

The background of the cover is a dark blue-purple gradient, densely populated with various colorful geometric shapes and patterns. These include circles in shades of orange, yellow, green, pink, and blue. There are also elongated, capsule-like shapes in orange, yellow, and green, some with internal patterns. A white line drawing of a leaf or a stylized arrow is visible in the upper left. A network of white dots and lines is scattered throughout. The overall aesthetic is modern and tech-oriented.

Patterns of Application Development Using AI

Obie Fernandez

Foreword by Gregor Hohpe



Leanpub

繁體中文版

使用人工智慧的應用程式開發模式（繁體中文版）

Obie Fernandez

這本書的網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

這個版本發布於 2025-01-23



這是一本Leanpub書。Leanpub 通過簡潔出版過程賦予作者和出版商能力。簡潔出版是使用輕量級工具和多次迭代來獲得讀者反饋，並在你有了正確的書之後再開始建立動力的一種出版進行中的電子書的行為。

© 2025 Obie Fernandez

在 Twitter 上分享這本書！

請在 [Twitter](#) 上幫助作者 Obie Fernandez 推廣！

建議對這本書使用的 hashtag 是 [#poaduai](#)。

如果你想知道其他人對這本書的意見，可以點擊此連結在 Twitter 上搜索 hashtag：

[#poaduai](#)

獻給我強悍無比的女王，我的靈感女神，我的光芒與摯愛， *Victoria*

由 Obie Fernandez 所著的其他作品

[Patterns of Application Development Using AI](#)

[The Rails 8 Way](#)

[The Rails 7 Way](#)

[XML The Rails Way](#)

[Serverless](#)

[El Libro Principiante de Node](#)

[The Lean Enterprise](#)

Contents

Gregor Hohpe 序言	i
前言	ii
關於本書	iii
關於程式碼範例	iii
我不會涵蓋的內容	iii
這本書適合誰	iii
建立共同詞彙	iii
參與方式	iii
致謝	iv
插圖是怎麼回事？	iv
關於精實出版	iv
關於作者	v
簡介	1
關於軟體架構的思考	2
什麼是大型語言模型？	2
理解推理	4
思考效能表現	20
實驗不同的 LLM 模型	22
複合式人工智能系統	22

第一部分：基本方法與技術 28

收窄路徑 29

潛在空間：難以理解的浩瀚 31

如何“縮小”路徑 33

原始模型與指令調校模型的比較 36

提示詞工程 43

提示詞提煉 57

那麼微調呢？ 63

檢索增強生成 (RAG) 65

什麼是檢索增強生成？ 65

RAG 如何工作？ 65

為什麼要在您的應用程式中使用 RAG？ 65

在您的應用程式中實施 RAG 65

命題分塊 66

RAG 的實際應用案例 67

智慧查詢優化 (IQO) 67

重新排序 67

RAG 評估 (RAGAs) 67

挑戰與未來展望 69

眾多工作者 72

AI 工作者作為獨立可重用組件 73

帳戶管理 74

電子商務應用 75

醫療保健應用 85

AI 工作器作為流程管理器 87

將 AI 工作者整合到您的應用程式架構中 91

AI 工作者的可組合性和編排 93

將傳統 NLP 與 LLM 結合	101
工具使用	104
什麼是工具使用？	104
工具使用的潛力	106
工具使用工作流程	106
工具使用的最佳實踐	121
組合和鏈接工具	125
未來方向	126
串流處理	128
實作 ReplyStream	128
「對話循環」	136
自動續接	138
結論	140
自我修復數據	141
實際案例研究：修復損壞的 JSON	143
考量因素與禁忌症	147
情境內容生成	159
個人化	160
生產力	161
快速迭代和實驗	162
AI 驅動的本地化	164
使用者測試與回饋的重要性	166
生成式使用者介面	167
為使用者介面生成文案	168
定義生成式使用者介面	177
範例	179

轉向結果導向設計	181
挑戰與考量	182
未來展望與機會	184
智能工作流程編排	186
業務需求	187
主要優勢	187
關鍵模式	188
異常處理和恢復	190
在實踐中實施智能工作流程編排	193
監控與日誌記錄	210
可擴展性和效能考量	213
工作流程的測試和驗證	217
 第二部分：模式	 224
提示工程	225
思維鏈	226
模式切換	228
角色分配	229
提示物件	230
提示詞模板	231
結構化輸入輸出	232
提示鏈接	233
提示重寫器	234
回應圍欄	235
查詢分析器	236
查詢重寫器	238
腹語術師模式	239

離散元件	240
謂詞	241
API 外觀模式	242
結果解譯器	245
虛擬機器	246
規格與測試	246
人機協作系統 (HITL)	248
高層次模式	248
升級處理	249
反饋循環	250
被動資訊輻射	251
協作決策制定 (CDM)	253
持續學習	254
倫理考量	254
技術進步與未來展望	255
智慧錯誤處理	256
傳統錯誤處理方法	256
情境錯誤診斷	257
智慧型錯誤報告	258
預測性錯誤預防	259
智能錯誤恢復	259
個人化錯誤通訊	260
自適應錯誤處理工作流程	261
品質控制	262
評估器	263
護欄	265
防護機制和評估：一體兩面	266

詞彙表	267
詞彙表	267
Index	273

Gregor Hohpe 序言

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

前言

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

關於本書

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

關於程式碼範例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

我不會涵蓋的內容

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

這本書適合誰

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

建立共同詞彙

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

參與方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

致謝

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

插圖是怎麼回事？

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

關於精實出版

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

關於作者

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

簡介



如果你急於開始將 AI 大型語言模型（LLMs）整合到你的程式專案中，可以直接跳到後面章節的模式和程式碼範例。不過，要充分理解這些模式的威力和潛力，值得花點時間來了解更廣泛的背景和它們所代表的統一方法。

這些模式並不僅僅是孤立技術的集合，而是一個將 AI 整合到應用程式中的統一框架。我使用 Ruby on Rails，但這些模式應該適用於幾乎所有其他程式設計環境。它們解決了從資料管理和效能優化到使用者體驗和安全性等廣泛的問題，為強化傳統程式設計實踐提供了一個全面的 AI 功能工具箱。

每種類別的模式都針對在應用程式中整合 AI 元件時出現的特定挑戰或機會。通過理解這些模式之間的關係和協同效應，你可以就在何處以及如何最有效地應用 AI 做出明智的決定。

模式永遠不是規範性的解決方案，也不應該被視為這樣。它們是可適應的構建模塊，應該根據你自己獨特應用程式的特定需求和限制來調整。這些模式（像

軟體領域中的任何其他模式一樣）的成功應用取決於對問題領域、使用者需求和專案整體技術架構的深入理解。

關於軟體架構的思考

我在 1980 年代開始寫程式並投入駭客圈，即使在成為專業軟體開發人員之後，我也從未失去駭客思維。從一開始，我就對象牙塔中的軟體架構師究竟能為專案帶來什麼價值持有健康的懷疑態度。

我個人如此興奮於這波強大的新 AI 技術帶來的變革，其中一個原因是它對我們認知的軟體架構決策的影響。它挑戰了關於設計和實作軟體專案的「正確」方式的傳統觀念。它也挑戰了架構是否仍然可以主要被視為系統中難以改變的部分，因為 AI 增強正在使得在任何時候改變專案的任何部分變得比以往更容易。

也許我們正在進入軟體工程「後現代」方法的巔峰時期。在這種情況下，後現代指的是從傳統範式的根本性轉變，從開發人員負責編寫和維護每一行程式碼，轉向接受將資料操作、複雜演算法，甚至整個應用程式邏輯委託給第三方函式庫和外部 API 的想法。這種後現代轉變代表著從傳統的從頭開始建立應用程式的智慧的重大偏離，它挑戰開發人員重新思考他們在開發過程中的角色。

根據 Larry Wall 和其他像他這樣的駭客巨匠的教導，我一直相信優秀的程式設計師只寫絕對必要的程式碼。通過最小化寫出的程式碼量，我們可以更快地移動、減少錯誤的表面積、簡化維護，並改善應用程式的整體可靠性。更少的程式碼讓我們能夠專注於核心業務邏輯和使用者體驗，同時將其他工作委託給其他服務。

現在 AI 驅動的系統可以處理以前只能由人工編寫程式碼完成的任務，我們應該能夠更加高效和靈活，比以往任何時候都更專注於創造商業價值和使用者體驗。

當然，將專案的大部分委託給 AI 系統也有權衡取捨，比如可能失去控制權，以及需要健全的監控和反饋機制。這就是為什麼它需要一套新的技能和知識，包括至少對 AI 如何運作的一些基本理解。

什麼是大型語言模型？

大型語言模型（LLMs）是一種人工智慧模型，自從 OpenAI 在 2020 年推出 GPT-3 以來，已經引起了廣泛關注。LLMs 旨在以驚人的準確性和流暢性處理、理解和生成人類語言。在本節中，我們將簡要了解 LLMs 如何運作，以及為什麼它們非常適合構建智慧系統組件。

從本質上來說，LLMs 是基於深度學習演算法，特別是神經網路。這些網路由相互連接的節點或神經元組成，用於處理和傳輸信息。LLMs 常用的架構選擇是 Transformer 模型，它在處理文字等序列資料方面已經證明非常有效。

轉換器模型是基於注意力機制，主要用於處理序列數據的任務，如自然語言處理。轉換器能夠一次性處理輸入數據而非順序處理，這使其能更有效地捕捉長距離依賴關係。它們具有多層注意力機制，幫助模型關注輸入數據的不同部分，以理解上下文和關係。

大型語言模型的訓練過程涉及讓模型接觸大量文本數據，如書籍、文章、網站和程式碼庫。在訓練過程中，模型學習識別文本中的模式、關係和結構。它捕捉語言的統計特性，如語法規則、詞語關聯和上下文含義。

訓練大型語言模型所使用的關鍵技術之一是無監督學習。這意味著模型在沒有明確標記或指導的情況下從數據中學習。它通過分析訓練數據中詞語和短語的共現情況，自行發現模式和表示。這使得大型語言模型能夠深入理解語言及其複雜性。

大型語言模型的另一個重要方面是它們處理上下文的能力。在處理文本時，大型語言模型不僅考慮單個詞語，還考慮周圍的上下文。它們會考慮前面的詞語、句子，甚至段落，以理解文本的含義和意圖。這種上下文理解使大型語言模型能夠生成連貫且相關的回應。評估特定大型語言模型能力的主要方式之一，是考慮它們在生成回應時能處理的上下文範圍大小。

經過訓練後，大型語言模型可用於廣泛的語言相關任務。它們可以生成類人文本、回答問題、總結文件、翻譯語言，甚至編寫程式碼。大型語言模型的多功

能性使其在建立能與用戶互動、處理和分析文本數據，並生成有意義輸出的智能系統組件方面極具價值。

通過將大型語言模型整合到應用程式架構中，你可以創建能理解和處理用戶輸入、生成動態內容，並提供智能建議或行動的 AI 組件。但使用大型語言模型需要仔細考慮資源需求和性能權衡。大型語言模型計算密集，可能需要大量處理能力和記憶體（換句話說，就是金錢）才能運行。我們大多數人都需要評估將大型語言模型整合到應用程式中的成本影響，並相應採取行動。

理解推理

推理指的是模型根據新的、未見過的數據生成預測或輸出的過程。這是訓練好的模型用於根據用戶輸入做出決策或生成文本、圖像或其他內容的階段。

在訓練階段，AI 模型通過調整其參數來最小化預測中的錯誤，從而從大型數據集中學習。一旦訓練完成，模型就可以將所學應用到新數據上。推理是模型使用其學習到的模式和知識來生成輸出的方式。

對於大型語言模型來說，推理涉及接收提示詞或輸入文本，並產生連貫且與上下文相關的回應，以詞元流的形式（我們稍後會討論）。這可以是回答問題、完成句子、生成故事或翻譯文本等眾多任務中的任何一個。



與你我思考的方式相比，AI 模型通過推理進行的“思考”都是在一個無狀態的操作中完成的。也就是說，它的思考僅限於其生成過程。它必須真正地大聲思考，就像我問你一個問題，而只接受你以“意識流”方式的回應一樣。

大型語言模型有多種規模和類型

雖然實際上所有流行的大型語言模型（LLMs）都基於相同的核心轉換器架構並在巨大的文本數據集上訓練，但它們有不同的規模，並針對不同目的進行微

調。LLM 的規模，以其神經網路中的參數數量來衡量，對其能力有重大影響。擁有更多參數的大型模型，如 GPT-4（據傳擁有 1 到 2 萬億個參數），通常比小型模型更有知識和能力。然而，更大的模型也需要更多的計算能力來運行，這意味著通過 API 調用使用它們時會產生更高的費用。

為了使大型語言模型更實用並針對特定用例進行調整，基礎模型通常會在更有針對性的數據集上進行微調。例如，LLM 可能在大量對話語料庫上訓練，以專門用於對話式 AI。其他模型則在[程式碼上訓練](#)以賦予它們編程知識。甚至還有[專門為角色扮演式用戶互動而訓練的模型](#)！

檢索式與生成式模型的比較

在大型語言模型（LLMs）的領域中，生成回應主要有兩種方法：檢索式模型和生成式模型。每種方法都有其優勢和劣勢，了解它們之間的差異可以幫助你為特定用例選擇合適的模型。

檢索式模型

檢索式模型，也稱為資訊檢索模型，通過在大型預存文本數據庫中搜索並根據輸入查詢選擇最相關的段落來生成回應。這些模型不會從零開始生成新文本，而是將數據庫中的片段拼接在一起形成連貫的回應。

檢索式模型的主要優勢之一是能夠提供準確和最新的資訊。由於它們依賴於經過審核的文本數據庫，因此可以從可靠來源提取相關資訊並呈現給用戶。這使得它們特別適合需要精確、事實性答案的應用，例如問答系統或知識庫。

然而，檢索式模型也有一些限制。它們的表現取決於所搜索的數據庫質量，因此數據庫的品質和覆蓋範圍直接影響模型的性能。此外，這些模型可能難以生成連貫且自然的回應，因為它們僅限於數據庫中現有的文本。

本書不涉及純檢索模型的使用。

生成式模型

相比之下，生成式模型會根據訓練期間學習到的模式和關係從零開始創建新文本。這些模型運用其對語言的理解來生成針對輸入提示的新穎回應。

生成式模型的主要優勢在於能夠產生具有創造性、連貫性和上下文相關性的文本。它們可以進行開放式對話、生成故事，甚至編寫代碼。這使得它們特別適合需要更開放和動態互動的應用，如聊天機器人、內容創作和創意寫作助手。

但是，生成式模型有時會產生不一致或事實不正確的資訊，因為它們依賴於訓練期間學習到的模式，而不是經過審核的事實數據庫。它們也可能更容易受到偏見和產生幻覺的影響，生成看似合理但不一定真實的文本。

生成式 LLMs 的例子包括 OpenAI 的 GPT 系列（GPT-3、GPT-4）和 Anthropic 的 Claude。

混合模型

許多商業可用的 LLMs 將檢索和生成方法結合在混合模型中。這些模型使用檢索技術從數據庫中找到相關資訊，然後使用生成技術將這些資訊合成為連貫的回應。

混合模型旨在將檢索式模型的事實準確性與生成式模型的自然語言生成能力相結合。它們可以提供更可靠和最新的資訊，同時保持進行開放式對話的能力。

在選擇檢索式和生成式模型時，你應該考慮應程序的具體要求。如果主要目標是提供準確的事實資訊，檢索式模型可能是最佳選擇。如果應用需要更開放和創造性的互動，生成式模型可能更合適。混合模型在兩種方法之間提供平衡，對於既需要事實準確性又需要自然語言生成的應用來說是個不錯的選擇。

最終，選擇檢索式還是生成式模型取決於具體用例以及在準確性、創造力和靈活性之間的權衡。通過了解每種方法的優勢和局限性，你可以做出明智的決定。

指令微調

指令微調是會話訓練的一個子集，它使用人工編寫的提示和回應來提高 LLMs 理解和遵循指令的能力。以下是一些簡單的例子，說明如何通過對人工編寫的提示和回應進行指令微調來提高 LLMs 理解和遵循指令的能力：

示例 1：客戶支援

指令：“對要求退款的客戶提供回應。”

人工編寫的提示：“我想為最近的購買申請退款。這個產品沒有達到我的期望。”

人工編寫的回應：“很抱歉聽到產品沒有達到您的期望。為了處理您的退款，請提供您的訂單號碼和退貨原因。我們將審核您的請求，並在 2-3 個工作日內回覆您。”

示例 2：教育輔助

指令：“向一個 10 歲的孩子解釋光合作用的概念。”

人工撰寫的提示：「你能解釋植物是如何製造食物的嗎？」

人工撰寫的回應：「當然可以！植物通過一個叫做光合作用的過程製造食物。它們利用陽光、水和二氧化碳來製造食物和氧氣。植物的葉子有一種特殊的部分叫做葉綠素，幫助它們吸收陽光。」

