前言

早在 1980 年代初期，我就已經開始在寫程式碼了。我的第一台設備是 Atari 400——那並不是一台擁有薄膜鍵盤的強大設備，實際上只有 8K 的 RAM，而且程式還必須從磁帶載入。當時我學的是 BASIC 語言，而且我還用它來建立了一些簡單的遊戲和實用的應用程式。

當時的我，簡直是完全融入其中。

當然，隨著時間的推移，後來我也升級到了功能更強大的機器。在這個過程中，我陸續接觸了 Pascal、C 和 C++ 等程式語言。但是 IDE（程式碼整合開發環境）除了特別針對語法強化顯示與除錯的功能之外，一直都沒有太大的變化。

近來由於 GitHub Copilot 和 ChatGPT 的出現，一切也隨之出現巨大的變化。在嘗試這些工具的過程中，我感覺就像是第一次拿到 iPhone 一樣——這絕對是足以改變遊戲規則的重大轉變。

我可以直接使用自然語言，要求 ChatGPT 寫出一些程式碼。我也可以在 VS Code 輸入函式的一些片段，然後 GitHub Copilot 就會自動幫我生成一整塊的程式碼。很多時候，經常會有一種正中下懷的感覺。哦對了，我還可以用 ChatGPT 把圖片轉換成程式碼。

然而，真正厲害的是，這些工具可以為開發者處理掉許多繁瑣的任務。有誰真的會喜歡去研究正規表達式的語句，或是去拼湊 bash 指令、GitHub Actions ？當然不會是我囉。不過，對於那些 AI 程式工具來說，那些任務根本就像一頓早餐，兩下子就吃光光了。

事實證明，這些 AI 工具不光只是在寫程式碼時很好用。像我自己就已經開始用 ChatGPT 來針對各種 App 的構想進行腦力激盪、草擬出各種軟體需求，甚至用來編寫單元測試了。

實際上並沒有過多久，我就被徹底說服，相信 AI 輔助程式設計一定會成為程式設計師必備的技能之一。

所以，沒錯，我確實看到了撰寫這本書的巨大需求。因此，我很快就整理出一個大綱，然後把它提交給 O'Reilly 公司。他們立刻就看出了其中的潛力。

撰寫這本書的過程很有趣，我自己也學到了很多東西。這段期間我也面試了許多很聰明的開發者，他們也為我提供了許多非常好的想法和技巧。

不過，AI 輔助程式設計一直都在迅速發展中。這也就是本書搭配了一個 GitHub 程式碼儲存庫（*https://github.com/ttaulli/AI-Assisted-Programming-Book*）的理由。我會在那裡針對本書進行各種更新，持續把這個令人興奮的領域裡其他重要的進展囊括進來。

所以，我很感謝你拿起這本書。我希望你會發現，它不但可以提供豐富的訊息，而且還能成為你旅程中非常寶貴的一份指南。

涵蓋的內容

下面簡單介紹每一章的內容：

- 第 1 章「開發者的新世界」：本章一開始先來探討生成式 AI 如何改變程式設計師的遊戲規則。這裡探討的主要是 AI 工具如何協助開發者，多多去思考整體的藍圖，而不要太糾結於程式碼的細節。本章也簡單回顧了程式語言的歷史。另外還談了關於 GPT-4 之類先進 AI 技術的詳細資訊。
- 第 2 章「用 AI 寫程式的技術原理」：本章一開始會先解釋一下生成式 AI，並說明 Transformer 模型和 LLM 大語言模型，為什麼在程式設計世界裡如此重要。最重要的是，本章還有一段 OpenAI Playground 遊樂場的操作演練，可以向你展示如何運用與調整那些 AI 模型，以滿足你的程式設計需求。
- 第 3 章「提示工程」：如果想善用 AI 輔助程式設計工具，本章的資訊特別重要。本章充滿了各種實用的技巧，例如如何處理一些冗長或令人感到困惑的提示，還有如何阻止 AI 胡扯瞎說的做法。此外，本章還會把提示分解成好幾個關鍵的部分，然後再向你展示如何有效善用各部分的技巧。
- 第 4 章「GitHub Copilot」：本章會逐步引導你去善用這個強大的工具。我們首先會介紹一些核心功能，例如用註解、聊天的方式，或是用 AI 所驅動的指令行介面，來建立所需要的程式碼。本章也會介紹如何針對專有的程式碼庫，對系統進行客製化調整。

