 PyCharm)中。完成安装后，应在 IDE 中创建一个新项目。首先，创建一个名为 `main.py` 的新文件。在这个文件的开头，输入以下注释/提示：

```
# 你是一名计算机科学专业的学生，正在一家软件公司面试工程师职位。面试官问了你以下问题：根据之前定义的单链表(用 Python)，如何编写方法来反转遍历方法的输出？如何实现这样的方法？
```

注意开头的井号字符表示 Python 注释。特别有趣的是，在 ChatGPT 和 GitHub Copilot 中使用了相同的提示。

接下来，注意当开始输入 `ListNode` 类的定义时，Copilot 会提供代码建议。这是内联模式的交互方式。或者，可以按 `Ctrl+Enter` 键激活交互模式，让 Copilot 根据当前语境生成多达 10 个建议。

注意：整本书主要使用内联模式，可以按下 `Tab` 键接受给定建议。

得到的代码可能类似于 ChatGPT 创建的源代码。下面尝试一个稍微不同的编程挑战，不重复上一节中的相同代码。要求 Copilot 执行以下任务：

```
# 给定一个未排序的整数数组，编写一个函数来找出数组中的第 k 个最小元素。  
可以讨论一下解决方案中的时间复杂度和空间复杂度吗？
```

在几个换行之后，Copilot 会添加注释，该注释讨论这段代码的时间复杂度和空间复杂度以及整个问题(见代码清单 2.8)。

代码清单 2.8 ChatGPT 计算并报告此代码的复杂度

```
def kth_smallest_element(arr, k):  
    # 时间复杂度: O(N)  
    # 空间复杂度: O(1)  
    if k > len(arr):  
        return None  
    return sorted(arr)[k - 1]
```

Copilot 的建议虽然值得赞赏且完全合理，但在面试中可能不会被接受。这感觉像是在作弊。我们应预料到会有后续问题，要求解释如何在不完全排序列表的情况下实现这个功能。下面改进一下提示，确保在取第 `k` 个元素之前不对列表进行排序。

```
# 作为一名计算机专业毕业生，应聘软件工程师职位，想象在面试中遇到这样一个问题：如何在 Python 中实现一个函数，在未排序的整数数组中找到第 k 小的元素，而不对整个数组进行排序或使用任何内置排序函数？此外，能解释一下算法的时间复杂度和空间复杂度吗？这个问题测试了设计高效算法的能力以及对复
```

杂度分析的理解，同时要遵循给定的约束条件。

Copilot 提供的代码看起来非常类似于二分查找(见代码清单 2.9)，这无疑是一个有趣的选择。

代码清单 2.9 Copilot 解决第 k 个元素问题的方法

以下代码演示了在 Python 中实现查找第 k 小元素函数的过程，并附有其时间复杂度和空间复杂度的解释。

```
# 用于确定无序数组中第 k 小元素的函数。
# 检查输入数组是否为空或 k 是否越界
if not arr or k < 1 or k > len(arr):
    return None

# 初始化变量以存储数组的下界和上界
low = min(arr)
high = max(arr)

# 执行二分查找以找到第 k 小的元素
while low <= high:
    # 计算数组的中间元素
    mid = low + (high - low) // 2

    # 统计小于或等于中间元素的元素数量
    count = sum(1 for num in arr if num <= mid)

    # 根据数量调整搜索范围
    if count < k:
        low = mid + 1
    else:
        high = mid - 1

# 返回第 k 小的元素
return low

# 时间复杂度:  $O(n \log(\max(arr) - \min(arr)))$ 
# 空间复杂度:  $O(1)$ 
```


2.3 CodeWhisperer

开始使用 CodeWhisperer 之前,应该先禁用 Copilot。单击 Extensions 选项卡,搜索 Copilot。找到后,单击 Disable 按钮。然后需要重启应用程序。应用程序重新启动后,便可以开始使用 CodeWhisperer。如果需要了解如何安装或配置 CodeWhisperer,请参阅附录 C。完成插件安装并登录开发人员账户后,创建一个名为 `asset.py` 的文件。图 2.3 显示如何查找和禁用此插件。