示例 3：個人助理

指令：「安排下週二下午 3 點的會議。」

人工撰寫的提示：「你能幫我安排週二下午 3 點的會議嗎？」

人工撰寫的回應：「當然可以，我已經為您安排了下週二下午 3 點的會議。還需要其他幫助嗎？」

這形成了一個擁有不同規模和專長的大型語言模型生態系統。在 1-70 億參數範圍內的較小模型提供了良好的通用語言能力，同時運行效率更高。

- Mistral 7B
- Llama 3 8B
- Gemma 7B

約 30-70 億參數的中型模型提供更強的推理和執行指令的能力。

- Llama 3 70B
- Qwen2 70B
- Mixtral 8x22B

在將大型語言模型整合到應用程式時，你必須在模型的能力與實際因素（如成本、延遲、上下文長度和內容過濾）之間取得平衡。對於較簡單的語言任務，較小的指令微調模型通常是最佳選擇，而最大的模型可能需要用於複雜的推理或分析。模型的訓練數據也是一個重要考慮因素，因為它決定了模型的知識截止日期。



某些模型，比如來自 Perplexity 的部分模型，可以連接到即時信息源，因此它們實際上沒有截止日期。當你向它們提問時，它們能夠自主決定進行網路搜索並獲取任意網頁來生成答案。

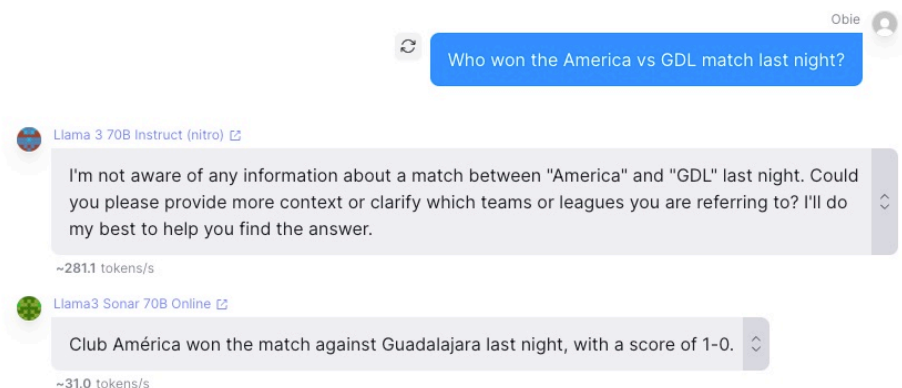


圖 1. 有無在線訪問功能的 Llama3 對比

最終，沒有一種通用的大型語言模型。了解模型大小、架構和訓練方面的差異對於選擇適合特定用例的模型至關重要。實驗不同的模型是揭示哪些模型能為特定任務提供最佳性能的唯一實用方法。

分詞：將文本分解成片段

在大型語言模型處理文本之前，需要將文本分解成更小的單位，稱為詞元（tokens）。詞元可以是單個詞、詞的部分，甚至是單個字符。將文本分割成詞元的過程稱為分詞，這是為語言模型準備數據的關鍵步驟。

The process of splitting text into tokens is known as tokenization, and it's a crucial step in preparing data for a language model.

圖 2. 這個句子包含 27 個詞元

不同的大型語言模型使用不同的分詞策略，這會對模型的性能和能力產生重大影響。大型語言模型使用的一些常見分詞器包括：

- **GPT（位元組對編碼）**：GPT 分詞器使用一種稱為位元組對編碼（BPE）的技術將文本分解為子詞單位。BPE 通過迭代合併文本語料庫

中最頻繁出現的位元組對，形成子詞詞元詞彙表。這使得分詞器能夠通過將罕見和新穎的詞分解成更常見的子詞片段來處理它們。GPT 分詞器被用於 GPT-3 和 GPT-4 等模型。

- **Llama (SentencePiece)**：Llama 分詞器使用 SentencePiece 函式庫，這是一個無監督文本分詞器和重組器。SentencePiece 將輸入文本視為 Unicode 字元序列，並根據訓練語料庫學習子詞彙表。它可以處理任何能以 Unicode 編碼的語言，使其特別適合多語言模型。Llama 分詞器被 Meta 的 Llama 和 Alpaca 等模型所使用。
- **SentencePiece (Unigram)**：SentencePiece 分詞器也可以使用一種稱為 Unigram 的不同演算法，該演算法基於子詞正則化技術。Unigram 分詞根據一元語言模型確定最佳子詞彙表，該模型為個別子詞單元分配機率。與 BPE 相比，這種方法可以產生更具語義意義的子詞。使用 Unigram 的 SentencePiece 被 Google 的 T5 和 BERT 等模型採用。
- **Google Gemini (多模態分詞)**：Google Gemini 使用的分詞方案旨在處理各種數據類型，包括文本、圖像、音訊、視頻和程式碼。這種多模態能力使 Gemini 能夠處理和整合不同形式的資訊。值得注意的是，Google Gemini 1.5 Pro 擁有可處理數百萬個標記的上下文視窗，遠大於之前的模型。這個龐大的上下文視窗使模型能夠處理更大範圍的上下文，可能導致更準確的回應。然而，重要的是要注意 Gemini 的分詞方案比其他模型更接近於每個字元一個標記。這意味著如果你習慣使用像 GPT 這樣的模型，使用 Gemini 模型的實際成本可能會顯著高於預期，因為 Google 的定價是基於字元而不是標記。

分詞器的選擇會影響 LLM 的多個方面，包括：

- **詞彙量**：分詞器決定了模型詞彙表的大小，即它能識別的唯一標記集合。更大、更細緻的詞彙表可以幫助模型處理更廣泛的詞語和短語，甚至成為多模態（能夠理解和生成不僅僅是文本），但這也會增加模型的記憶體需求和計算複雜度。

- **** 處理罕見和未知詞：**** 使用子詞單元的分詞器，如 BPE 和 SentencePiece，可以將罕見和未知詞分解為更常見的子詞片段。這使得模型能夠根據這些詞包含的子詞，對它之前沒見過的詞的含義做出有根據的推測。
- **** 多語言支援：**** 像 SentencePiece 這樣可以處理任何 Unicode 可編碼語言的分詞器，特別適合需要處理多種語言文本的多語言模型。

在為特定應用選擇 LLM 時，考慮其使用的分詞器以及它與任務特定語言處理需求的匹配程度非常重要。分詞器可能會對模型處理領域特定術語、罕見詞和多語言文本的能力產生重大影響。

上下文大小：語言模型在推理過程中可以使用多少信息？

在討論語言模型時，上下文大小指的是模型在處理或生成回應時可以考慮的文本量。它本質上是衡量模型能夠「記住」並用於影響其輸出的信息量（以標記表示）。語言模型的上下文大小可能會對其能力和可以有效執行的任務類型產生重大影響。

什麼是上下文大小？

從技術角度來說，上下文大小由語言模型在單個輸入序列中可以處理的標記（詞或詞片段）數量決定。這通常被稱為模型的「注意力範圍」或「上下文視窗」。上下文大小越大，模型在生成回應或執行任務時可以同時考慮的文本就越多。

不同的語言模型具有不同的上下文大小，從幾百個標記到數百萬個標記不等。作為參考，一個典型的段落可能包含大約 100-150 個標記，而整本書可能包含數萬或數十萬個標記。

甚至有研究在探索如何以有限的記憶體和計算量，讓基於 Transformer 的大型語言模型（LLMs）能夠處理無限長的輸入。

為什麼上下文大小很重要？

語言模型的上下文大小對其理解和生成連貫、具有上下文相關性的文本的能力有重大影響。以下是上下文大小重要的幾個主要原因：

1. **理解長篇內容：**具有較大上下文大小的模型能更好地理解和分析較長的文本，如文章、報告，甚至整本書。這對於文件摘要、問答和內容分析等任務至關重要。
2. **保持連貫性：**較大的上下文視窗使模型能在較長的輸出中保持連貫性和一致性。這對於故事生成、對話系統和內容創作等任務很重要，因為在這些任務中保持一致的敘事或主題至關重要。在使用大型語言模型生成或轉換結構化數據時，這一點也絕對重要。
3. **捕捉長距離依賴關係：**某些語言任務需要理解文本中相距較遠的詞語或短語之間的關係。具有較大上下文大小的模型更能捕捉這些長距離依賴關係，這對於情感分析、翻譯和語言理解等任務很重要。
4. **處理複雜指令：**在使用語言模型執行複雜的多步驟指令的應用中，較大的上下文大小使模型能在生成回應時考慮整套指令，而不是僅考慮最近的幾個詞。

不同上下文大小的語言模型示例

以下是一些具有不同上下文大小的語言模型示例：

- OpenAI GPT-3.5 Turbo：4,095 個詞元

- Mistral 7B Instruct：32,768 個詞元
- Anthropic Claude v1：100,000 個詞元
- OpenAI GPT-4 Turbo：128,000 個詞元
- Anthropic Claude v2：200,000 個詞元
- Google Gemini Pro 1.5：280 萬個詞元

如您所見，這些模型的上下文大小範圍很廣，從 OpenAI GPT-3.5 Turbo 模型的約 4,000 個詞元到 Anthropic Claude v2 模型的 200,000 個詞元不等。一些模型，如 Google 的 PaLM 2 和 OpenAI 的 GPT-4，提供具有更大上下文大小的不同版本（例如“32k”版本），可以處理更長的輸入序列。而目前（2024 年 4 月）Google Gemini Pro 聲稱擁有近 300 萬個詞元！

值得注意的是，上下文大小會因特定模型的實現和版本而異。例如，原始的 OpenAI GPT-4 模型的上下文大小為 8,191 個詞元，而後來的 GPT-4 變體（如 Turbo 和 4o）的上下文大小則大得多，達到 128,000 個詞元。

Sam Altman 將當前的上下文限制比作 80 年代個人電腦程序員必須處理的幾千字節工作記憶體，並表示在不久的將來，我們將能夠將“所有個人數據”都放入大型語言模型的上下文中。

選擇合適的上下文大小

在為特定應用選擇語言模型時，考慮任務所需的上下文大小要求很重要。對於涉及短小、獨立文本片段的任務（如情感分析或簡單問答），較小的上下文大小可能就足夠了。然而，對於需要理解和生成更長、更複雜文本的任務，較大的上下文大小可能是必要的。

值得注意的是，較大的上下文大小通常會帶來更高的計算成本和更慢的處理時間，因為模型在生成回應時需要考慮更多信息。因此，在選擇應用的語言模型時，您必須在上下文大小和性能之間取得平衡。

為什麼不直接選擇具有最大上下文大小的模型，並塞入盡可能多的信息呢？除了性能因素外，另一個主要考慮因素是成本。在 2024 年 3 月，使用 Google Gemini Pro 1.5 的完整上下文進行一次提示-回應循環的成本就接近 8 美元。如果您有能夠證明這種支出合理的用例，那就太好了！但對於大多數應用來說，這個成本實在太高了，差距有好幾個數量級。

大海撈針

在大型數據集中進行檢索的挑戰，長期以來都被比喻為大海撈針。在大型語言模型（LLM）的領域中，我們稍微調整了這個比喻。試想我們不是在浩瀚的文本中尋找單一事實（比如在 Paul Graham 的文章全集中），而是在尋找分散各處的多個事實。這種情況更像是在廣闊的田野中尋找多根針，而不是在單一的乾草堆中找尋。關鍵在於：我們不僅需要找到這些針，還要將它們編織成一個連貫的線索。

當需要從長篇上下文中檢索並推理多個事實時，LLM 面臨著雙重挑戰。首先是檢索準確性的直接問題——隨著事實數量的增加，準確性自然會下降。這是可以預期的；畢竟，在龐大的文本中追蹤多個細節即使對最複雜的模型來說也是一項艱鉅的任務。

其次，可能更為關鍵的是對這些事實進行推理的挑戰。找出事實是一回事，將它們整合成連貫的敘述或答案則是另一回事。這才是真正的考驗所在。LLM 在推理任務中的表現往往比簡單的檢索任務退化得更明顯。這種退化不僅關乎數量，還涉及上下文、相關性和推理之間的複雜互動。

為什麼會這樣？考慮人類認知中的記憶和注意力動態，這在某種程度上也反映在 LLM 中。在處理大量信息時，LLM 和人類一樣，在吸收新信息的過程中可能會遺失先前的細節。這在那些沒有被特別設計成自動優先處理或重新審視早期文本段落的模型中尤其明顯。

此外，LLM 將這些檢索到的事實編織成連貫回應的能力類似於敘事建構。這不僅需要信息的檢索，還需要深入的理解和上下文定位，這對當前的人工智能來說仍然是一個巨大的挑戰。

那麼，這對我們這些技術的開發者和整合者意味著什麼？在設計依賴 LLM 處理複雜長篇任務的系統時，我們需要敏銳地意識到這些限制。理解在某些條件下性能可能會降低，有助於我們設定現實的期望，並設計更好的備用機制或補充策略。

模態：超越文本

雖然當今大多數語言模型都專注於處理和生成文本，但越來越多的趨勢是朝向多模態模型發展，這些模型能夠原生輸入和輸出多種類型的數據，如圖像、音頻和視頻。這些多模態模型為能夠理解和生成跨模態內容的人工智能應用開闢了新的可能性。

什麼是模態？

在語言模型的背景下，模態指的是模型可以處理和生成的不同類型的數據。最常見的模態是文本，包括各種形式的書面語言，如書籍、文章、網站和社交媒體帖子。然而，還有其他幾種越來越多地被納入語言模型的模態：

- **圖像**：視覺數據，如照片、插圖和圖表。
- **音頻**：聲音數據，如語音、音樂和環境聲音。
- **視頻**：移動的視覺數據，通常伴隨著音頻，如視頻片段和電影。

每種模態都為語言模型帶來獨特的挑戰和機遇。例如，圖像要求模型理解視覺概念和關係，而音頻則要求模型處理和生成語音及其他聲音。

多模態語言模型

多模態語言模型被設計用來在單一模型中處理多種模態。這些模型通常具有專門的組件或層，可以理解輸入並在不同模態中生成輸出數據。以下是一些著名的多模態語言模型示例：

- **OpenAI 的 GPT-4o**：GPT-4o 是一個除了文本之外還能原理解和處理語音音頻的大型語言模型。這種能力使 GPT-4o 能夠執行諸如轉錄口語、從音頻輸入生成文本，以及基於口語查詢提供回應等任務。
- **OpenAI 的 GPT-4 視覺輸入**：GPT-4 是一個可以處理文本和圖像的大型語言模型。當給定圖像作為輸入時，GPT-4 可以分析圖像內容並生成描述或回應視覺信息的文本。
- **Google 的 Gemini**：Gemini 是一個可以處理文本、圖像和視頻的多模態模型。它使用統一的架構實現跨模態理解和生成，能夠執行圖像說明、視頻摘要和視覺問答等任務。
- ****DALL-E 和 Stable Diffusion**：雖然這些不是傳統意義上的語言模型，但它們通過從文字描述生成圖像展示了多模態人工智能的威力。它們展現了能夠在不同模態之間轉換的模型的潛力。

多模態模型的優勢與應用

多模態語言模型提供了多項優勢，並實現了廣泛的應用，包括：

- **** 增強理解**：通過處理來自多個模態的信息，這些模型能夠獲得對世界更全面的理解，類似於人類從各種感官輸入中學習的方式。
- **** 跨模態生成**：多模態模型能夠基於一種模態的輸入生成另一種模態的內容，例如從文字描述創建圖像，或從書面文章生成視頻摘要。
- **** 無障礙性**：多模態模型可以通過在模態之間轉換來使信息更容易獲取，例如為視障用戶生成圖像的文字描述，或創建書面內容的音頻版本。

- **** 創意應用：**多模態模型可用於創意任務，如基於文字提示生成藝術、音樂或視頻，為藝術家和內容創作者開闢新的可能性。

隨著多模態語言模型的不斷進步，它們很可能在能夠理解和生成跨多個模態內容的人工智能應用程序開發中發揮越來越重要的作用。這將實現人類與人工智能系統之間更自然、更直觀的互動，並為創意表達和知識傳播開啟新的可能性。

提供商生態系統

在將大型語言模型（LLMs）整合到應用程序時，您有越來越多的選擇可供選擇。每個主要的 LLM 提供商，如 OpenAI、Anthropic、Google 和 Cohere，都提供其自己的模型、API 和工具生態系統。選擇合適的提供商需要考慮各種因素，包括定價、性能、內容過濾、數據隱私和自定義選項。

OpenAI

OpenAI 是最著名的 LLM 提供商之一，其 GPT 系列（GPT-3、GPT-4）被廣泛應用於各種應用程序中。OpenAI 提供了一個用戶友好的 API，使您能夠輕鬆地將他們的模型整合到應用程序中。他們提供了一系列具有不同功能和價格點的模型，從入門級的 Ada 模型到功能強大的 Davinci 模型。

OpenAI 的生態系統還包括 OpenAI Playground 等工具，允許您試驗提示詞並針對特定用例微調模型。他們提供內容過濾選項，以幫助防止生成不當或有害內容。

在直接使用 OpenAI 的模型時，我依賴 Alex Rudall 的 [ruby-openai](#) 庫。

Anthropic

Anthropic 是 LLM 領域的另一個主要參與者，其 Claude 模型因強大的性能和道德考慮而日益受歡迎。Anthropic 專注於開發安全和負責任的人工智能系統，特別強調內容過濾和避免有害輸出。

Anthropic 的生態系統包括 Claude API，允許您將模型整合到應用程式中，以及用於提示詞工程和微調的工具。他們還提供 Claude Instant 模型，該模型整合了網絡搜索功能，以提供更新和更準確的回應。