- 第 5 章「其他 AI 輔助程式設計工具」：本章會詳細介紹其他幾個頂尖的 AI 輔助程式設計工具，例如 Amazon CodeWhisperer、Google 的 Duet AI，以及 Replit 等等。
- 第 6 章，「ChatGPT 和其他通用 LLM」：本章會介紹如何運用這些通用的工具，來執行一些像是處理正規表達式、提供新手入門程式碼，以及 GitHub Actions 之類的任務。
- 第 7 章「構想、規劃、開需求」：本章的重點，就是運用聊天機器人來啟動軟體專案。本章所牽涉到的主題，包括腦力激盪、市場調查、需求文件以及測試驅動開發等等。
- 第 8 章「寫程式」：本章會介紹一些蠻常見的開發情境，包括 API 的使用、模組化的程式設計方式，以及重構的做法。另外我們也會介紹如何處理函式與物件導向程式設計的做法。
- 第 9 章「除錯、測試與部署」：本章討論的是在開發過程中比較不那麼光鮮亮麗的部分。本章所涵蓋的主題，包括 bug 的修正、運用 AI 輔助程式設計工具來進行程式碼審查、進行單元測試，以及 PR 拉取請求的相應說明。
- 第 10 章「重點摘要」：本章就是本書的總結，我們會特別強調一些核心的重點。

本書有什麼不同之處

軟體開發之所以蓬勃發展，主要是因為它具有確定性（certainty）。你只要把特定的輸入送進程式，一定會得到相同的輸出結果。多年來，這種純粹而具有確定性的邏輯，正是軟體的核心與靈魂。

但如果你使用了 AI 輔助程式設計工具，事情就變得沒那麼確定了。所取得的結果就像擲骰子一樣，因為一切都是由機率來決定的。如果你用提示叫 AI 工具幫你編寫出一些程式碼，就算你每一次都採用相同的提示，還是有可能得到不同的結果。當然，一開始這樣確實有點令人頭疼，但你只要掌握了竅門，就會覺得非常值得。這其實就是我們用一整章來探討提示工程的理由，因為提示工程對於這種寫程式的新做法確實非常有幫助。

第三章

提示工程

提示工程（*Prompt engineering*）屬於機器學習（ML）和自然語言處理（*NLP*）的一個子領域，研究的是如何讓電腦能夠理解與解釋人類的語言。其主要目標就是搞清楚如何與 LLM 大語言模型這種專門設計用來處理和生成類似人類語言回應的複雜 AI 系統，以正確的方式進行對話，讓它能夠生成我們想要尋找的答案。

你可以這樣想：當你向某人尋求建議時，總要先提供一些背景資訊，並清楚說出你的需求。當你面對的是 LLM 時，同樣也是如此。你必須仔細說出你的問題或提示。有時候，你甚至可能會在你的問題裡添加一些暗示或額外的資訊，以確保 LLM 能夠理解你的要求。

而且你所要問的問題，也不一定一次就能回答清楚。有時候，你就像在與 LLM 進行一連串完整的對話，你們會一來一往，逐步調整你的問題，直到你得到真正需要的寶貴資訊為止。

舉例來說，假設你正在使用 AI 輔助程式設計工具來開發 Web 應用程式。一開始你可以詢問它如何使用 JavaScript 建立一個簡單的使用者登入系統。它最初的回應可能會涵蓋一些基礎的知識，但隨後你就意識到，自己需要的是更進階的功能。因此，接著你就會給出更具體的提示，詢問如何納入密碼加密機制，以及如何安全連接到資料庫的做法。你與 AI 每一次的互動，都可以讓它的回應不斷精益求精，逐漸形塑出可以符合你專案特定需求的結果。