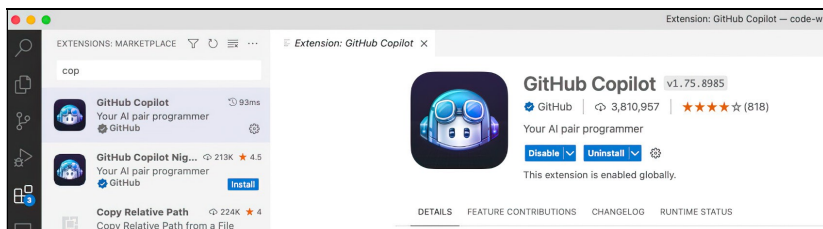


图 2.3 在切换到 CodeWhisperer 之前,必须先禁用 Copilot。在 Extensions 选项卡中关闭该扩展程序

我们将使用与 Copilot 中相同的提示。为方便起见,这里重新列出。

#作为一名计算机专业毕业生,应聘软件工程师职位,想象在面试中遇到了一个问题:如何在 Python 中实现一个函数来确定一个未排序整数数组中的第 k 小元素,而不对整个数组进行排序或使用任何内置排序函数?此外,能解释一下算法时间复杂度和空间复杂度吗?这个问题测试了设计高效算法的能力以及对复杂性分析的理解,同时要求遵循指定的约束条件。

CodeWhisperer 提供的代码实现了 Quickselect 算法,该算法与 QuickSort 排序算法密切相关(见代码清单 2.10)。Quickselect 特别适用于高效查找未排序数组中的第 k 小元素。

代码清单 2.10 CodeWhisperer 解决第 k 个元素问题的方法

```
def partition(arr, low, high):
```

```
pivot = arr[high]
i = low - 1
for j in range(low, high):
    if arr[j] <= pivot:
        i += 1
        arr[i], arr[j] = arr[j], arr[i]
arr[i + 1], arr[high] = arr[high], arr[i + 1]
return i + 1

def quickselect(arr, k, low=0, high=None):
    if high is None:
        high = len(arr) - 1
    if low < high:
        pivot = partition(arr, low, high)
        if pivot == k:
            return arr[k]
        elif pivot > k:
            return quickselect(arr, k, low, pivot - 1)
        else:
            return quickselect(arr, k, pivot + 1, high)
    return arr[low]

# 使用示例
arr = [5, 2, 4, 6, 1, 3]
k = 2
print(quickselect(arr, k))
# 输出: 2
```

Copilot 和 CodeWhisperer 生成的代码之间有一个有趣的差异。CodeWhisperer 将 k 解释为排序数组中元素的索引。因为大多数编程语言中的数组索引从 0 开始，所以如果 k 是 2，CodeWhisperer 将找到第 3 小的元素(因为索引 0、1 和 2 分别对应第 1、第 2 和第 3 小的元素)。另一方面，Copilot 假定 k 表示元素的排名，而不是索引。因此，如果 k 是 2，Copilot 将返回数组中第 2 小的元素。这就好比说是“第二名”而不是“索引 2”。

本节引入了 AWS CodeWhisperer。与其“前辈们”一样，

CodeWhisperer 成功生成了解决问题的代码，进一步证明了 AI 在软件开发中的变革潜力。

鉴于这些工具生成的代码惊人相似，有趣的问题自然浮现：这些产品究竟有何不同？考虑到每个工具的独特优势和局限性，答案并不是想象的那么简单。

下一节将深入探讨这个问题，比较 ChatGPT、Copilot 和 AWS CodeWhisperer 这 3 种工具，以便了解其独特功能、最佳用例以及如何重塑软件开发的未来。最终目标是提供全方位指南，帮助软件开发人员顺应 AI 驱动工具领域的快速发展。

2.4 比较 ChatGPT、Copilot 和 CodeWhisperer

首先考虑交互模式：如何与 AI 交互。对于 ChatGPT，方式为登录聊天网站并在聊天框中输入提示。然后在后续的提示中细化需求。反馈循环会从前提示的语境中提取信息，并将其应用于当前提示，生成用户可以回应并重新触发的输出。如果将这种交互模式与 Copilot 和 CodeWhisperer 进行对比，就会发现后两者是在 IDE 中工作的。无论怎么尝试，都无法在 IDE 之外使用。这种方法并不一定逊色，只是有所不同。

Copilot 和 CodeWhisperer 保留在 IDE 中的方式可以被视为一种优势而非缺陷。后面的章节将介绍 Copilot Chat，其结合了 ChatGPT 和 GPT-4 的优点，且这三者均保留在 IDE 中。这些工具有助于长时间专注代码而不受干扰。无干扰工作是提高效率的关键。Copilot 和 CodeWhisperer 能有效避免中断工作流程，防止切换语境，有利于免受干扰，并能保持更长时间的高效状态。通过对话与 ChatGPT 交互的同时，Copilot 和 CodeWhisperer 能够提供建议。对话需要时间，而建议则迅速且免费。