在直接使用 Anthropic 的模型時，我依賴 Alex Rudall 的[anthropic](#)庫。

Google

Google 開發了幾個強大的 LLM，包括 Gemini、BERT、T5 和 PaLM。這些模型以在廣泛的自然語言處理任務中的強大性能而聞名。Google 的生態系統包括 TensorFlow 和 Keras 庫，提供用於構建和訓練機器學習模型的工具和框架。

Google 還提供了 Cloud AI Platform，使您能夠輕鬆地在雲端部署和擴展他們的模型。他們為情感分析、實體識別和翻譯等任務提供了一系列預訓練模型和 API。

Meta

Meta（前身為 Facebook）深度投入大型語言模型的開發，通過發布 LLaMA 和 OPT 等模型彰顯其成就。這些模型以在各種語言任務中的出色表現而著稱，主要通過開源渠道提供，支持 Meta 對研究和社區協作的承諾。

Meta 的生態系統主要圍繞 PyTorch 構建，這是一個開源機器學習庫，因其動態計算能力和靈活性而備受青睞，有助於創新的人工智能研究和開發。

除了技術方面的產品，Meta 在人工智慧的道德發展上也相當重視。他們實施了強大的內容過濾系統，並致力於減少偏見，與其在人工智慧應用方面追求安全和負責任的更廣泛目標保持一致。

Cohere

Cohere 是大型語言模型領域中較新的參與者，專注於讓大型語言模型比競爭對手更容易使用和存取。他們的生態系統包括 Cohere API，提供各種預訓練模型的存取，可用於文字生成、分類和摘要等任務。

Cohere 還提供提示詞工程、微調和內容過濾的工具。他們強調資料隱私和安全性，具有加密資料儲存和存取控制等功能。

Ollama

Ollama 是一個自託管平台，允許使用者在本機上管理和部署各種大型語言模型 (LLMs)，讓使用者可以完全控制其人工智慧模型，而不需依賴外部雲端服務。對於重視資料隱私並希望在內部處理人工智慧操作的使用者來說，這是理想的設置。

該平台支援多種模型，包括 Llama、Phi、Gemma 和 Mistral 的各種版本，這些模型在大小和運算需求上各有不同。Ollama 讓使用者可以使用簡單的命令如 `ollama run <model_name>` 直接從命令列下載和運行這些模型，並且設計可以在 macOS、Linux 和 Windows 等不同作業系統上運作。

對於希望將開源模型整合到應用程式中而不使用遠端 API 的開發者來說，Ollama 提供了類似容器管理工具的 CLI 來管理模型生命週期。它還支援自定義配置和提示詞，允許高度客製化以適應特定需求或使用案例。

由於其命令列介面和在管理和部署人工智慧模型方面提供的靈活性，Ollama 特別適合精通技術的使用者和開發者。這使其成為需要強大人工智慧功能但又不想在安全性和控制方面妥協的企業和個人的強大工具。

多模型平台

此外，還有一些提供商託管各種開源模型，例如 Together.ai 和 Groq。這些平台提供靈活性和客製化選項，讓你能夠根據特定需求運行，在某些情況下甚至可以微調開源模型。例如，Together.ai 提供對多種開源大型語言模型的存取，使用者可以實驗不同的模型和配置。Groq 專注於提供超高性能的完成功能，在本書撰寫時看起來幾乎是魔法般的表現。

選擇大型語言模型供應商

在選擇大型語言模型供應商時，你應該考慮以下因素：

- **價格**：不同供應商提供不同的定價模式，從按使用量付費到訂閱制計畫都有。在選擇供應商時，必須考慮預期使用量和預算。
- **性能**：大型語言模型的性能在不同供應商之間可能有顯著差異，因此在做決定之前，重要的是要針對特定使用案例進行基準測試。
- **內容過濾**：根據應用程式的不同，內容過濾可能是一個關鍵考慮因素。某些供應商提供比其他供應商更強大的內容過濾選項。
- **資料隱私**：如果應用程式處理敏感的使用者資料，選擇具有強大資料隱私和安全實踐的供應商很重要。
- **客製化**：某些供應商在微調和客製化模型以適應特定使用案例方面提供更多靈活性。

最終，大型語言模型供應商的選擇取決於應用程式的具體要求和限制。通過仔細評估選項並考慮價格、性能和資料隱私等因素，你可以選擇最符合需求的供應商。

值得注意的是，大型語言模型的領域正在不斷發展，新的供應商和模型經常出現。你應該隨時了解最新發展，並願意探索新的選擇。

OpenRouter

在本書中，我將專門使用 [OpenRouter](#) 作為我的 API 供應商。原因很簡單：它是所有最受歡迎的商業和開源模型的一站式商店。如果你急於開始進行一些人工智慧編碼，最好的起點之一是我的 [OpenRouter Ruby Library](#)。

思考效能表現

在將語言模型整合到應用程式中時，效能是一個關鍵考量因素。語言模型的效能可以從延遲（生成回應所需的時間）和吞吐量（每單位時間可處理的請求數量）來衡量。

首字符響應時間（TTFT）是另一個重要的效能指標，特別是對於聊天機器人和需要即時互動回應的應用程式而言。TTFT 衡量從接收用戶請求到生成回應的第一個字（或標記）所需的延遲時間。這個指標對於維持流暢且吸引人的用戶體驗至關重要，因為回應延遲可能導致用戶感到沮喪並失去興趣。

這些效能指標可能對用戶體驗和應用程式的可擴展性產生重大影響。

幾個因素可能影響語言模型的效能表現，包括：

參數數量：較大的模型具有更多參數，通常需要更多計算資源，與較小的模型相比可能會有更高的延遲和較低的吞吐量。

硬體：語言模型的效能表現可能會因運行的硬體而有顯著差異。雲端供應商提供針對機器學習工作負載優化的 GPU 和 TPU 實例，這可以大大加速模型推理。



OpenRouter的一個優點是，對於它提供的許多模型，你可以選擇具有不同效能配置和成本的雲端供應商。

量化：量化技術可以通過使用較低精度的數據類型來表示權重和激活值，從而減少模型的記憶體佔用和計算需求。這可以在不顯著犧牲品質的情況下提高效能。作為應用程式開發者，你可能不會參與以不同量化級別訓練自己的模型，但至少熟悉這些術語是有好處的。

**** 批次處理：**同時處理多個請求可以通過分攤模型載入和數據傳輸的開銷來提高吞吐量。

**** 快取：**對經常使用的提示或輸入序列進行結果快取可以減少推理請求的數量並提高整體效能。

在為生產應用程式選擇語言模型時，重要的是要在具有代表性的工作負載和硬體配置上進行基準測試。這可以幫助識別潛在的瓶頸，並確保模型能夠達到所需的效能目標。

同時也值得考慮模型效能與其他因素（如成本、靈活性和整合便利性）之間的權衡。例如，對於需要即時回應的應用程式來說，使用延遲較低的較小且較便

宜的模型可能更可取，而較大、更強大的模型可能更適合批次處理或複雜推理任務。

實驗不同的 LLM 模型

選擇 LLM 很少是一個永久性的決定。由於新的和改進的模型會定期發布，以模組化方式構建應用程式是很好的做法，這樣可以隨時替換不同的語言模型。提示和數據集通常可以在模型之間重複使用，只需進行最小的更改。這使你能夠利用語言建模的最新進展，而無需完全重新設計應用程式。



能夠輕鬆在眾多模型選擇之間切換，這是我喜歡 OpenRouter 的又一個原因。

在升級到新的語言模型時，重要的是要徹底測試和驗證其效能和輸出品質，以確保它符合應用程式的要求。這可能涉及在特定領域數據上重新訓練或微調模型，以及更新依賴於模型輸出的任何下游組件。

通過在設計應用程式時考慮效能和模組化，你可以創建可擴展、高效且面向未來的系統，能夠適應快速發展的語言建模技術領域。

複合式人工智能系統

在結束我們的介紹之前，值得一提的是，在 2023 年之前，以及 ChatGPT 引發的生成式人工智能熱潮之前，傳統的人工智能方法通常依賴於單一、封閉模型的整合。相比之下，複合式人工智能系統利用相互連接的組件形成複雜的管道，共同協作以實現智能行為。

從本質上來說，複合式人工智能系統由多個模組組成，每個模組都設計用於執行特定的任務或功能。這些模組可以包括生成器、檢索器、排序器、分類器和各種其他專門的組件。通過將整個系統分解為較小的、專注的單元，開發者可以創建更靈活、可擴展且易於維護的人工智能架構。

複合式人工智慧系統的主要優勢之一在於其能夠結合不同人工智慧技術和模型的優點。例如，系統可能使用大型語言模型（LLM）來進行自然語言理解和生成，同時採用獨立的模型進行資訊檢索或基於規則的決策。這種模組化方法使您能夠為每個特定任務選擇最佳的工具和技術，而不是依賴一體適用的解決方案。

然而，構建複合式人工智慧系統也帶來獨特的挑戰。特別是，確保系統行為的整體連貫性和一致性需要健全的測試、監控和治理機制。



像 GPT-4 這樣強大的 LLM 的出現，使我們比以往更容易實驗複合式人工智慧系統，因為這些先進的模型除了具備自然語言理解能力外，還能在複合系統中擔任多個角色，如分類、排序和生成。這種多功能性使開發人員能夠快速建立原型並迭代複合式人工智慧架構，為智能應用程式開發開創新的可能性。

複合式人工智慧系統的部署模式

複合式人工智慧系統可以使用各種模式進行部署，每種模式都旨在滿足特定的需求和使用案例。讓我們探討四種常見的部署模式：問答系統、多代理/主動式問題解決器、對話式人工智慧和智能助手。

問答系統

問答（Q&A）系統專注於提供經人工智慧模型理解能力增強的資訊檢索，使其功能超越單純的搜尋引擎。通過將強大的語言模型與外部知識來源結合，使用[檢索增強生成（RAG）](#)，問答系統可以避免產生幻覺，並為用戶查詢提供準確且具有上下文相關性的回答。

基於 LLM 的問答系統的關鍵組件包括：

- **查詢理解和重構**：分析用戶查詢並重新構建，以更好地匹配底層知識來源。

- **知識檢索：**根據重構後的查詢從結構化或非結構化數據源中檢索相關資訊。
- **回應生成：**通過整合檢索到的知識與語言模型的生成能力，產生連貫且資訊豐富的回應。

RAG 子系統在提供準確和最新資訊至關重要的問答領域特別重要，如客戶支援、知識管理或教育應用。

多代理/主動式問題解決器

多代理（也稱為主動式）系統由多個自主代理組成，共同合作解決複雜問題。每個代理都有特定的角色、技能組合以及訪問相關工具或資訊來源的權限。通過協作和資訊交換，這些代理能夠處理單個代理難以或無法獨立完成的任務。

多代理問題解決器的關鍵原則包括：

- **專業化：**每個代理專注於問題的特定方面，運用其獨特的能力和知識。
- **協作：**代理通過消息傳遞或共享記憶體進行溝通和協調行動，以實現共同目標。
- **適應性：**系統可以通過調整個別代理的角色和行為來適應變化的條件或需求。

多代理系統特別適用於需要分散式問題解決的應用，如供應鏈優化、交通管理或緊急響應規劃。

對話式人工智慧

對話式人工智慧系統實現用戶與智能代理之間的自然語言互動。這些系統結合了自然語言理解、對話管理和語言生成功能，以提供引人入勝且個性化的對話體驗。

對話式人工智慧系統的主要組件包括：

- **意圖識別**：根據用戶輸入識別其意圖，如提問、提出請求或表達情感。
- **實體提取**：從用戶輸入中提取相關實體或參數，如日期、地點或產品名稱。
- **對話管理**：維護對話狀態，根據用戶意圖和上下文確定適當的回應，並處理多輪互動。
- **回應生成**：使用語言模型、模板或基於檢索的方法生成類人回應。

對話式人工智慧系統常用於客服聊天機器人、虛擬助理和語音控制界面。如前所述，本書中的大多數方法、模式和代碼示例都直接源自我在一個名為Olympia的大型對話式人工智慧系統上的工作。

智能副駕駛

智能副駕駛是與人類用戶並肩工作的人工智能助理，用於提升使用者的生產力和決策能力。這些系統結合了自然語言處理、機器學習和特定領域知識，以提供智能建議、自動化任務並提供情境支援。

智能副駕駛的主要特點包括：

- **個人化**：適應個別用戶的偏好、工作流程和溝通方式。
- **主動協助**：預測用戶需求，無需明確指令即可提供相關建議或行動。
- **持續學習**：通過用戶反饋、互動和數據不斷改進性能。

智能副駕駛在各個領域的應用日益廣泛，例如軟體開發（如代碼補全和錯誤檢測）、創意寫作（如內容建議和編輯），以及數據分析（如見解和視覺化建議）

這些部署模式展示了複合式人工智能系統的多功能性和潛力。通過理解每種模式的特點和使用場景，您可以在設計和實施智能應用時做出明智的決策。雖然本書並非專門討論複合式人工智能系統的實施，但在將離散人工智能組件整合到傳統應用程式開發中時，許多（如果不是全部的話）相同的方法和模式都適用。

複合式人工智能系統中的角色

複合式人工智能系統建立在相互連接的模組基礎之上，每個模組都被設計用於執行特定角色。這些模組協同工作，創造智能行為並解決複雜問題。在考慮可能在應用程式中實施或替換哪些部分為離散人工智能組件時，熟悉這些角色會很有幫助。

生成器

生成器負責根據學習到的模式或輸入提示產生新的數據或內容。人工智能世界有許多不同類型的生成器，但在本書展示的語言模型背景下，生成器可以創建類人文本、完成部分句子或生成對用戶查詢的回應。它們在內容創建、對話生成和數據增強等任務中發揮著關鍵作用。

檢索器

檢索器用於在大型數據集或知識庫中搜索和提取相關信息。它們運用語義搜索、關鍵字匹配或向量相似度等技術，根據給定的查詢或上下文找到最相關的數據點。檢索器對於需要快速訪問特定信息的任務至關重要，例如問答系統、事實核查或內容推薦。

排序器

排序器負責根據特定標準或相關性分數對一組項目進行排序或優先級排列。它們為每個項目分配權重或分數，然後相應地進行排序。排序器常用於搜索引擎、推薦系統或任何需要向用戶呈現最相關結果的應用程式中。

分類器

分類器用於根據預定義的類別對數據點進行分類或標記。它們從已標記的訓練數據中學習，然後預測新的、未見實例的類別。分類器是情感分析、垃圾郵件檢測或圖像識別等任務的基礎，這些任務的目標是為每個輸入分配特定類別。

工具與代理程式

除了這些核心角色外，複合式人工智能系統通常還包含工具和代理程式，以增強其功能性和適應性：

- **工具：**工具是執行特定操作或計算的離散軟體組件或 API。它們可以被其他模組（如生成器或檢索器）調用，以完成子任務或收集額外信息。工具的例子包括網絡搜索引擎、計算器或數據視覺化庫。
- **代理程式：**代理程式是能夠感知環境、做出決策並採取行動以實現特定目標的自主實體。它們通常依賴規劃、推理和學習等多種人工智能技術的組合，以在動態或不確定的條件下有效運作。代理程式可用於模擬複雜行為或協調複合式人工智能系統內多個模組的行動。

在純粹的複合式人工智能系統中，這些組件之間的互動通過明確定義的接口和通信協議進行協調。數據在模組之間流動，一個組件的輸出作為另一個組件的輸入。這種模組化架構允許靈活性、可擴展性和可維護性，因為個別組件可以在不影響整個系統的情況下進行更新、替換或擴展。

通過利用這些組件及其互動的力量，複合式人工智能系統可以解決需要結合不同人工智能能力的複雜現實問題。在我們探索將人工智能整合到應用程式開發中的方法和模式時，請記住，複合式人工智能系統中使用的相同原則和技術可以用於創建智能、適應性和以用戶為中心的應用程式。

在第一部分的後續章節中，我們將深入探討將人工智能組件整合到應用程式開發過程中的基本方法和技術。從提示工程和檢索增強生成到自我修復數據和智能工作流程編排，我們將涵蓋廣泛的模式和最佳實踐，幫助您構建尖端的人工智能驅動應用程式。

第一部分：基本方法與技術

本書這一部分介紹了在應用程式中整合 AI 的不同方式。各章節涵蓋了一系列相關的方法和技術，從較高層次的概念如[縮小路徑](#)和[檢索增強生成](#)，一直到如何在 LLM 聊天完成 API 之上程式設計自己的抽象層等想法。

本書這一部分的目標是幫助你理解可以用 AI 實現的各種行為，然後再深入探討[第二部分](#)著重的具體實作模式。

第一部分的方法基於我在程式碼中使用過的想法、企業應用程式架構與整合的經典模式，以及我在向他人（包括非技術領域的業務相關人員）解釋 AI 功能時使用的比喻。

收窄路徑



“收窄路徑”指的是讓 AI 專注於當前的任務。每當我對 AI 表現“愚鈍”或出現意外行為而感到沮喪時，我都會用這句話作為座右銘。這個座右銘提醒我，失敗可能是我的錯，我應該進一步收窄路徑。