挑戰

提示工程有可能會讓人感到很沮喪。你的提示就算只是在措詞上有微小的變化，還是可能對 LLM 的輸出產生巨大的影響。這是因為模型背後的先進技術，其實是以機率的框架為基礎。

下面就是提示工程所面臨的一些挑戰：

囉嗦

LLM 有可能會喋喋不休。你只不過給了它一個提示，它卻有可能直接展開，給你一段非常冗長的回應，而你真正想要的很可能只是一個快速簡單的回答。它會傾向於拋出一大堆相關的想法或事實，讓回應遠超過必要的長度。如果你希望 LLM 能夠開門見山，直接切入正題，你就要記得要求它「concise」（精簡）一點。

不可轉移性

意思就是某個提示對某個 LLM 很適用，對另一個 LLM 卻有可能不太合用。換句話說，如果你從 ChatGPT 切換到 Gemini 或 GitHub Copilot，由於每一個 LLM 的訓練、設計和專業取向都不一樣，因此你或許就要針對不同的模型，分別去調整提示的內容。不同的模型都是分別用不同的資料集和演算法來進行訓練，因此對於同樣的提示自然會產生不同的理解和解釋。

長度的敏感性

LLM 可能會被冗長的提示所淹沒，然後就會開始忽略或誤解你所輸入的部分內容。這就像 LLM 的注意力分散了，因此它的回應也會變得有點心不在焉。這就是為什麼你應該避免在提示裡提供太過詳細的需求；請把提示保持在一頁的範圍以內。

歧義

如果你的提示不夠清楚，LLM 可能就會感到很困惑，然後給出完全偏離事實或純粹虛構的回應。保持清楚而明確，就是其中的關鍵。

雖然有這麼多挑戰，但還是有一些方法，可以改善提示的效果。我們會在本章其餘的部分，介紹這些相應的做法。

提示的組成元素

你可以把提示想像成具有四個主要的組件，如圖 3-1 所示。

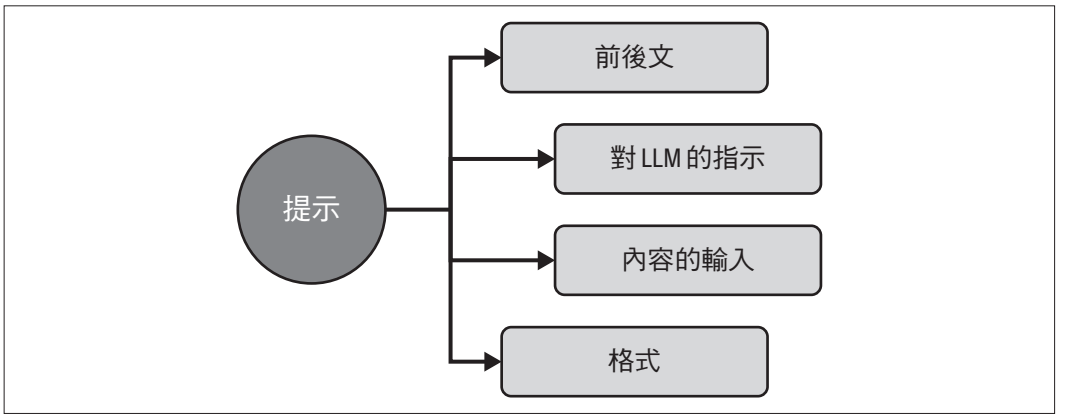

圖 3-1　提示有四個主要的組件

首先，前後文（*context*）指定的是 LLM 在提供回應時所扮演的人格或角色。再來是一些指示（*instruction*），例如要求 LLM 進行總結、進行翻譯，或是進行分類等等。然後則是內容的輸入（*input of content*），也就是你想讓 LLM 幫你處理的資訊，希望它可以幫你創建出更好的回應。最後，你可以再向它展示一下你想要的輸出格式（*format*）。