接下来将研究代码的呈现和生成方式。ChatGPT 可以将代码创建为代码块、方法、类或项目。如果被问及，ChatGPT 会谨慎地呈

现项目。但实际上，它会在后台创建项目。毕竟，ChatGPT 喜欢对话。使用 Copilot 和 CodeWhisperer 时，代码最初是一次呈现一个方法。随着使用愈发频繁，就会发现这些工具能够为给定的类编写越来越多的代码。但遗憾的是，它们无法通过简短的提示写出整个项目。

它们都具备响应提示的能力。对于 ChatGPT，提示是与工具交互的唯一方式。对于 Copilot 和 CodeWhisperer，响应提示并不是必需的，但编写这些提示会使得输出更贴近最初的设想。

结合这些因素可以得出结论：ChatGPT 是探索和进行原型设计的绝佳选择。然而，ChatGPT 也可能带来不必要的干扰，部分原因是离开了 IDE，转而在浏览器中操作，而浏览器本身带有各种诱惑。ChatGPT 本身就会造成不必要的干扰。这个工具极有可能会让人陷入所谓的“兔子洞”。但不要因此被吓倒，它仍然是非常宝贵的资源。

Copilot 和 CodeWhisperer 要求使用时有明确的目标。因此，在有精确要求和紧迫期限的情况下进行编程，这些工具是完美的选择。在熟悉编程语言和框架的情况下使用 Copilot 和 CodeWhisperer，其表现最佳。它们可以自动化处理许多烦琐的任务，这有助于你专注业务需求，这些需求增加了价值，也可能是你编写软件的初衷。图 2.4 简要总结了这 3 种生成式 AI 的优势和局限性。

本章做了很多工作，实现了基本的数据结构并解决了一些经典的计算机科学问题。本章的工作是基础性的，有助于更好地判断何时使用 ChatGPT，以及何时使用其他聚焦 IDE 的工具(如 Copilot 和 CodeWhisperer)。后续章节将利用这些知识选择最合适的工具。

最后需要注意的一点是：这些工具协同工作时效果最佳。例如，ChatGPT 是一个提供示例和结构的绝佳工具；而 Copilot 和 CodeWhisperer 则允许扩展和自定义代码。

	ChatGPT	Copilot	CodeWhisperer		
基于提示	◇	△	△	◇	专门
基于IDE	○	◇	◇	△	支持
生成方法	△	△	△	○	不支持
生成类	△	△	△		
生成项目	△	○	○		
生成文档	△	△	△		
切换语言	△	○	○		
切换库	△	△	△		

图 2.4 ChatGPT、Copilot 和 CodeWhisperer 的优缺点对比

2.5 本章小结

- ChatGPT 是基于提示的生成式 AI，通过与用户对话帮助其探索想法，以辅助设计和开发整个项目。此外，ChatGPT 还能巧妙地为每个编写的方法生成文档。在本章开头使用 ChatGPT 的原因之一是，其有助于定义本章剩余部分使用的模板。这是一个有趣的产品，虽然令人愉悦，但有时会导致不必要的分心。
- Copilot 和 CodeWhisperer 是聚焦具体任务的工具，知道想做什么并且需要建议来完成任务的情况下最为有效。与这些工具交互的方式非常相似，产生的结果也大同小异。

- 截至目前，ChatGPT 不支持在 IDE 中进行开发。然而，与 GitHub Copilot 和 AWS CodeWhisperer 不同的是，它可以生成整个项目，并能轻松地将代码从一种编程语言翻译成另一种。Copilot 和 CodeWhisperer 会根据注释推断想要编写的代码，而使用 ChatGPT 时需要明确地编写提示，这样它才会根据这些提示创建代码。
- 角色模式的目的是设计提示，为 AI 建立特定角色，从而引导模型以一致且符合语境的方式作出响应。遵循定义好的角色，AI 的响应会更加可预测，并符合用户的预期。
- 实习生通常热衷于学习，在相应领域的知识水平为初级到中级，并愿意承担各种任务来获取经验和技能。实习生可能会提出明确的问题，寻求指导，并展示积极主动解决问题的态度。他们通常具备较强的资源利用能力，但在相应领域的深度专业知识可能不如更有经验的专业人士。这种角色在模拟学习和成长型思维的场景下非常有用。
- 细化模式是指逐步优化或改进提示，以便获得更准确、相关或精细的响应。这一过程从一般到具体，并随着与 LLM(如 ChatGPT)的交互而发展，逐步提升输出质量。
- 受众角色模式是提示工程中角色模式的一种变体。它是指为特定应用程序或用例的目标受众定义的画像或描述，这有助于根据特定用户群体的需求和预期来调整 LLM 作出的响应。