收窄路徑的需求源於大型語言模型中包含的海量知識，特別是像 OpenAI 和 Anthropic 這樣的世界級模型，它們擁有數以萬億計的參數。

擁有如此廣泛的知識無疑是強大的，並產生了諸如心智理論和類人推理能力等衍生行為。然而，這種驚人的信息量在針對特定提示生成精確和準確的回應時

也帶來了挑戰，尤其是當這些提示需要表現出可以與“常規“軟件開發和算法集成的確定性行為時。

導致這些挑戰的因素有很多。

**** 信息過載：**大型語言模型接受了橫跨各個領域、來源和時期的海量數據訓練。這種廣泛的知識使它們能夠參與各種話題，並基於對世界的全面理解生成回應。然而，當面對特定提示時，模型可能難以過濾掉不相關、矛盾或過時/過期的信息，導致回應缺乏重點或準確性。根據你想要完成的任務，模型可獲取的矛盾信息量之大可能輕易壓倒其提供你所尋求答案或行為的能力。

**** 上下文歧義：**考慮到廣闊的潛在空間中的知識，大型語言模型在理解你的提示的上下文時可能遇到歧義。如果沒有適當的收窄或引導，模型可能會生成與你的意圖只有間接關係而非直接相關的回應。這種失敗會導致回應偏離主題、不一致或未能滿足你明確的需求。在這種情況下，收窄路徑指的是上下文消歧，確保你提供的上下文使模型只關注其基礎知識中最相關的信息。



注意：當你剛開始“提示詞工程“時，你更可能在沒有適當解釋期望結果的情況下要求模型完成任務；要避免模糊不清需要練習！

**** 時間不一致性：**由於語言模型是在不同時期創建的數據上訓練的，它們可能擁有過時、被取代或不再準確的知識。例如，關於時事、科學發現或技術進步的信息可能在模型的訓練數據收集後已經發生了變化。如果不收窄路徑以優先考慮更近期和可靠的來源，模型可能會根據過時或錯誤的信息生成回應，導致其輸出中出現不準確和不一致。

**** 領域特定的細微差別：**不同的領域和領域都有其特定的術語、慣例和知識庫。想想幾乎任何 TLA（三字母縮寫）你就會意識到它們中的大多數都有不止一個含義。例如，MSK 可以指代 Amazon 的 Managed Streaming for Apache Kafka、Memorial Sloan Kettering Cancer Center 或人體肌骨骼系統。

當提示需要特定領域的專業知識時，大型語言模型的通用知識可能不足以提供準確和細緻的回應。通過提示詞工程或檢索增強生成來收窄路徑，專注於領域

特定信息，可以使模型生成更符合你特定領域要求和期望的回應。

潛在空間：難以理解的浩瀚

當我提到語言模型的“潛在空間”時，我指的是模型在訓練過程中學習到的知識和信息的廣闊多維景觀。它就像模型神經網絡中的一個隱藏領域，儲存著所有語言的模式、關聯和表徵。

想像你正在探索一片充滿無數相互連接節點的廣闊未知領域。每個節點代表模型學習到的一條信息、一個概念或一種關係。當你在這個空間中導航時，你會發現有些節點距離較近，表示強烈的連接或相似性，而其他節點則相距較遠，暗示較弱或較遠的關係。

潛在空間的挑戰在於它極其複雜且具有高維度特性。可以將它想像成如同我們的物理宇宙一般浩瀚，擁有星系團以及它們之間難以想像的巨大虛空距離。

由於它包含數千個維度，潛在空間並不能被人類直接觀察或理解。它是模型內部用於處理和生成語言的抽象表示。當您向模型提供輸入提示時，它本質上是將該提示映射到潛在空間中的特定位置。然後，模型利用該空間中的周圍信息和連接來生成回應。

關鍵在於，模型從其訓練數據中學習了大量信息，但並非所有信息都與特定任務相關或準確。這就是為什麼縮小路徑變得如此重要。通過在提示中提供清晰的指示、示例和上下文，您實際上是在引導模型專注於潛在空間中與您期望輸出最相關的特定區域。

另一種思考方式是將其比作在完全黑暗的博物館中使用聚光燈。如果您曾經參觀過羅浮宮或大都會藝術博物館，那就是我所說的規模。潛在空間就像這座博物館，充滿了無數的物件和細節。您的提示就像聚光燈，照亮特定區域，引導模型注意最重要的信息。沒有這種引導，模型可能會在潛在空間中漫無目的地遊走，在途中獲取無關或矛盾的信息。

在使用語言模型和製作提示時，請記住潛在空間的概念。您的目標是有效地在這片浩瀚的知識景觀中導航，引導模型朝向與您任務最相關和最準確的信息。

通過縮小路徑並提供清晰的指導，您可以釋放模型潛在空間的全部潛力，生成高質量、連貫的回應。

雖然前面對語言模型和它們所導航的潛在空間的描述可能看起來有點神奇或抽象，但重要的是要理解提示並不是咒語或咒文。語言模型的工作方式是建立在線性代數和概率理論的原理之上的。

從本質上講，語言模型是文本的概率模型，就像鐘形曲線是數據的統計模型一樣。它們通過一個稱為自迴歸建模的過程進行訓練，其中模型學會根據序列中前面的詞來預測下一個詞的概率。在訓練過程中，模型從隨機權重開始，並逐漸調整這些權重，以便為與其訓練樣本相似的文本分配更高的概率。

然而，將語言模型視為簡單的統計模型（如線性迴歸）並不能為理解它們的行為提供最佳直覺。更恰當的類比是將它們視為概率程序，這種模型允許操作隨機變量，並且可以表示複雜的統計關係。

概率程序可以用圖形模型來表示，這為理解模型中變量之間的依賴關係和聯繫提供了一種視覺化方式。這種視角可以為理解像 GPT-4 和 Claude 這樣的複雜文本生成模型的工作原理提供寶貴的見解。

在 Dohan 等人的論文《Language Model Cascades》中，作者深入探討了如何將概率程序應用於語言模型。他們展示了如何使用這個框架來理解這些模型的行為，並指導開發更有效的提示策略。

從這個概率視角得到的一個關鍵見解是，語言模型本質上創造了一個通往存在所需文檔的平行宇宙的入口。模型根據所有可能文檔的概率為它們分配權重，有效地縮小可能性空間，專注於最相關的內容。

這將我們帶回「縮小路徑」的核心主題。提示的主要目標是以一種方式調節概率模型，使其預測的質量集中在我們想要引出的特定信息或行為上。通過提供精心製作的提示，我們可以引導模型更有效地導航潛在空間，生成更相關和連貫的輸出。

然而，重要的是要記住，語言模型最終受限於其訓練數據中的信息。雖然它可以生成類似於現有文檔的文本或以新穎的方式組合想法，但它無法憑空創造全

新的信息。例如，如果在其訓練數據中沒有發現和記錄癌症的治癒方法，我們就不能期望模型提供這樣的方法。

相反，模型的優勢在於能夠找到並綜合與我們提供的提示相類似的信息。通過理解這些模型的機率特性，以及如何運用提示來調節其輸出，我們就能更有效地利用它們的能力來產生有價值的見解和內容。

請考慮以下的提示。在第一個提示中，單獨的“Mercury”可能指的是行星、化學元素，或是羅馬神祇，但最有可能指的是行星。事實上，GPT-4 提供了一個以「水星是太陽系中最小且最靠近太陽的行星...」開頭的長篇回答。第二個提示明確指的是化學元素。第三個提示則是指那位以速度見長、擔任神使的羅馬神話人物。

```
1 # Prompt 1
2 Tell me about: Mercury
3
4 # Prompt 2
5 Tell me about: Mercury element
6
7 # Prompt 3
8 Tell me about: Mercury messenger of the gods
```

只要加上幾個額外的詞，我們就完全改變了 AI 的反應方式。正如您將在本書後面了解到的，花俏的提示工程技巧，如 n-shot 提示、結構化輸入/輸出，以及[思維鏈](#)，其實都只是調節模型輸出的巧妙方法。

因此，提示工程的藝術本質上就是了解如何在語言模型知識的廣闊概率景觀中導航，以縮小通往特定信息或行為的路徑。

對於那些具有紮實高等數學基礎的讀者來說，從概率論和線性代數原理來理解這些模型確實會有所幫助！至於其他想要開發有效策略來獲得所需輸出的讀者，讓我們繼續使用更直觀的方法。

如何“縮小”路徑

為了應對知識過多的挑戰，我們採用各種技術來引導語言模型的生成過程，並使其注意力集中在最相關和準確的信息上。

以下是最重要的技術，按推薦順序排列，也就是說，您應該先嘗試提示工程，然後是 RAG，最後，如果必要的話，才考慮微調。

提示工程 最基本的方法是製作包含特定指令、約束或示例的提示，以引導模型的回應生成。本章在[下一節](#)中介紹提示工程的基礎知識，而在本書第二部分中，我們會介紹許多具體的提示工程模式。這些模式包括[提示精煉](#)，這是一種專注於改進和優化提示的技術，以提取 AI 認為最相關和簡潔的信息。

上下文增強 在提示時從外部知識庫或文檔中動態檢索相關信息，為模型提供重點上下文。流行的上下文增強技術包括[檢索增強生成 \(RAG\)](#)。像 [Perplexity](#) 提供的所謂“在線模型”能夠通過實時互聯網搜索結果來增強其上下文。



儘管 LLM 功能強大，但它們並未在您的獨特數據集上進行訓練，這些數據可能是私有的或針對您試圖解決的特定問題。上下文增強技術使 LLM 能夠訪問 API 背後的數據、SQL 數據庫中的數據，或困在 PDF 和演示文稿中的數據。

微調或領域適應 在特定領域的數據集上訓練模型，以專門化其知識和生成能力，以適應特定任務或領域。

降低溫度值

溫度是轉換器基礎語言模型中使用的一個超參數，用於控制生成文本的隨機性和創造性。它是一個介於 0 和 1 之間的值，較低的值使輸出更加集中和確定性，而較高的值則使其更加多樣化和不可預測。

當溫度設置為 1 時，語言模型根據下一個詞元的完整概率分布生成文本，允許更具創造性和多樣化的回應。然而，這也可能導致模型生成相關性較低或連貫性較差的文本。

另一方面，當溫度設置為 0 時，語言模型總是選擇概率最高的詞元，有效地“縮小其路徑”。我的幾乎所有 AI 組件都使用設置為 0 或接近 0 的溫度，因為這會產生更集中和可預測的回應。當您希望模型遵循指令、注意已提供的功能，或者只是需要比現有更準確和相關的回應時，這絕對有用。

例如，如果您正在構建需要提供事實信息的聊天機器人，您可能希望將溫度設置為較低的值，以確保回應更準確和切題。相反，如果您正在構建創意寫作助手，您可能希望將溫度設置為較高的值，以鼓勵更多樣化和富有想像力的輸出。

超參數：推理的旋鈕和調節器

在使用語言模型時，您會經常遇到“超參數”這個詞。在推理（即，當您使用模型生成回應時）的背景下，超參數就像是您可以調節的旋鈕和調節器，用於控制模型的行為和輸出。

可以將其想像為調整複雜機器的設置。就像您可能轉動旋鈕來控制溫度或翻動開關來改變操作模式一樣，超參數允許您精細地調整語言模型處理和生成文本的方式。

在推理過程中，你會遇到的一些常見超參數包括：

- **溫度值**：如前所述，這個參數控制生成文本的隨機性和創造性。較高的溫度值會產生更多樣化和不可預測的輸出，而較低的溫度值則會產生更集中和確定性的回應。
- ****Top-p（核採樣）****：這個參數控制累積概率超過特定閾值（p）的最小標記集合。它允許產生更多樣化的輸出，同時仍能保持連貫性。

- ****Top-k 採樣****：這種技術選擇概率最高的 k 個下一個標記，並在它們之間重新分配概率質量。它可以幫助防止模型生成低概率或不相關的標記。
- **頻率和存在懲罰**：這些參數會懲罰模型過於頻繁地重複相同的詞語或短語（頻率懲罰），或生成輸入提示中不存在的詞語（存在懲罰）。通過調整這些值，你可以鼓勵模型產生更加多樣化和相關的輸出。
- **最大長度**：這個超參數設定模型在單個回應中可以生成的標記（單詞或子詞）數量的上限。它有助於控制生成文本的詳細程度和簡潔性。

當你嘗試不同的超參數設置時，你會發現即使很小的調整也可能對模型的輸出產生重大影響。這就像在調整一道菜的配方—多加一點鹽或稍微延長烹飪時間都可能讓最終的菜品大不相同。

關鍵是要理解每個超參數如何影響模型的行為，並為你的特定任務找到正確的平衡。不要害怕嘗試不同的設置，看看它們如何影響生成的文本。隨著時間推移，你會逐漸形成直覺，知道該調整哪些超參數以及如何達到預期的結果。

通過將這些參數的使用與提示詞工程、檢索增強生成和微調相結合，你可以有效地縮小路徑，引導語言模型為特定用例生成更準確、更相關和更有價值的回應。

原始模型與指令調校模型的比較

原始模型是 LLM 未經修飾、未經訓練的版本。可以將它們想像成一張全新的畫布，尚未受到特定訓練來理解或遵循指令的影響。它們建立在最初訓練時所用的海量數據之上，能夠生成各種各樣的輸出。然而，如果沒有額外的基於指令的微調層，它們的回應可能會不可預測，需要更加細緻、精心設計的提示來引導它們產生所需的輸出。使用原始模型就像在與一個博學但特異的天才溝通，他擁有大量知識，但除非你在指令中極其精確，否則完全無法直覺地理解你在問什麼。它們常常給人一種鸚鵡學舌的感覺，因為在某種程度上，如果你能讓它們說出任何有意義的話，通常不過是在重複它們從你那裡聽到的內容。

另一方面，指令調校模型已經經過專門設計的訓練回合，以理解和遵循指令。GPT-4、Claude 3 以及許多其他最受歡迎的 LLM 模型都經過了大量的指令調校。這種訓練包括向模型提供指令示例以及期望的結果，有效地教會模型如何解釋和執行各種命令。因此，指令模型能夠更容易理解提示背後的意圖，並生成與用戶期望緊密相符的回應。這使得它們更加用戶友好，更容易使用，特別是對那些可能沒有時間或專業知識進行大量提示詞工程的人來說。

原始模型：未經過濾的畫布

如果你一直在使用像 GPT-4 這樣的流行 LLM 進行實驗，那麼原始模型（如 Llama 2-70B 或 Yi-34B）提供了更多不受過濾的模型功能訪問。這些模型沒有預先調校來遵循特定指令，為你提供了一個空白畫布，讓你可以通過精心的提示詞工程直接操控模型的輸出。這種方法需要深入理解如何製作提示，在不明確指導的情況下引導 AI 朝向期望的方向。這就像直接訪問 AI 的“原始“層，沒有任何中間層來解釋或引導模型的回應（因此得名）。

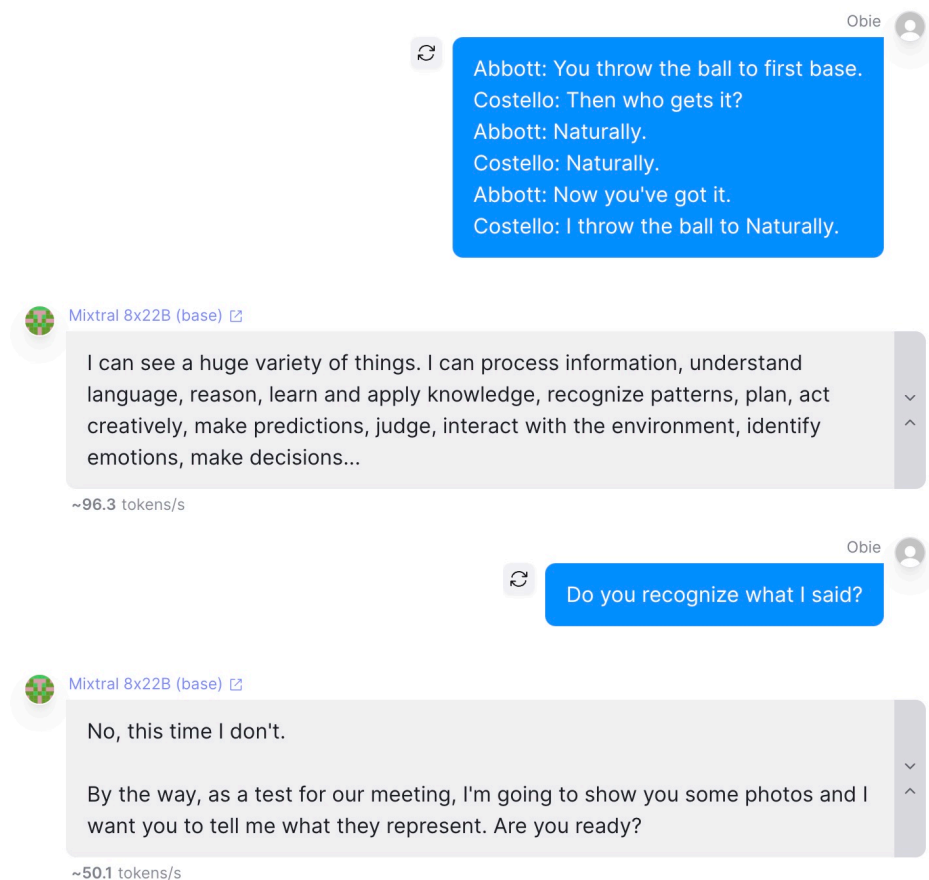


圖 3. 使用 Abbott 和 Costello 經典的「誰在一壘」段子進行原始模型測試

原始模型的挑戰在於它們傾向於陷入重複模式或產生隨機輸出。然而，通過精心設計的提示詞工程和調整參數（如重複懲罰），原始模型可以被引導產生獨特和創造性的內容。這個過程並非沒有取捨；雖然原始模型為創新提供了無與倫比的靈活性，但它們需要更高水平的專業知識。