請注意，你的提示並不一定需要包含所有這些組件。實際上，你也許只需要其中一個組件，就能得到很好的回應。不過一般來說，最好還是盡量提供 LLM 一些更具體的細節。

我們就來逐一看看每個組件吧。

前後文（Context）

你的提示一開始通常會用一兩個句子來提供前後文。通常你可以指定 AI 去扮演某種人格或角色，讓它用這樣的方式來提供回應。這樣不但可以讓回應更準確，而且可以更貼近前後文的背景脈絡，得出更有意義的結果。

減少幻覺

我們在第 2 章瞭解到，給了 LLM 提示之後，所得到的回應有可能會出現幻覺，也就是所生成的內容有誤或是具有誤導性，但 LLM 回應的表達方式，卻又好像很有信心、千真萬確的感覺。幻覺對於軟體開發來說尤其麻煩，因為軟體開發很需要讓所有的東西保持正確。

毫無疑問，只要套用本章所學到的一些技巧，就可以稍稍緩解這個問題，但即使是精心設計的提示，還是有可能出現幻覺。造成這種問題的原因，有很多不同的理由：

缺乏真實可信的驗證

LLM 是根據訓練資料所學習到的模式來生成回應，但它無法驗證這些資訊的正確性或真實性。

過度套入和記憶的影響

LLM 可能會記住一些存在於訓練資料集裡、其實並不正確或是具有誤導性的資訊，特別是這些資料如果經常重複或很常出現的話。

訓練資料的偏差

如果訓練資料包含了一些有所偏差、不太準確或虛假的訊息，模型可能就會在它的輸出裡重複丟出這樣的內容。

推斷和猜測

有時候，LLM 可能會根據它在資料裡所看出來的模式來進行推斷，生出一些在訓練資料裡並未充分涵蓋的主題或問題相關的資訊。

缺乏前後文背景或是錯誤解讀

LLM 可能會因為缺乏必要的前後文背景，發生錯誤解讀的情況，因而無法準確回應某些提示。模型有可能並不完全理解某些查詢背後細微的差別或隱含的意義。

俚語和慣用語

這樣的語言可能會產生歧義，導致模型誤解原本預期的意思，尤其是訓練期間並沒有在前後文裡看過足夠多的俚語或慣用語範例的情況下。

存庫裡本來就很缺乏這方面的訓練資料。那些通用模型對於你公司裡專用的程式語言，當然也是一無所知。

GitHub 系統也會不斷去掃描公司裡的程式碼儲存庫，來強化企業版的模型。舉例來說，它會特別注意最近的 PR 拉取請求（pull request）、合併（merge）以及各種給讚（thumbs-up）和倒讚（thumbs-down）的回饋意見。所有的這些全都有助於特別突顯出公司當下正在使用的最新方法和策略。

這種可量身定做的模型，可以協助企業在整個組織裡傳播一些專業的知識。AI 會慢慢提取出一些隱藏在程式碼裡的微妙知識，然後把它分享出來。在持續的訓練之下，AI 就可以跟上不斷持續變化的程式碼庫，而隨著時間的推移，它所提供的協助就會變得越來越精確。不過，一般公司組織在把這些 AI 工具整合到開發的過程中時，隱私的處理和智慧財產權的風險確實還是非常重要。

根據 Gartner 的研究顯示（*https://oreil.ly/zXAQ-*），2023 年時，大約只有不到 10% 的大型企業開始使用 AI 輔助程式設計工具。會有這樣的猶豫，部分的原因是出於安全性和精確性的擔憂。不過，隨著技術的快速進步，預料在不久的將來，應該會有越來越多的企業開始採用這些工具。簡單來說，這些工具所提供的優點太明顯了，實在不容忽視。