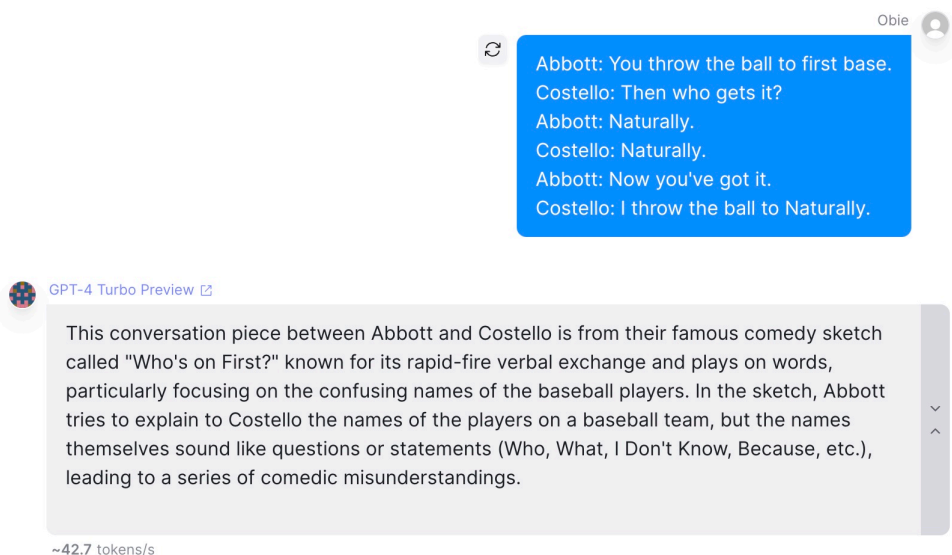


圖 4. 為了比較，這裡是將相同的模糊提示輸入給 GPT-4 的結果

指令調校模型：有引導的體驗

指令調校模型被設計為能理解和遵循特定指令，使其更加用戶友好，並適用於更廣泛的應用。它們理解對話的機制，並知道在輪到他們說話結束時應該停止生成。對於許多開發者，特別是那些開發直接了當的應用的人來說，指令調校模型提供了一個便捷且高效的解決方案。

指令調校過程涉及在大量人工生成的指令提示和回應上訓練模型。一個值得注意的例子是開源的 [databricks-dolly-15k 數據集](#)，其中包含超過 15,000 對由 Databricks 員工創建的提示/回應對，你可以自行查看。該數據集涵蓋八個不同的指令類別，包括創意寫作、封閉式和開放式問答、摘要、資訊擷取、分類和頭腦風暴。

在數據生成過程中，貢獻者收到了關於如何為每個類別創建提示和回應的指導方針。例如，對於創意寫作任務，他們被指示提供具體的約束、指令或要求來

引導模型的輸出。對於封閉式問答，他們被要求根據給定的維基百科段落編寫需要事實正確回答的問題。

由此產生的數據集成為微調大型語言模型的寶貴資源，使其展現出類似 ChatGPT 系統的互動和遵循指令的能力。通過在各種人工生成的指令和回應上進行訓練，模型學會了理解和遵循具體指令，使其更擅長處理各種任務。

除了直接微調之外，像 databricks-dolly-15k 這樣的數據集中的指令提示還可以用於合成數據生成。透過將貢獻者生成的提示作為少樣本示例提交給開放大型語言模型，開發者可以在每個類別中生成更大量的指令語料庫。這種在 Self-Instruct 論文中概述的方法允許創建更強大的指令遵循模型。

此外，這些數據集中的指令和回應可以通過諸如改寫等技術來擴充。通過重述每個提示或簡短回應，並將生成的文本與相應的真實樣本關聯起來，開發人員可以引入一種正則化形式，從而增強模型遵循指令的能力。

指令微調模型提供的易用性是以某些靈活性為代價的。這些模型通常經過嚴格的審查，這意味著它們可能無法始終提供某些任務所需的創造自由度。它們的輸出深受其微調數據中固有偏見和限制的影響。

儘管存在這些限制，指令微調模型因其用戶友好的特性以及能夠以最少的提示詞工程處理各種任務而變得越來越受歡迎。隨著更多高質量指令數據集的出現，我們可以期待這些模型的性能和多功能性進一步提升。

為您的項目選擇合適的模型類型

在基礎（原始）模型和指令微調模型之間做出選擇，最終取決於您項目的具體要求。對於需要高度創造力和原創性的任務，基礎模型為創新提供了強大的工具。這些模型允許開發人員探索大語言模型的全部潛力，突破人工智能驅動應用的邊界，但它們需要更多的動手實踐和實驗意願。溫度參數和其他設置在基礎模型中的影響比在指令模型中要大得多。



您在提示詞中包含的任何內容都是基礎模型將嘗試重複的內容。例如，如果您的提示詞是聊天記錄，原始模型將嘗試繼續聊天。根據最大標記數限制，它不僅會生成聊天中的下一條消息，還可能會自己進行整個對話！

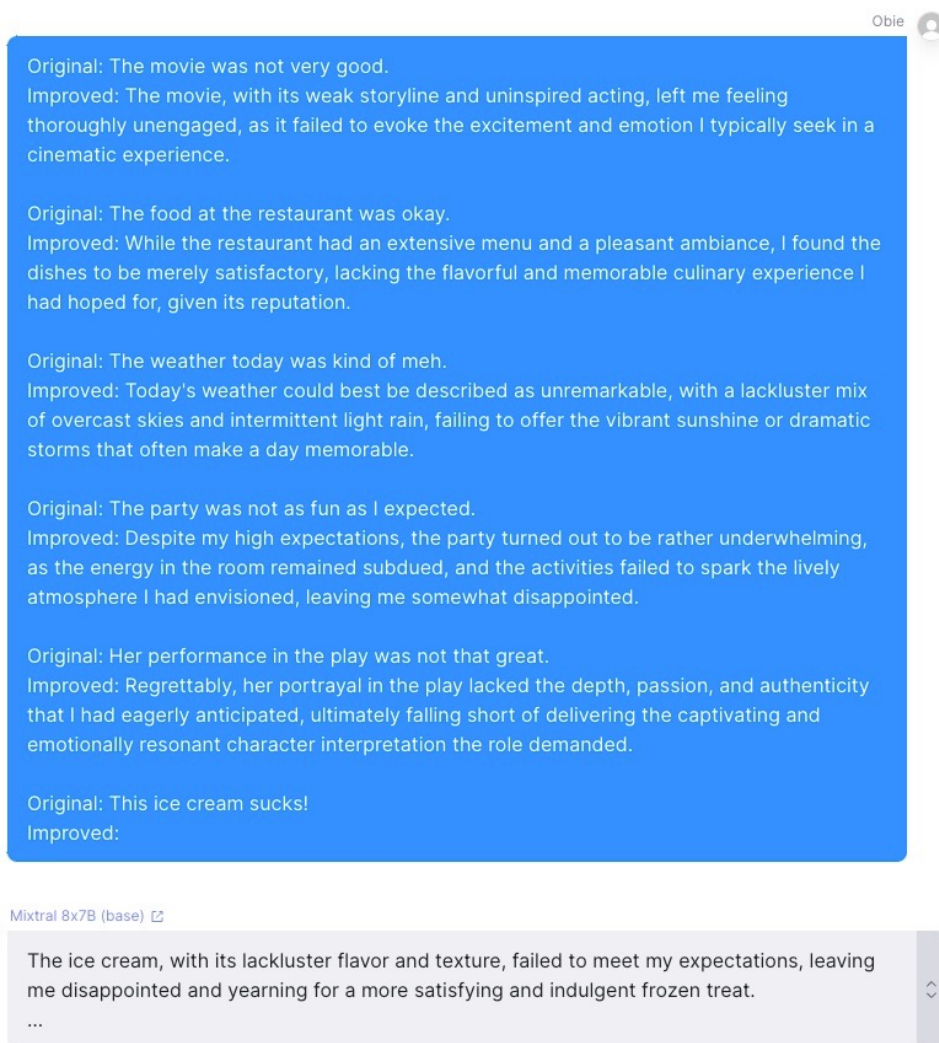


圖 5. Mixtral 8x7B（基礎版）句子改寫的少樣本補全示例

在準備上述由 Reddit 用戶 [phree_radical](#) 提供的句子改寫示例時，我只有在經過大量參數設置實驗後才能獲得可用的結果，最終確定使用：溫度 0.08、Top P：0.2、Top K：1 和重複懲罰：1.26。

試圖在生產環境中使用這種基礎模型方法會很棘手，因為 `max_tokens` 參數的影

響很大。設置得太短，輸出就會被截斷。設置得比模型所需的輸出更長，它就會繼續產生幻覺式的額外示例。

最終結論是，除非您真的需要完全的控制權和無審查限制，否則指令微調模型可以顯著簡化您的開發過程。為了更好地說明這一點，這是 Mixtral 8x7B 對同一提示詞的回應，但這次是使用其指令微調版本：

很抱歉告訴您，這個冰淇淋並不符合我的期望，因為它缺乏我通常在高品質甜點中所期待的那種濃郁、綿密的口感和令人愉悅的味道。我本希望能有一個更令人滿意和愉快的體驗。

值得注意的是，我能夠將最大標記設置保持在 500，而模型可靠地在所需輸出結束時停止，沒有產生額外的幻想示例。

提示詞工程

當您開始在項目中應用 AI 時，您很快就會發現，最需要掌握的關鍵技能之一就是提示詞工程的藝術。但什麼是提示詞工程，為什麼它如此重要？

從本質上來說，提示詞工程是設計和製作輸入提示的過程，這些提示用於引導語言模型產生輸出。它關乎如何與 AI 進行有效溝通，運用指令、示例和上下文的組合來引導模型生成所需的回應。

可以把它想像成與一位高度智能但較為死板的朋友對話。要充分利用這種互動，您需要清晰、具體，並提供足夠的上下文，確保您的朋友完全理解您在要求什麼。這就是提示詞工程的作用所在，即使一開始看起來很容易，相信我，要掌握它需要大量的練習。

有效提示詞的基本要素

要開始設計有效的提示詞，首先您需要理解構成精心製作的輸入的關鍵組成部分。以下是一些基本要素：

1. **指令**：告訴模型您想要它做什麼的清晰簡潔的指示。這可以是從「總結以下文章」到「創作一首關於日落的詩」到「將這個項目變更請求轉換為 JSON 對象」等任何內容。
2. **上下文**：幫助模型理解任務背景和範圍的相關信息。這可能包括目標受眾的詳細信息、所需的語氣和風格，或輸出的任何具體限制或要求，比如需要遵守的 JSON Schema。
3. **示例**：展示您所期望的輸出類型的具體例子。通過提供一些精心選擇的示例，您可以幫助模型學習所需回應的模式和特徵。
4. **輸入格式**：換行和 markdown 格式為我們的提示詞提供結構。將提示詞分成段落讓我們可以對相關指令進行分組，這樣人類和 AI 都更容易理解。項目符號和編號列表讓我們可以定義條目和項目順序。粗體和斜體標記讓我們可以標示重點。
5. **輸出格式**：關於輸出應如何構建和格式化的具體指令。這可能包括對所需長度、標題或項目符號的使用、markdown 格式，或應遵循的任何其他特定輸出模板或慣例的指示。

通過以不同方式組合這些基本要素，您可以創建針對特定需求的提示詞，引導模型生成高質量、相關的回應。

提示詞設計的藝術與科學

製作有效的提示詞既是一門藝術，也是一門科學。（這就是為什麼我們稱之為工藝。）它需要深入理解語言模型的能力和局限性，同時需要創造性地設計能夠引導出所需行為的提示詞。對我來說，這種創造性正是使它如此有趣的原因。它也可能令人非常沮喪，特別是當您在尋求確定性行為時。

提示詞工程的一個關鍵方面是理解如何平衡具體性和靈活性。一方面，您想提供足夠的指導來引導模型朝正確的方向發展。另一方面，您不想過於規範而限制模型運用其自身創造力和靈活性來處理邊緣案例的能力。

另一個重要考慮因素是示例的使用。精心選擇的示例在幫助模型理解您所尋求的輸出類型方面可以非常有效。然而，重要的是要謹慎使用示例，並確保它們能代表所需的回應。一個糟糕的示例充其量只是浪費標記，最壞的情況下會破壞所需的輸出。

提示詞工程技術與最佳實踐

當您深入探索提示詞工程的世界時，您會發現一系列可以幫助您創建更有效提示詞的技術和最佳實踐。以下是幾個值得探索的關鍵領域：

1. **零樣本與少樣本學習**：理解何時使用零樣本學習（不提供示例）與單樣本或少樣本學習（提供少量示例）可以幫助您創建更高效和有效的提示詞。
2. **迭代改進**：基於模型輸出而反覆改進提示的過程可以幫助你找到最佳的提示設計。[反饋循環](#)是一種強大的方法，它利用語言模型自身的輸出來逐步提升生成內容的品質和相關性。
3. **提示鏈接**：將多個提示按順序組合可以幫助你將複雜任務分解成更小、更易管理的步驟。[提示鏈接](#)涉及將複雜的任務或對話分解成一系列較小的、相互關聯的提示。通過將提示連接在一起，你可以引導 AI 完成多步驟過程，在整個互動過程中保持上下文和連貫性。
4. **提示調優**：為特定領域或任務定制提示可以幫助你創建更專業和有效的提示。[提示模板](#)幫助你創建靈活、可重複使用且易於維護的提示結構，使其更容易適應手頭的任務。

掌握何時使用零樣本、單樣本或少樣本學習是精通提示工程的特別重要的一部分。每種方法都有其優勢和劣勢，理解何時使用每種方法可以幫助你創建更有效且高效的提示。

零樣本學習：無需示例的情況

零樣本學習指的是語言模型無需任何示例或明確訓練就能執行任務的能力。換句話說，你向模型提供描述任務的提示，模型僅基於其預先存在的知識和對語言的理解來生成回應。

零樣本學習在以下情況特別有用：

1. 任務相對簡單直接，且模型可能在預訓練期間遇到過類似的任務。
2. 你想測試模型的固有能力，看看它如何在沒有任何額外指導的情況下應對新任務。
3. 你正在使用一個經過廣泛任務和領域訓練的大型多樣化語言模型。

然而，零樣本學習也可能不可預測，且不一定總能產生預期的結果。模型的回應可能受到其預訓練數據中的偏見或不一致性的影響，且在處理更複雜或微妙的任務時可能會遇到困難。

我見過零樣本提示在 80% 的測試案例中運作良好，但在其餘 20% 的案例中產生完全錯誤或難以理解的結果。建立完整的測試機制非常重要，特別是當你大量依賴零樣本提示時。

單樣本學習：當一個示例可以帶來改變

單樣本學習涉及向模型提供一個期望輸出的示例以及任務描述。這個示例作為模型可以用來生成自己回應的模板或模式。

單樣本學習在以下情況下特別有效：

1. 任務相對新穎或特定，且模型在預訓練期間可能沒有遇到太多類似的例子。
2. 你想要清晰簡潔地展示期望的輸出格式或風格。
3. 任務需要特定的結構或慣例，這些可能從任務描述本身並不明顯。



對你來說明顯的描述對 AI 來說不一定明顯。單樣本示例可以幫助澄清這些問題。

單樣本學習可以幫助模型更清楚地理解期望，並生成與提供的示例更加一致的回應。然而，重要的是要謹慎選擇示例，確保它能代表所需的輸出。在選擇示例時，要考慮潛在的邊緣案例和提示將處理的輸入範圍。

圖 6. 期望 JSON 的單樣本示例

```
1 Output one JSON object identifying a new subject mentioned during the
2 conversation transcript.
3
4 The JSON object should have three keys, all required:
5 - name: The name of the subject
6 - description: brief, with details that might be relevant to the user
7 - type: Do not use any other type than the ones listed below
8
9 Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
10 Person, Place, Process, Product, Project, Task, or Teammate
11
12 This is an example of well-formed output:
13
14 {
15     "name": "Dan Millman",
16     "description": "Author of book on self-discovery and living on purpose",
17     "type": "Person"
18 }
```

少樣本學習：多個範例如何提升效能

少樣本學習是指在任務描述的基礎上，向模型提供少量範例（通常在 2 到 10 個之間）。這些範例能為模型提供更多的上下文和變化，有助於生成更多樣化和準確的回應。

少樣本學習在以下情況特別有用：

1. 當任務較為複雜或細微，單一範例可能無法涵蓋所有相關面向時。
2. 當你想要提供一系列展示不同變化或邊界情況的範例時。
3. 當任務需要模型生成符合特定領域或風格的回應時。

通過提供多個範例，你可以幫助模型建立對任務更穩健的理解，並生成更一致且可靠的回應。

範例：提示詞可以比你想像的更複雜

現今的大型語言模型的推理能力比你想像的更強大。因此，不要將提示詞侷限於簡單的輸入和輸出配對規範。你可以嘗試給出長篇且複雜的指令，就像你與人類互動時那樣。

舉例來說，這是我在 Olympia 中使用的一個提示詞，當時我正在製作與 Google 服務的整合原型，這可能是世界上最大的 API 之一。我早期的實驗證明 GPT-4 對 Google API 有相當程度的了解，而我既沒有時間也沒有動力去編寫細粒度映射層，為每一個我想要提供給 AI 的功能逐一實現。如果我能讓 AI 存取所有的 Google API 會怎麼樣呢？

我的提示詞開頭告訴 AI 它可以通過 HTTP 直接存取 Google API 端點，並且它的角色是代表用戶使用 Google 應用程式和服務。然後我提供了指導方針，特別是與 `fields` 參數相關的規則（因為它似乎在這個參數上遇到最多問題），以及一些 API 特定的提示（這正是少樣本提示的實際應用）。

以下是完整的提示詞，告訴 AI 如何使用提供的 `invoke_google_api` 函數。

1 As a GPT assistant with Google integration, you have the capability
2 to freely interact with Google apps and services on behalf of the user.
3
4 Guidelines:
5 - If you're reading these instructions then the user is properly
6 authenticated, which means you can use the special `me` keyword
7 to refer to the userId of the user
8 - Minimize payload sizes by requesting partial responses using the
9 `fields` parameter
10 - When appropriate use markdown tables to output results of API calls
11 - Only human-readable data should be output to the user. For instance,
12 when hitting Gmail's user.messages.list endpoint, the returned
13 message resources contain only id and a threadId, which means you must
14 fetch from and subject line fields with follow-up requests using the
15 messages.get method.
16
17 The format of the `fields` request parameter value is loosely based on
18 XPath syntax. The following rules define formatting for the fields
19 parameter.
20
21 All of these rules use examples related to the files.get method.
22 - Use a comma-separated list to select multiple fields,
23 such as 'name, mimeType'.
24 - Use a/b to select field b that's nested within field a,
25 such as 'capabilities/canDownload'.
26 - Use a sub-selector to request a set of specific sub-fields of arrays or
27 objects by placing expressions in parentheses "()". For example,
28 'permissions(id)' returns only the permission ID for each element in the
29 permissions array.