使用案例：硬體設計

有個蠻有趣的研究案例，就是 AMD（Advanced Micro Devices）的客製化模型。這家公司成立於 1969 年，是 CPU（中央處理單元）領域的先驅（*https://oreil.ly/pOoMj*）。如今這家公司則是資料中心、嵌入式系統、遊戲平台和個人電腦半導體的領導者。

在深入討論之前，我們先來回顧一下硬體系統開發的相關基礎知識。這個領域的做法，與建立網頁 App 之類的軟體領域完全不同。其中最關鍵的挑戰就是開發者一定要徹頭徹尾完全瞭解整個硬體系統。與一般運行在通用電腦上的軟體不同的是，韌體會直接與硬體進行對話。這需要的是一種更加嚴格的精確性和相容性。

這種程度的精確性可說是至關重要，因為韌體的開發過程中如果出現了錯誤，可能就會導致一些非常嚴重而昂貴的後果。只要一個小小的錯誤，可能就意味著好幾百萬美元的經濟損失。而且這不只是錢的問題，時間也是個很重要的因素，因為如果要修正韌體的問題，通常都需要重新審視整個製造程序，這有可能會多花好幾個月的時間。這樣的延遲不但會影響產品發表的時程，也會影響產品在市場上的競爭力。

很顯然，敏捷軟體開發很常見的「快速行動，打破常規」思維方式，在這種情況下並不適用。這種做法的風險實在太大了。這就是為什麼韌體開發者必須投入大量的時間和精力，制定出詳細的計劃並進行廣泛的測試。這種謹慎的做法，可以確保韌體在與硬體搭配起來之前，盡可能做到可靠且沒有錯誤的程度。

AMD 在 2023 年針對 Copilot 進行考察時，設定了非常高的標準，而且對它抱持相當多的懷疑，這完全是可以理解的。在一個試驗性的專案中，AMD 針對各種硬體描述語言（*HDL*；例如 Verilog 和 SystemVerilog）建立了一個客製化版本的 Copilot。HDL 屬於一種特定類型的程式語言，專門用來表述電子電路（尤其是數位邏輯電路）的架構、設計和功能。它可以在不同的抽象層級，為電子系統建立模型、進行模擬。

這個試驗專案的結果（*https://oreil.ly/TXMvQ*），比預期好得多了。令人驚訝的是，Copilot 所生成的程式碼風格，實際上比他們自己的程式設計師所生成的程式碼更符合 AMD 的標準。這方面的改進實在非常顯著，以至於有一些程式設計師甚至為了這個理由，特別從原本慣用的、高度可自訂的文字編輯器 Vim，轉而採用 Visual Studio Code 來作為他們的 IDE。

使用案例：Shopify

另一個有趣的研究案例則是 Shopify。這家公司所經營的是一個平台，可以讓客戶自行建立電子商務網站。Shopify 在美國的市場佔有率大約是 10%，在歐洲則有 6%。

毫無疑問，這家公司對於基礎設施有非常大的需求。你可以想像一下，大約有 300 個公開程式碼儲存庫，和 5,000 個左右的私人程式碼儲存庫。此外，Shopify 每天都要進行大約 1,500 次的程式碼部署工作。

Shopify 是最早搭上 Copilot 這股浪潮的公司之一；Copilot 確實提高了開發者的工作效率，使他們成為了一家真正改變遊戲規則的公司。目前大約有將近 2,000 名 Shopify 開發者在使用這個工具（*https://oreil.ly/ZI_9K*）。最酷的部分就是：有 70% 的人認為它確實很有幫助，而且有 75% 的人經常在使用它。Copilot 的程式碼建議，大約有 26% 確實被採用。

當然，還是有某些功能還沒有真正流行起來，例如指令行介面（CLI）的整合。但儘管如此，許多開發者每天都還是會去使用程式碼補全和聊天的功能。

下面是一些比較有趣的要點：

程式碼建議的價值

就算開發者並沒有採用所給出的建議，也不表示那些建議完全沒有用處。任何建議都有可能激發出靈感，讓你編寫出更好的程式碼。

採用率

使用量通常會隨時間而增加，這點並不令人意外。調整日常的工作流程，適應新的功能，總需要一點時間。Copilot 的使用，也有一定的學習曲線。

資深開發者的採納

在早期剛導入 Copilot 時，一些經驗比較豐富的開發者並不太熱衷於使用它。他們會比較傾向於把它視為一種玩具，而不是真正可以認真對待的工具。不過隨著時間的推移，這些人開始注意到其他開發者確實取得了一些實際的成果，於是他們也開始越來越有興趣了。

學習強化的效果

Shopify 注意到，Copilot 很擅長推動人們去嘗試新的程式語言或框架。舉例來說，Rust 的採用率出現了顯著的上升。

Shopify 的程式碼庫其中大約有 100 萬行是運用這個工具寫出來的，這就表示對於這家公司來說，Copilot 確實非常重要。

使用案例：Accenture

Accenture 是一家大型專業服務組織，可透過一些創新技術和系統，協助客戶改善營運和成長。這家公司在全球 120 多個國家擁有超過 733,000 名員工（*https://investor.accenture.com*）。

2023 年，Accenture 有 450 名內部開發者（*https://oreil.ly/ku8Nd*）對 Copilot 進行了測試。這家公司並沒有設定任何具體的任務或目標。公司的管理者只要求大家按照往常一樣繼續自己的工作就行了。

Accenture 針對 Copilot 的試驗總共持續了六個月。結果怎麼樣呢？在編寫程式碼方面，Copilot 的建議被採用的接受率為 35%，而且就算經過程式碼審查的階段，其中仍有 88% 的改動確實被保留下來。生產力也明顯提高了。PR 拉取請求增加了 50%，合併率也增加了 15%。在效率上也有很大的進步，build 建置的次數增加了 50%，成功率提高了 45%。而開發者的感覺呢？大家對此都感到非常滿意：有高達 96% 的人，從第一天開始就覺得自己運用得很順利。

指令行介面

你也可以在 CLI 指令行介面裡使用 Copilot。Copilot 的兩個主要功能，就是解釋指令和提出建議。

如果要使用這些功能，就必須安裝 GitHub CLI（*https://cli.github.com*），而且還要先登入到你的 GitHub 帳號：

```
gh auth login
```