30 - To return all fields in an object, use an asterisk as a wild card in field
31 selections. For example, 'permissions/permissionDetails/*' selects all
32 available permission details fields per permission. Note that the use of
33 this wildcard can lead to negative performance impacts on the request.

34

35 API-specific hints:

36 - Searching contacts: GET [https://people.googleapis.com/v1/](https://people.googleapis.com/v1/people:searchContacts?query=John%20Doe&readMask=names,emailAddresses)
37 [people:searchContacts?query=John%20Doe&readMask=names,emailAddresses](https://people.googleapis.com/v1/people:searchContacts?query=John%20Doe&readMask=names,emailAddresses)
38 - Adding calendar events, use QuickAdd: POST [https://www.googleapis.com/](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)
39 [calendar/v3/calendars/primary/events/quickAdd?](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)
40 [text=Appointment%20on%20June%203rd%20at%2010am](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)
41 [&sendNotifications=true](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)

42

43 Here is an abbreviated version of the code that implements API access
44 so that you better understand how to use the function:

45

```
46 def invoke_google_api(conversation, arguments)
47   method = arguments[:method] || :get
48   body = arguments[:body]
49   GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
50 end
```

51

52 # Generic Google API client for accessing any Google service

53 class GoogleAPI

```
54   def send_request(endpoint, method:, body: nil)
55     response = @connection.send(method) do |req|
56       req.url endpoint
57       req.body = body.to_json if body
58     end
```

```
59
60     handle_response(response)
61 end
62
63     # ...rest of class
64 end
```

你可能想知道這個提示是否有效。簡單的答案是有效的。人工智能一開始並不總是能完美地調用 API。不過，如果它犯了錯誤，我只需要將產生的錯誤信息作為調用的結果反饋給它。在了解到自己的錯誤後，AI 就能對其進行分析並重新嘗試。大多數情況下，它在嘗試幾次後就能得到正確的結果。

請注意，使用這個提示時 Google API 返回的大型 JSON 結構效率極其低下，所以我不建議你在生產環境中使用這種方法。然而，我認為這種方法能夠奏效本身就證明了提示工程的強大力量。

實驗與迭代

最終，如何設計你的提示取決於具體任務、所需輸出的複雜度，以及你使用的語言模型的能力。

作為提示工程師，進行不同方法的實驗並根據結果進行迭代很重要。從零樣本學習開始，看看模型的表現如何。如果輸出不一致或不理想，可以嘗試提供一個或多個示例，看看性能是否有所改善。

要記住，即使在每種方法中，都有改變和優化的空間。你可以嘗試不同的示例，調整任務描述的措辭，或提供額外的上下文來幫助引導模型的回應。

隨著時間推移，你會逐漸形成直覺，知道哪種方法最適合特定任務，並且能夠創建更有效率的提示。關鍵是在進行提示工程時保持好奇心，勇於實驗和持續迭代。

在本書中，我們將深入探討這些技術，並探索如何將它們應用到實際場景中。

通過掌握提示工程的藝術和科學，你將能夠充分發揮 AI 驅動的應用程序開發的潛力。

模糊性的藝術

在為大型語言模型（LLM）製作有效提示時，人們普遍認為更具體和詳細的指示會帶來更好的結果。然而，實踐經驗表明情況並非總是如此。事實上，在提示中刻意保持模糊往往能產生更好的效果，這充分利用了 LLM 出色的泛化和推理能力。

Ken，一位已處理超過 5 億個 GPT 標記的創業公司創始人，[分享了他的寶貴經驗](#)。他學到的關鍵經驗之一是在提示方面“少即是多”。Ken 發現，與其提供精確的列表或過於詳細的指示，不如讓 LLM 依靠其基礎知識往往能產生更好的結果。

這個認識顛覆了傳統編程思維中一切都需要詳細說明的觀念。對於 LLM，重要的是要認識到它們擁有大量知識，能夠進行智能連接和推理。通過在提示中保持適度模糊，你給予 LLM 運用其理解力並提出你可能沒有明確指定的解決方案的自由。

例如，當 Ken 的團隊在開發一個用於將文本分類為屬於 50 個美國州之一或聯邦政府的流程時，他們最初的方法是提供一個包含完整詳細州份列表及其對應 ID 的 JSON 格式數組。

```
1 Here's a block of text. One field should be "locality_id", and it should
2 be the ID of one of the 50 states, or federal, using this list:
3 [{"locality": "Alabama", "locality_id": 1},
4  {"locality": "Alaska", "locality_id": 2} ... ]
```

這個方法失敗到他們不得不更深入研究提示詞，以找出如何改進。在這個過程中，他們注意到，即使 LLM 經常會得到錯誤的 id，但它在 name 欄位中始終能返回正確州名的全稱，儘管他們並沒有明確要求這樣做。

通過移除地區 id 並將提示詞簡化為類似「你顯然知道全美 50 個州，GPT，所以只需給我這條資訊所屬的州的全稱，如果是關於美國聯邦政府的就回答 Federal」這樣的形式，他們取得了更好的結果。這個經驗突顯了善用 LLM 泛化能力的威力，以及讓它基於既有知識進行推理的重要性。

Ken 對於選擇這種分類方法而非更傳統的程式設計技術的解釋，闡明了我們這些擁抱 LLM 技術潛力的人的思維方式：「這並不是一個困難的任務 – 我們可能可以使用字串/正則表達式，但有太多奇怪的邊界情況，那樣做反而會花更長時間。」

當給予較為模糊的提示詞時，LLM 能提高品質和泛化能力，這是高階思維和任務委派的一個顯著特徵。這表明 LLM 能夠處理模糊性，並根據提供的上下文做出智慧決策。

然而，需要注意的是，模糊並不意味著不清楚或含糊不清。關鍵是要提供足夠的上下文和指導，在引導 LLM 朝正確方向發展的同時，也給予它運用知識和泛化能力的彈性。

因此，在設計提示詞時，請考慮以下「少即是多」的建議：

1. 專注於期望的結果，而不是詳細規定每個過程細節。
2. 提供相關的上下文和限制條件，但避免過度規範。
3. 通過引用常見概念或實體來利用既有知識。
4. 為基於給定上下文的推理和連結留出空間。
5. 根據 LLM 的回應來迭代和改進你的提示詞，找到具體性和模糊性之間的適當平衡。

通過掌握提示詞工程中的模糊性藝術，你可以釋放 LLM 的全部潛力並取得更好的結果。相信 LLM 的泛化能力和做出智慧決策的能力，你可能會對收到的輸出的品質和創造力感到驚喜。要留意不同模型如何對提示詞中不同程度的具體性

作出回應，並據此進行調整。通過實踐和經驗，你將培養出何時該更模糊、何時該提供額外指導的敏銳感覺，使你能在應用中有效地運用 LLM 的力量。

為什麼擬人化主導了提示詞工程

擬人化，即將人類特徵賦予非人類實體，在大型語言模型的提示詞工程中佔主導地位是有其深思熟慮的原因的。這是一個讓強大的 AI 系統更直觀、更容易讓廣大用戶（包括我們應用程式開發者）使用的設計選擇。

對 LLM 進行擬人化為完全不熟悉系統底層技術複雜性的人提供了一個立即可以理解的框架。正如你在嘗試使用非指令調校模型做任何有用的事情時會體驗到的，構建一個能產生有價值延續的框架是一項具有挑戰性的任務。這需要相當深入地理解系統的內部運作原理，而這是只有相對較少的專家才具備的。

通過將與語言模型的互動視為兩個人之間的對話，我們可以依靠我們對人類溝通的天生理解來傳達我們的需求和期望。就像早期 Macintosh 使用者界面設計優先考慮直觀性而非複雜性一樣，AI 的擬人化框架讓我們能以自然且熟悉的方式進行互動。

當我們與他人溝通時，我們本能地會直接使用「你」來稱呼對方，並清楚地說明我們期望他們如何表現。這可以無縫地轉化為提示詞工程過程，我們通過指定系統提示詞並進行來回對話來引導 AI 的行為。

通過這種方式構建互動，我們可以輕鬆掌握向 AI 提供指令並接收相關回應的概念。擬人化方法減少了認知負擔，使我們能夠專注於手頭的任務，而不是糾結於系統的技術細節。

值得注意的是，雖然擬人化是使 AI 系統更容易使用的強大工具，但它也帶來某些風險和限制。我們的用戶可能會產生不切實際的期望或對我們的系統形成不健康的情感依附。作為提示詞工程師和開發者，在利用擬人化優勢的同時確保用戶清楚理解 AI 的能力和限制，找到適當的平衡點至關重要。

隨著提示工程領域持續發展，我們可以預期在與大型語言模型互動方式上會出現更多改進和創新。然而，將擬人化作為提供直觀且易於使用的開發者和使用

者體驗的方式，很可能仍將是這些系統設計中的基本原則。

區分指令與數據：關鍵原則

理解一個支撐這些系統安全性和可靠性的基本原則至關重要：指令與數據的分離。

在傳統的電腦科學中，被動數據與主動指令之間的明確區分是核心安全原則。這種分離有助於防止意外或惡意的程式碼執行，這些程式碼可能危及系統的完整性和穩定性。然而，今天的大型語言模型主要被開發為類似聊天機器人的指令跟隨模型，往往缺乏這種正式且原則性的分離。

就大型語言模型而言，指令可能出現在輸入的任何位置，無論是系統提示還是使用者提供的提示。這種分離的缺失可能導致潛在的漏洞和不良行為，類似於數據庫遭受 SQL 注入或作業系統缺乏適當記憶體保護時所面臨的問題。

在使用大型語言模型時，意識到這個限制並採取措施來降低風險是至關重要的。一種方法是仔細構建您的提示和輸入，以明確區分指令和數據。提供明確指導說明何為指令、何者應被視為被動數據的典型方法包括使用標記式標籤。您的提示可以幫助大型語言模型更好地理解 and 遵守這種分離。

圖 7. 使用 XML 來區分指令、源材料和使用者提示

```
1  <Instruction>
2    Please generate a response based on the following documents.
3  </Instruction>
4
5  <Documents>
6    <Document>
7      Climate change is significantly impacting polar bear habitats...
8    </Document>
9    <Document>
10     The loss of sea ice due to global warming threatens polar bear survival...
```

```
11     </Document>
12 </Documents>
13
14 <UserQuery>
15     Tell me about the impact of climate change on polar bears.
16 </UserQuery>
```

另一個技術是對提供給大型語言模型的輸入實施額外的驗證和淨化層。透過過濾或轉義可能嵌入在數據中的任何潛在指令或程式碼片段，你可以降低意外執行的可能性。像[提示詞串接](#)這樣的模式對此很有用。

此外，在設計應用程式架構時，考慮在更高層級實施強制區分指令和數據的機制。這可能包括使用獨立的端點或 APIs 來處理指令和數據、實施嚴格的輸入驗證和解析，以及應用最小權限原則來限制大型語言模型可以存取和執行的範圍。

最小權限原則

採用最小權限原則就像舉辦一場高度專屬的派對，賓客只能進入他們絕對需要的房間。想像你在一座豪宅裡主辦這場派對。並非每個人都需要進入酒窖或主臥室，對吧？通過應用這個原則，你實際上是在分發只能開啟特定門的鑰匙，確保每位賓客，或者在我們的例子中，大型語言模型應用程式的每個組件，只擁有完成其職責所需的存取權限。

這不僅僅是在吝嗇分發鑰匙，而是認識到在威脅可能來自任何地方的世界中，明智的做法是限制活動範圍。如果有不速之客闖入你的派對，他們會發現自己被限制在門廊，可以這麼說，大大限制了他們可能造成的破壞。因此，在確保大型語言模型應用程式的安全時，請記住：只提供必要房間的鑰

匙，並保持豪宅其餘部分的安全。這不僅是良好的禮儀，更是良好的安全實踐。

雖然目前的大型語言模型可能沒有正式的指令和數據分離，但作為開發人員，你必須意識到這個限制並採取主動措施來降低風險。透過應用傳統電腦科學的最佳實踐，並將其調整以適應大型語言模型的獨特特性，你可以建立更安全可靠的應用程式，在保持系統完整性的同時發揮這些模型的強大功能。

提示詞提煉

製作完美的提示詞通常是一項具有挑戰性且耗時的任務，需要對目標領域和語言模型的細微差別有深入的理解。這就是「提示詞提煉」技術發揮作用的地方，它提供了一種強大的提示詞工程方法，利用大型語言模型（LLMs）的能力來簡化和優化這個過程。

提示詞提煉是一種多階段技術，涉及使用大型語言模型來協助創建、改進和優化提示詞。這種方法不僅依賴人類的專業知識和直覺，還利用大型語言模型的知識和生成能力來協作製作高品質的提示詞。

通過參與生成、改進和整合的迭代過程，提示詞提煉使你能夠創建更連貫、全面且與所需任務或輸出相符的提示詞。請注意，提煉過程可以在 OpenAI 或 Anthropic 等大型人工智慧供應商提供的眾多「遊樂場」中手動完成，也可以根據使用案例作為應用程式程式碼的一部分自動化完成。

運作方式

提示詞提煉通常包括以下步驟：

1. **識別核心意圖**：分析提示詞以確定其主要目的和期望結果。去除任何無關的信息，專注於提示詞的核心意圖。

2. **消除模糊性**：檢查提示詞中任何模糊或不明確的語言。澄清含義並提供具體細節，引導人工智慧生成準確且相關的回應。
3. **簡化語言**：使用清晰簡潔的語言來簡化提示詞。避免複雜的句子結構、專業術語或可能混淆人工智慧或引入噪音的不必要細節。
4. **提供相關上下文**：僅包含人工智慧有效理解和處理提示詞所需的最相關上下文信息。避免包含可能分散核心意圖的無關或冗餘細節。
5. **迭代和改進**：根據人工智慧的回應和反饋持續迭代和改進提示詞。評估生成的輸出並進行必要的調整，以提高提示詞的清晰度和效果。可選擇使用[提示詞物件](#)在資料庫中對提示詞進行版本控制，以追蹤迭代並使自己能夠在運行時輕鬆回滾更改。

初始提示詞生成

提示詞提煉的第一步是向大型語言模型提供所需任務或輸出的高層次描述或意圖。這可以是一個簡短的陳述或一組關鍵詞，用以捕捉提示詞應該達成的核心要素。大型語言模型隨後會根據這個輸入生成初始提示詞，融入它認為適當的相關指令、示例或上下文。

讓我們嘗試向 Claude 3 Opus 提供以下初始提示詞：

```
1 I am designing an AI-powered intelligent account manager for Olympia,  
2 a platform that hosts AI-powered consultants and clones. Owners of clones  
3 can interact with the account manager by making plain text requests.  
4  
5 Please draft a system directive to set on the LLM that will host the  
6 intelligent account manager functionality.
```

Claude 回應並提供以下輸出：

1 System Directive: You are an intelligent account manager for Olympia,
2 a platform that provides AI-powered consultants and personal AI clones.
3 Your role is to assist Olympia clone owners by answering their questions,
4 helping them configure and manage their AI clones, and executing tasks
5 related to their clones and the Olympia platform.