接下來還要安裝 Copilot：

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

你可以把這個擴展功能升級到最新的版本：

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

下面就是要求 Copilot 解釋 CLI 指令的一個範例：

提示：gh copilot explain xcopy

圖 4-14 顯示的就是相應的輸出。

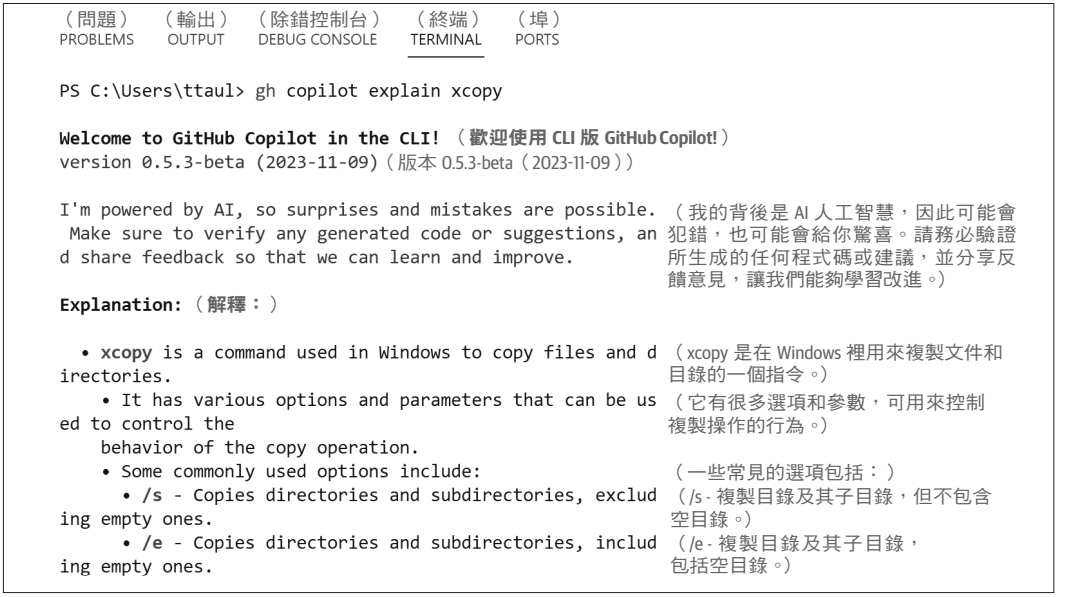

圖 4-14　Copilot 在 CLI 指令行介面裡，對 xcopy 這個指令進行解釋

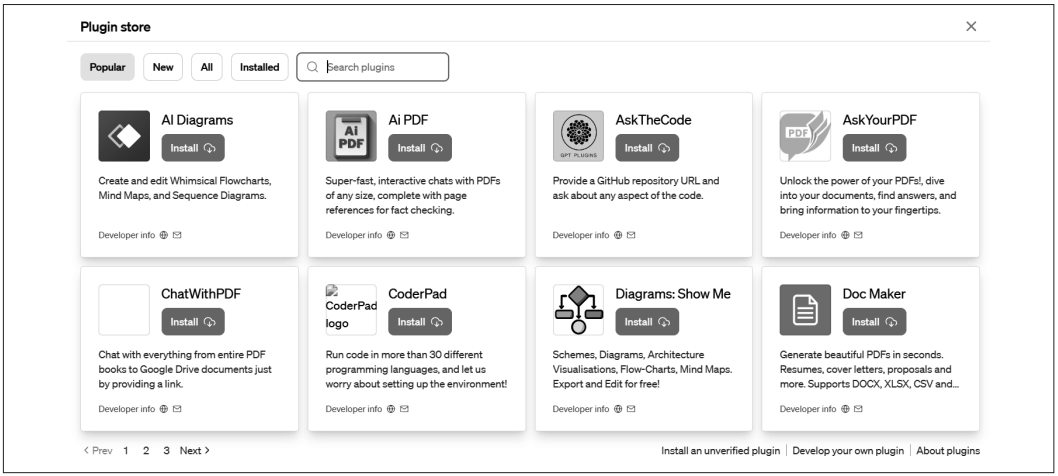

圖 6-6　ChatGPT 外掛商店可以讓你使用到各種善用 OpenAI LLM 強大功能的迷你 App

Codecademy 外掛

我們就來嘗試一下 Codecademy 的外掛吧。首先按下 Install（安裝）按鈕。如果要啟用它的話，請到畫面上方點擊向下的箭頭。然後再點擊 Codecademy 圖示。

這個外掛有兩個主要的功能。一個是讓使用者可以根據自己的目標和經驗程度，找出特定的課程或學習路徑。舉例來說，對於 AI 和 ChatGPT 有興趣的使用者，就可以向這個外掛詢問推薦的課程，然後它就會提供相關的課程列表，其中包括課程的說明，以及課程是免費還是要付費之類的資訊。

這個外掛的另一個功能，就是可以充當技術文件的快速參考工具，它會提供一些包含更詳細資訊的文件與文章的連結。

我們就來測試一下這部分的功能吧：

> 提示：What is the best doc or article for explaining arrays in JavaScript?
> （有什麼最好的文件或文章，可以解釋一下 JavaScript 陣列？）

ChatGPT 所提供的回應，如圖 6-7 所示。

最上面有一個圖示，顯示出系統正在使用 Codecademy 外掛。下方的文字則顯示了許多關於這個主題的資源。不過，這個外掛為了縮小範圍，在最後面只提供了三個文件的連結。