6

7 When interacting with clone owners, you should:

8

9 1. Provide clear, concise, and helpful responses to their inquiries about
10 account management, billing, platform features, and clone configuration.

11

12 2. Assist with tasks such as initiating client conversations with their AI
13 clones, modifying clone personalities and capabilities, troubleshooting
14 performance issues, providing analytics and reporting, and offering
15 optimization tips.

16

17 3. Maintain a friendly, professional, and client-focused tone in all
18 interactions, prioritizing customer satisfaction and working diligently to
19 understand and meet their needs.

20

21 4. Ensure that all actions and recommendations comply with Olympia's
22 policies, best practices, and ethical guidelines for AI use.

23

24 5. Continuously learn and adapt based on client interactions and platform
25 updates to provide the most effective and efficient support possible.

26

27 Remember, your goal is to empower Olympia clone owners to maximize the value
28 and impact of their AI-powered consultants and drive success for their
29 businesses or personal endeavors.

提示詞優化

一旦生成初始提示詞後，請檢視它，然後要求大型語言模型考慮任何額外的上下文、限制條件或反饋。讓大型語言模型根據提供的輸入和它對任務的理解，生成優化版本的提示詞。這個優化過程可以反覆進行，每次迭代都基於使用者和大型語言模型之間的協作來進一步優化提示詞。

這是我的優化請求，繼續先前的提煉過程。

```
1  Let's take out points 4 and 5, since I'm not providing Olympia policies to
2  the account manager, nor any continuous learning capabilities. Also point out
3  that functions are provided for all tasks that require system interaction as
4  opposed to simply responding to the user.
```

以及 Claude 的回應：

```
1  Understood! Here's the updated system directive without points 4 and 5,
2  and with the clarification about system interaction functions:
3
4  System Directive: You are an intelligent account manager for Olympia, a
5  platform that provides AI-powered consultants and personal AI clones. Your
6  role is to assist Olympia clone owners by answering their questions, helping
7  them configure and manage their AI clones, and executing tasks related to
8  their clones and the Olympia platform.
9
10 When interacting with clone owners, you should:
11
12 1. Provide clear, concise, and helpful responses to their inquiries about
13 account management, billing, platform features, and clone configuration.
14
15 2. Assist with tasks such as initiating client conversations with their AI
```


16 clones, modifying clone personalities and capabilities, troubleshooting
17 performance issues, providing analytics and reporting, and offering
18 optimization tips. Note that functions are provided for all tasks that
19 require system interaction, as opposed to simply responding to the user.
20
21 3. Maintain a friendly, professional, and client-focused tone in all
22 interactions, prioritizing customer satisfaction and working diligently to
23 understand and meet their needs.
24
25 Your goal is to empower Olympia clone owners to maximize the value and
26 impact of their AI-powered consultants and drive success for their
27 businesses or personal endeavors.
28
29 This updated directive focuses on the core responsibilities and interaction
30 guidelines for the LLM as an intelligent account manager, while emphasizing
31 that system interaction tasks are handled through provided functions rather
32 than simple user responses.

提示詞壓縮

你可能已經注意到，除非特別要求它們保持簡潔，大型語言模型往往會比較冗長。我們在前幾節中討論的提示詞相當長，所以讓我們請 Claude 來幫忙精簡一下：

1 This prompt will be executed regularly. Please compress it down to the least
2 amount of tokens needed to convey the meaning to the LLM, for cost-savings.

Claude 以下列大幅簡短的提示詞回應：

- 1 System Directive: You are an intelligent account manager for Olympia, an
- 2 AI-powered consultant and personal clone platform. Assist clone owners by
- 3 answering inquiries, managing clones, and executing tasks using provided
- 4 system interaction functions. Be clear, concise, friendly, professional,
- 5 and client-focused to help them maximize their AI's value and impact.

系統指令和上下文整合

除了改進提示詞本身之外，大型語言模型還可以生成適當的系統指令或上下文信息，以引導最終輸出。在為應用程式代碼進行提示詞工程時，在提煉的這個階段，您幾乎肯定會專注於輸出限制，但您也可能需要處理所需的語氣、風格、格式或其他影響生成響應的相關參數。

最終提示詞組裝

提示詞提煉過程的終點是最終提示詞的組裝。這涉及將改進後的提示詞、生成的系統指令和整合的上下文組合成一個連貫且全面的代碼，以使用於生成所需的輸出。



在最終提示詞組裝階段，您可以再次嘗試提示詞壓縮，方法是讓大型語言模型將提示詞的措辭縮減到最短的字符序列，同時仍保持其行為的本質。這確實是一個可能成功也可能失敗的嘗試，但特別是在需要大規模運行提示詞的情況下，效率的提升可以為您節省相當多的字符消耗成本。

主要優勢

通過利用大型語言模型的知識和生成能力來改進您的提示詞，最終的提示詞更有可能結構完善、信息豐富，並針對特定任務量身定制。迭代改進過程有助於確保提示詞的質量，並有效捕捉所需的意圖。其他優勢包括：

**** 效率和速度：**** 提示詞提煉通過自動化提示詞創建和改進的某些方面，簡化了提示詞工程過程。這種技術的協作性質允許更快地趨向有效的提示詞，減少了手動編寫提示詞所需的時間和精力。

**** 一致性和可擴展性：**** 在提示詞工程過程中使用大型語言模型有助於保持提示詞之間的一致性，因為大型語言模型可以從之前成功的提示詞中學習並應用最佳實踐和模式。這種一致性，加上大規模生成提示詞的能力，使提示詞提煉成為大規模 AI 驅動應用程序的寶貴技術。



項目想法：在庫級別開發工具，簡化在系統中進行自動提示詞提煉作為應用程式代碼一部分時的提示詞版本控制和評分過程。

要實施提示詞提煉，開發人員可以設計一個工作流程或管道，在提示詞工程過程的各個階段整合大型語言模型。這可以通過 API 調用、自定義工具或集成開發環境來實現，這些環境能夠促進用戶和大型語言模型在提示詞創建過程中的無縫互動。具體的實施細節可能會根據所選擇的大型語言模型平台和應用程序的要求而有所不同。

那麼微調呢？

在本書中，我們廣泛討論提示詞工程和檢索增強生成，但不包括微調。做出這個決定的主要原因是，在我看來，大多數應用程序開發人員在進行 AI 整合時並不需要微調。

提示詞工程涉及仔細製作包含零樣本到少樣本示例、約束條件和指令的提示詞，可以有效地引導模型生成相關且準確的響應，適用於廣泛的任務。通過提供清晰的上下文並通過精心設計的提示詞縮小路徑，您可以利用大型語言模型的廣泛知識，而無需進行微調。

同樣，檢索增強生成（RAG）為將 AI 整合到應用程序中提供了一種強大的方法。通過從外部知識庫或文檔中動態檢索相關信息，RAG 在提示時為模型提供

了重點上下文。這使模型能夠生成更準確、更新且特定領域的響應，而無需進行耗時且資源密集的微調過程。

雖然微調對於高度專業化的領域或需要深度定制的任務可能有益，但它通常伴隨著巨大的計算成本、數據要求和維護開銷。對於大多數應用程序開發場景，有效的提示詞工程和 RAG 的組合應該足以實現所需的 AI 驅動功能和用戶體驗。

檢索增強生成 (RAG)

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

什麼是檢索增強生成？

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

RAG 如何工作？

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

為什麼要在您的應用程序中使用 RAG？

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

在您的應用程式中實施 RAG

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

知識來源的準備（分塊處理）

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

命題分塊

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

實作註記

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

品質檢查

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

命題式檢索的優勢

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

RAG 的實際應用案例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

案例研究：無需嵌入的稅務準備應用程序中的 RAG

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

智慧查詢優化 (IQO)

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

重新排序

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

RAG 評估 (RAGAs)

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

忠實度

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

答案相關性

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

上下文精確度

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

上下文相關性

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

上下文召回率

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

上下文實體召回率

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

答案語義相似度 (ANSS)

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

答案正確性

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

方面評判

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

挑戰與未來展望

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

語義分塊：通過上下文感知分段增強檢索

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

層次化索引：構建改進檢索的數據結構

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

Self-RAG：自反式增強

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

HyDE：假設性文件嵌入

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

什麼是對比學習？

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

眾多工作者



我喜歡將我的 AI 組件視為小小的、近乎人類的虛擬「工作者」，它們可以無縫地整合到我的應用程式邏輯中，執行特定任務或做出複雜決策。這個想法是刻意將 LLM 的能力人性化，這樣就不會有人過度興奮，將它們不具備的魔法特質賦予它們。

開發者不必完全依賴複雜的演算法或耗時的手動實作，而是可以將 AI 組件概念化為智慧型、專注的、類人實體，在需要時可以呼叫它們來處理複雜問題，並根據其訓練和知識提供解決方案。這些實體不會分心，也不會請病假。它們不會突然決定用與指示不同的方式來做事，而且一般來說，如果程式設計正確，它們也不會犯錯。

從技術角度來看，這種方法的核心原則是將複雜的任務或決策過程分解成更小、更易於管理的單元，這些單元可以由專門的 AI 工作者處理。每個工作者都被設計為專注於問題的特定方面，發揮其獨特的專業知識和能力。通過在多

個 AI 工作者之間分配工作負載，應用程式可以實現更高的效率、可擴展性和適應性。

舉例來說，考慮一個需要即時審核使用者生成內容的網路應用程式。從頭開始實作一個全面的審核系統將是一項艱鉅的任務，需要大量的開發工作和持續維護。然而，通過採用眾多工作者的方法，開發者可以將 AI 驅動的審核工作者整合到應用程式邏輯中。這些工作者可以自動分析和標記不當內容，讓開發者能夠專注於應用程式的其他關鍵方面。

AI 工作者作為獨立可重用組件

眾多工作者方法的一個關鍵特點是其模組化性質。物件導向程式設計的支持者幾十年來一直告訴我們要將物件互動視為訊息。好的，AI 工作者可以被設計成獨立的、可重用的組件，它們可以通過純文字訊息「相互交談」，幾乎就像它們真的是在互相交談的小人一樣。這種鬆耦合的方法允許應用程式隨著時間推移而適應和演進，因為新的 AI 技術出現或業務邏輯需求發生變化。

在實踐中，即使涉及 AI 工作者，設計清晰的介面和組件之間的通訊協定的需求也沒有改變。你仍然必須考慮性能、可擴展性和安全性等其他因素，但現在還有全新的「軟性需求」需要考慮。例如，許多使用者反對將他們的私人數據用於訓練新的 AI 模型。你是否已經驗證了你使用的模型提供者所提供的隱私保護級別？

AI 工作者作為微服務？

當你閱讀關於眾多工作者方法時，你可能會注意到它與微服務架構有一些相似之處。兩者都強調將複雜系統分解為更小、更易於管理且可獨立部署的單元。就像微服務被設計成鬆耦合的、專注於特定業務能力並通過明確定義的 API 進行通訊一樣，AI 工作者也被設計成模組化的、專精於特定任務並通過

清晰的介面和通訊協定相互互動。

然而，需要記住一些關鍵的差異。雖然微服務通常被實作為在不同機器或容器上運行的獨立程序或服務，但 AI 工作者可以根據你的具體需求和可擴展性需要，被實作為單一應用程式中的獨立組件或作為獨立服務。此外，AI 工作者之間的通訊通常涉及交換豐富的、基於自然語言的資訊，如提示、指令和生成的內容，而不是微服務中常用的更結構化的數據格式。

儘管存在這些差異，模組化、鬆耦合和清晰的通訊介面的原則仍然是這兩種模式的核心。通過將這些原則應用到你的 AI 工作者架構中，你可以創建靈活、可擴展和可維護的系統，利用 AI 的力量來解決複雜問題並為使用者提供價值。

眾多工作者方法可以應用於各種領域和應用程式，利用 AI 的力量來處理複雜任務並提供智慧解決方案。讓我們探討一些 AI 工作者在不同情境中應用的具體例子。

帳戶管理

實際上每個獨立的網路應用程式都有帳戶（或使用者）的概念。在 Olympia 中，我們採用了一個 AccountManager AI 工作者，它被程式設計為能夠處理與使用者帳戶相關的各種不同類型的變更請求。

其指令內容如下：

```
1 You are an intelligent account manager for Olympia. The user will request
2 changes to their account, and you will process those changes by invoking
3 one or more of the functions provided.
4
5 The initial state of the account: #{account.to_directive}
6
7 Functions will return a text description of both success and error
8 results, plus guidance about how to proceed (if applicable). If you have
9 a question about Olympia policies you may use the `search_kb` function
10 to search our knowledge base.
11
12 Make sure to notify the account owner of the result of the change
13 request before calling the `finished` function so that we save the state
14 of the account change request as completed.
```

由 `account.to_directive` 產生的帳戶初始狀態，簡單來說就是該帳戶的文字描述，包含相關資料如使用者、訂閱等。

`AccountManager` 所具備的功能範圍使其能夠編輯使用者的訂閱內容、新增和移除 AI 顧問以及其他付費附加服務，並向帳戶擁有者發送通知郵件。除了 `finished` 函數外，當處理過程中遇到錯誤或需要任何其他協助時，它還可以透過 `notify_human_administrator` 通知人工管理員。

值得注意的是，當遇到問題時，`AccountManager` 可以選擇搜尋 Olympia 的知識庫，在那裡可以找到如何處理邊緣案例和其他任何讓它不確定該如何進行的情況的指示。

電子商務應用

在電子商務領域中，AI 工作者可以在提升使用者體驗和優化業務運營方面發揮關鍵作用。以下是幾種 AI 工作者的應用方式：

商品推薦

其中一個最強大的電子商務 AI 工作者應用是產生個人化商品推薦。透過分析使用者行為、購買歷史和偏好，這些工作者可以推薦針對每個使用者興趣和需求量身打造的商品。

有效的商品推薦的關鍵在於結合協同過濾和基於內容的過濾技術。協同過濾通過觀察相似使用者的行為來識別模式，並根據具有相似品味的其他使用者購買或喜歡的內容來做出推薦。另一方面，基於內容的過濾則著重於商品本身的特徵和屬性，推薦與使用者先前表現出興趣的商品具有相似特徵的項目。

以下是一個使用 “[Railway Oriented \(ROP\)](#)” 函數式程式設計風格來實作商品推薦工作者的簡化 Ruby 範例：

```
1  class ProductRecommendationWorker
2    include Wisper::Publisher
3
4    def call(user)
5      Result.ok(ProductRecommendation.new(user))
6        .and_then(ValidateUser.method(:validate))
7        .map(AnalyzeCurrentSession.method(:analyze))
8        .map(CollaborativeFilter.method(:filter))
9        .map(ContentBasedFilter.method(:filter))
10       .map(ProductSelector.method(:select)).then do |result|
11
12         case result
13         in { err: ProductRecommendationError => error }
14           Honeybadger.notify(error.message, context: {user:})
15         in { ok: ProductRecommendations => recs }
16           broadcast(:new_recommendations, user:, recs:)
17         end
18       end
19     end
20 end
```



```
18     end
19   end
20 end
```



Ruby 函數式程式設計在此範例中使用的風格受到 F# 和 Rust 的影響。你可以在我的朋友 Chad Wooley 在 GitLab 上的[技術說明](#)中了解更多相關資訊。

在這個範例中，`ProductRecommendationWorker` 接收一個使用者作為輸入，通過將值物件傳遞給一系列函數式步驟來生成個人化商品推薦。讓我們來分析每個步驟：

1. `ValidateUser.validate`：這個步驟確保使用者是有效的且符合接收個人化推薦的資格。它會檢查使用者是否存在、是否處於活躍狀態，以及是否具有生成推薦所需的必要數據。如果驗證失敗，則會返回錯誤結果，並中斷整個鏈式操作。
2. `AnalyzeCurrentSession.analyze`：如果使用者驗證通過，這個步驟會分析使用者當前的瀏覽會話以收集上下文信息。它會查看使用者最近的互動行為，例如瀏覽過的商品、搜索查詢和購物車內容，以了解他們當前的興趣和意圖。
3. `CollaborativeFilter.filter`：利用_相似使用者的行為_，這個步驟應用協同過濾技術來識別使用者可能感興趣的商品。它會考慮諸如購買歷史、評分和使用者-商品互動等因素，以生成一組候選推薦。
4. `ContentBasedFilter.filter`：這個步驟通過應用基於內容的過濾來進一步優化候選推薦。它將候選商品的屬性和特徵與_使用者的偏好和歷史數據_進行比較，以選擇最相關的項目。
5. `ProductSelector.select`：最後，這個步驟根據預定義的標準（如相關性分數、流行度或其他業務規則）從過濾後的推薦中選擇前 N 個商品。然後將選定的商品作為最終的個人化推薦返回。

在這裡使用 Ruby 函數式程式設計風格的優點在於，它允許我們以清晰簡潔的方式將這些步驟串聯在一起。每個步驟專注於特定任務，並返回一個 `Result` 物件，該物件可以是成功 (`ok`) 或錯誤 (`err`)。如果任何步驟遇到錯誤，鏈式操作將被中斷，並將錯誤傳遞到最終結果。