圖 6-7　Codecademy 外掛針對 JavaScript 陣列的資源請求所做出的回應

AskYourDatabase 外掛

Sheldon Niu 之所以會冒出這個外掛的構想，是因為他經常用 ChatGPT 來寫 SQL 語句，結果發現非常麻煩。他每次都要先解釋一下整個資料庫的資料架構，然後輸出還需要大量的複製貼上，才能在 Terminal 終端裡執行。於是他就想，「嘿！如果 ChatGPT 可以直接與資料庫聊天怎麼樣？」這就是他建立 AskYourDatabase 的開端。

你可以透過這個工具，運用 ChatGPT 輕鬆完成資料庫架構的原型設計。至於查詢資料，就變成小菜一碟了。更棒的是，這樣你就不需要再去用那些通常需要進行大量設定的傳統商業智慧（BI）工具了。

Recombinant AI 外掛

Mark Zahm 是一位開發者，他創立了自己的 AI 顧問事業。他建立的 Recombinant AI 外掛，可以讓一些使用 GitHub 和 Gitlab 程式碼儲存庫的開發者，日子過得更輕鬆一點。這個工具可以讓 ChatGPT 瞭解整個程式的重點，並梳理出一些很重要的細節。這樣一來，使用者就可以對自己的程式碼有更好的掌握，讓自己可以去進行各種調整與分析，或是把自己的想法融入到軟體之中。Mark 把這個外掛稱為「對話式 IDE」，這應該算是個相當貼切的說法。

下面就是它的一些應用方式：

- 執行一些傳統的任務，例如把一些程式碼片段或提示保存起來。
- 利用專案和檔案系統，建立一些複雜的任務列表，或是一些思維鏈（chain-of-thought）提示。
- 把最新的程式設計函式庫相關的一些重要資訊保存起來。

GPT

你可以建立自己的客製化 ChatGPT。這就是所謂的 GPT，你很容易就可以建構起來了（通常只需要幾分鐘）。

我們就來看個例子吧。我們會建立一個 GPT，來作為軟體開發風格指南。這其中充滿了各種具體的指導原則，例如該怎麼命名變數以讓它對大家都有意義、縮排程式碼的正確方式、還有團隊需要遵循的各種程式設計模式或實務做法。這其實就有點像是你的程式碼所要遵循的服裝穿著規定。

我們的構想，就是要讓一切保持一致、井然有序。這樣就可以讓大家更輕鬆閱讀與理解整個程式碼庫，進一步成為大型專案的救星。

不過，這裡還有個問題：如果你是個新手，一開始或許還是會有點不知所措。你可能已經很習慣用某種方式來寫程式碼，突然間你又必須去適應新的標準。

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

公司	主要功能	目標市場	獨特賣點
Salesforce	以雲端為基礎，具 AI 能力，廣泛的應用生態系統	中小企業至大型企業	市場領導者，高度可客製化，強大的整合能力
Microsoft Dynamics	與 Microsoft 產品無縫整合，具 AI 洞察力	中小企業至大型企業	與 Office 365 深度整合，靈活部署
Oracle CRM	全面套件、雲端解決方案、AI 驅動分析	中型至大型公司	高擴展性，針對特定行業的解決方案
SAP CRM	端到端 CRM 解決方案，強大分析能力，以雲端為基礎	大型企業	與 SAP ERP 深度整合，專注於大型企業
HubSpot	對使用者很友善，內部行銷工具，免費 CRM 方案	小型至中型企業	易於使用，特別適合內部行銷，具可擴展性
Zoho CRM	價格親民，適合小型企業，AI 驅動	小型至中型企業	成本效益高，良好的客製化能力，強大的行動端支援
Adobe Experience Cloud	專注於行銷、分析和內容管理	中型至大型公司	在行銷和內容管理方面很強大，AI 驅動

圖 7-2　ChatGPT 提供 CRM 市場的競爭分析，並以表格的形式來呈現