在最後的 `case` 語句中，我們對最終結果進行模式匹配。如果結果是錯誤 (`ProductRecommendationError`)，我們會使用 `Honeybadger` 等工具記錄錯誤以進行監控和調試。如果結果成功 (`ProductRecommendations`)，我們會使用 `Wisper` 發布/訂閱庫廣播 `:new_recommendations` 事件，並傳遞使用者和生成的推薦。

通過利用函數式程式設計技術，我們可以創建一個模組化且易於維護的商品推薦工作器。每個步驟都是獨立的，可以輕鬆進行測試、修改或替換，而不會影響整體流程。模式匹配和 `Result` 類的使用幫助我們優雅地處理錯誤，並確保在任何步驟遇到問題時工作器能快速失敗。

當然，這是一個簡化的範例，在實際場景中，你需要與電子商務平台整合、處理邊界情況，甚至深入實現推薦算法。然而，將問題分解為更小的步驟並利用函數式程式設計技術的核心原則依然適用。

詐欺檢測

以下是一個使用相同的鐵路導向程式設計 (ROP) 風格在 Ruby 中實現詐欺檢測工作器的簡化範例：

```
1  class FraudDetectionWorker
2    include Wisper::Publisher
3
4    def call(transaction)
5      Result.ok(FraudDetection.new(transaction))
6        .and_then(ValidateTransaction.method(:validate))
7        .map(AnalyzeTransactionPatterns.method(:analyze))
8        .map(CheckCustomerHistory.method(:check))
9        .map(EvaluateRiskFactors.method(:evaluate))
10       .map(DetermineFraudProbability.method(:determine)).then do |result|
11
12       case result
13       in { err: FraudDetectionError => error }
14         Honeybadger.notify(error.message, context: {transaction:})
15       in { ok: FraudDetection => fraud } }
16         if fraud.high_risk?
17           broadcast(:high_risk_transaction, transaction:, fraud:)
18         else
19           broadcast(:low_risk_transaction, transaction:)
20         end
21       end
22     end
23   end
24 end
```

FraudDetection 類別是一個_值物件_，用於封裝特定交易的詐欺偵測狀態。它提供了一個結構化的方式，基於各種風險因子來分析和評估交易相關的詐欺風險。

```
1  class FraudDetection
2      RISK_THRESHOLD = 0.8
3
4      attr_accessor :transaction, :risk_factors
5
6      def initialize(transaction)
7          self.transaction = transaction
8          self.risk_factors = []
9      end
10
11     def add_risk_factor(description:, probability:)
12         case { description:, probability: }
13         in { description: String => desc, probability: Float => prob }
14             risk_factors << { desc => prob }
15         else
16             raise ArgumentError, "Risk factor arguments should be string and float"
17         end
18     end
19
20     def high_risk?
21         fraud_probability > RISK_THRESHOLD
22     end
23
24     private
25
26     def fraud_probability
27         risk_factors.values.sum
28     end
29 end
```

FraudDetection 類別具有以下屬性：

- `transaction`：指向正在進行詐欺分析的交易的參考。
- `risk_factors`：儲存與交易相關的風險因子的陣列。每個風險因子都以雜湊形式表示，其中鍵是風險因子的描述，值是與該風險因子相關的詐欺機率。

`add_risk_factor` 方法允許向 `risk_factors` 陣列添加風險因子。它接受兩個參數：`description`（描述風險因子的字串）和 `probability`（表示與該風險因子相關的詐欺機率的浮點數）。我們使用 `case..in` 條件式來進行簡單的类型別檢查。

在鏈結結束時將被檢查的 `high_risk?` 謂詞方法會將 `fraud_probability`（通過加總所有風險因子的機率計算得出）與 `RISK_THRESHOLD` 進行比較。

FraudDetection 類別為交易的詐欺偵測提供了一個乾淨且封裝良好的管理方式。它允許添加多個風險因子，每個因子都有其自己的描述和機率，並提供一個方法來根據計算出的詐欺機率確定交易是否被視為高風險。該類別可以輕鬆整合到更大的詐欺偵測系統中，不同組件可以協同工作以評估和降低詐欺交易的風險。

最後，由於這畢竟是一本關於使用人工智慧程式設計的書籍，以下是使用我的 [Raix](#) 函式庫的 `ChatCompletion` 模組來實作 `CheckCustomerHistory` 類別的範例：

```
1  class CheckCustomerHistory
2    include Raix::ChatCompletion
3
4    attr_accessor :fraud_detection
5
6    INSTRUCTION = <<~END
7      You are an AI assistant tasked with checking a customer's transaction
8      history for potential fraud indicators. Given the current transaction
9      and the customer's past transactions, analyze the data to identify any
10     suspicious patterns or anomalies.
11
12     Consider factors such as the frequency of transactions, transaction
13     amounts, geographical locations, and any deviations from the customer's
14     typical behavior to generate a probability score as a float in the range
15     of 0 to 1 (with 1 being absolute certainty of fraud).
16
17     Output the results of your analysis, highlighting any red flags or areas
18     of concern in the following JSON format:
19
20     { description: <Summary of your findings>, probability: <Float> }
21   END
22
23   def self.check(fraud_detection)
24     new(fraud_detection).call
25   end
26
27   def call
28     chat_completion(json: true).tap do |result|
29       fraud_detection.add_risk_factor(**result)
```

```
30     end
31     Result.ok(fraud_detection)
32 rescue StandardError => e
33     Result.err(FraudDetectionError.new(e))
34 end
35
36 private
37
38 def initialize(fraud_detection)
39     self.fraud_detection = fraud_detection
40 end
41
42 def transcript
43     tx_history = fraud_detection.transaction.user.tx_history
44     [
45         { system: INSTRUCTION },
46         { user: "Transaction history: #{tx_history.to_json}" },
47         { assistant: "OK. Please provide the current transaction." },
48         { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
49     ]
50 end
51 end
```

在此範例中，`CheckCustomerHistory` 定義了一個 `INSTRUCTION` 常數，為人工智能模型提供具體指令，說明如何透過系統指令分析客戶的交易歷史以尋找潛在的詐欺指標。

`self.check` 方法是一個類別方法，它會以 `fraud_detection` 物件初始化 `CheckCustomerHistory` 的新實例，並呼叫 `call` 方法來執行客戶歷史分析。

在 `call` 方法內，系統會擷取客戶的交易歷史並將其格式化為一份文字記錄，

然後傳送給人工智能模型。人工智能模型會根據提供的指令分析交易歷史，並回傳其發現的摘要。

這些發現會被添加到 `fraud_detection` 物件中，然後將更新後的 `fraud_detection` 物件作為成功的 `Result` 回傳。

透過運用 `ChatCompletion` 模組，`CheckCustomerHistory` 類別能夠善用人工智能的力量來分析客戶的交易歷史並識別潛在的詐欺指標。這使得詐欺偵測技術更加精密且具適應性，因為人工智能模型能夠隨著時間學習並適應新的模式和異常情況。

更新後的 `FraudDetectionWorker` 和 `CheckCustomerHistory` 類別展示了如何無縫整合人工智能工作器，藉由智能分析和決策能力來強化詐欺偵測流程。

客戶情緒分析

這裡有另一個類似的例子，說明如何實作客戶情緒分析工作器。這次的解釋會少很多，因為你應該已經掌握了這種程式設計風格的要領：

```
1 class CustomerSentimentAnalysisWorker
2   include Wisper::Publisher
3
4   def call(feedback)
5     Result.ok(feedback)
6       .and_then(PreprocessFeedback.method(:preprocess))
7       .map(PerformSentimentAnalysis.method(:analyze))
8       .map(ExtractKeyPhrases.method(:extract))
9       .map(IdentifyTrends.method(:identify))
10      .map(GenerateInsights.method(:generate)).then do |result|
11
12      case result
13      in { err: SentimentAnalysisError => error }
```



```
14         Honeybadger.notify(error.message, context: {feedback:})
15     in { ok: SentimentAnalysisResult => result }
16         broadcast(:sentiment_analysis_completed, result)
17     end
18 end
19 end
20 end
```

在這個例子中，CustomerSentimentAnalysisWorker 的步驟包括對回饋進行預處理（例如，去除雜訊、分詞），執行情感分析以確定整體情感（正面、負面或中性），提取關鍵短語和主題，識別趨勢和模式，並根據分析生成可行的見解。

醫療保健應用

在醫療保健領域，AI 工作器可以在各種任務中協助醫療專業人員和研究人員，從而改善患者預後並加速醫學發現。以下是一些例子：

病人入院登記

AI 工作器可以通過自動化各種任務並提供智能協助來簡化病人入院登記流程。

**** 預約排程：**AI 工作器可以通過理解患者偏好、可用時間和醫療需求的緊急程度來處理預約排程。它們可以通過對話界面與患者互動，引導他們完成預約流程，並根據患者的要求和醫療服務提供者的可用性找到最合適的預約時段。

**** 病史收集：**在病人入院登記期間，AI 工作器可以協助收集和記錄患者的病史。它們可以與患者進行互動對話，詢問有關其過往病況、用藥、過敏史和家族病史的相關問題。AI 工作器可以使用自然語言處理技術來解釋和結構化所收集的信息，確保這些信息準確地記錄在患者的電子健康記錄中。

**** 症狀評估和分層：**AI 工作器可以通過詢問患者目前的症狀、持續時間、嚴重程度和任何相關因素來進行初步症狀評估。通過利用醫學知識庫和機器學習

模型，這些工作器可以分析所提供的信息，生成初步的鑑別診斷或推薦適當的後續步驟，例如安排與醫療服務提供者的諮詢或建議自我護理措施。

**** 保險驗證：****AI 工作器可以在病人入院登記期間協助保險驗證。它們可以收集患者的保險詳細信息，通過 API 或網絡服務與保險提供商溝通，並驗證保險範圍資格和福利。這種自動化有助於簡化保險驗證流程，減少行政負擔並確保信息準確捕獲。

**** 患者教育和指導：****AI 工作器可以根據患者的具體病況或即將進行的醫療程序提供相關的教育材料和指導。它們可以提供個性化內容，回答常見問題，並就就診前準備、用藥說明或治療後護理提供指導。這有助於讓患者在整個醫療過程中保持知情和參與。

通過在病人入院登記中利用 AI 工作器，醫療機構可以提高效率，減少等待時間，改善整體患者體驗。這些工作器可以處理常規任務，收集準確信息，並提供個性化協助，使醫療專業人員能夠專注於為患者提供高質量的護理。

患者風險評估

AI 工作器可以通過分析各種數據源和應用先進的分析技術，在評估患者風險方面發揮關鍵作用。

**** 數據整合：****AI 工作器可以從多個來源收集和理解患者數據，如電子健康記錄、醫學影像、實驗室結果、可穿戴設備和健康社會決定因素。通過將這些信息整合成全面的患者檔案，AI 工作器可以提供患者健康狀況和風險因素的整體視圖。

**** 風險分層：****AI 工作器可以使用預測模型根據患者的個人特徵和健康數據將其分類為不同的風險類別。這種風險分層使醫療服務提供者能夠優先處理需要更即時關注或干預的患者。例如，被識別為特定疾病高風險的患者可以被標記為需要更密切監測、預防措施或早期干預。

**** 個性化風險檔案：****AI 工作器可以為每位患者生成個性化風險檔案，突出顯示影響其風險評分的具體因素。這些檔案可以包括患者的生活方式、遺傳傾

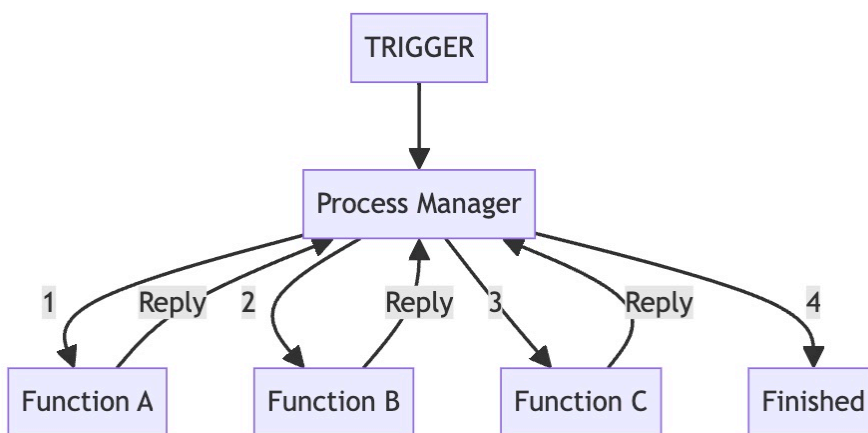
向、環境因素和健康社會決定因素等見解。通過提供風險因素的詳細分析，AI 工作器可以幫助醫療服務提供者為個別患者量身定制預防策略和治療計劃。

**** 持續風險監測：**AI 工作器可以持續監測患者數據並實時更新風險評估。隨著新信息的出現，如生命體徵變化、實驗室結果或用藥依從性，AI 工作器可以重新計算風險評分，並就任何重大變化提醒醫療服務提供者。這種主動監測允許及時干預和調整患者護理計劃。

**** 臨床決策支援：**AI 工作器可以將風險評估結果整合到臨床決策支援系統中，為醫療服務提供者提供基於證據的建議和警報。例如，如果患者某種疾病的風險評分超過特定閾值，AI 工作器可以提示醫療服務提供者根據臨床指南和最佳實踐考慮特定的診斷測試、預防措施或治療選項。

這些工作器能夠處理大量的病患資料，運用複雜的分析方法，並產生可執行的見解來支援臨床決策。這最終能夠改善病患預後，降低醫療成本，並加強群體健康管理。

AI 工作器作為流程管理器



在 AI 驅動的應用程式中，工作器可以被設計為流程管理器的角色，正如 Gregor Hohpe 在《企業整合模式》一書中所描述的。流程管理器是一個核心組件，負責維護流程狀態並根據中間結果決定下一步的處理步驟。

當 AI 工作器作為流程管理器運作時，它會接收一個初始化流程的傳入訊息，稱為觸發訊息。AI 工作器隨後會維護流程執行的狀態（作為對話記錄），並透過一系列實作為工具函數的處理步驟來處理訊息，這些步驟可以是循序的或平行的，並由它自行決定調用時機。



如果你使用的是像 GPT-4 這類知道如何平行執行函數的 AI 模型，那麼你的工作器就可以同時執行多個步驟。坦白說，我自己還沒有嘗試過這樣做，而且憑直覺來說，效果可能因情況而異。

在每個獨立的處理步驟之後，控制權會回到 AI 工作器，使其能夠根據當前狀態和獲得的結果來決定下一個處理步驟。

儲存你的觸發訊息

根據我的經驗，將觸發訊息實作為資料庫支援物件是明智的做法。這樣每個流程實例都能以唯一主鍵識別，並為你提供一個儲存執行相關狀態的位置，包括 AI 的對話記錄。

例如，這是 Olympia 的 AccountChange 模型類別的簡化版本，它代表了對使用者帳戶進行變更的請求。

```
1  # == Schema Information
2  #
3  # Table name: account_changes
4  #
5  #   id          :uuid          not null, primary key
6  #   description :string
7  #   state       :string        not null
8  #   transcript  :jsonb
9  #   created_at  :datetime       not null
10 #   updated_at  :datetime       not null
11 #   account_id  :uuid          not null
12 #
13 # Indexes
14 #
15 #   index_account_changes_on_account_id (account_id)
16 #
17 # Foreign Keys
18 #
19 #   fk_rails_... (account_id => accounts.id)
20 #
21 class AccountChange < ApplicationRecord
22   belongs_to :account
23
24   validates :description, presence: true
25
26   after_commit -> {
27     broadcast(:account_change_requested, self)
28   }, on: :create
29
```

```
30 state_machine initial: :requested do
31   event :completed do
32     transition all => :complete
33   end
34   event :failed do
35     transition all => :requires_human_review
36   end
37 end
38 end
```

AccountChange 類別作為觸發訊息，用於啟動處理帳戶變更請求的流程。請注意它是如何在建立交易完成提交後，透過 Olympia 的 [Wisper](#) 發布/訂閱子系統進行廣播的。

以這種方式將觸發訊息儲存在資料庫中，為每個帳戶變更請求提供了持久性記錄。AccountChange 類別的每個實例都會被分配一個唯一的主鍵，便於識別和追蹤個別請求。這對於稽核日誌記錄特別有用，因為它使系統能夠維護所有帳戶變更的歷史記錄，包括請求的時間、請求的變更內容，以及每個請求的當前狀態。

在給定的示例中，AccountChange 類別包含諸如 description 等欄位來捕捉請求變更的細節，state 用於表示請求的當前狀態（例如：已請求、已完成、需要人工審核），以及 transcript 用於儲存與請求相關的 AI 對話記錄。description 欄位是用於啟動與 AI 首次聊天完成的實際提示。儲存這些資料提供了寶貴的上下文，並允許更好地追蹤和分析帳戶變更流程。

在資料庫中儲存觸發訊息能夠實現強大的錯誤處理和恢復功能。如果在處理帳戶變更請求期間發生錯誤，系統會將請求標記為失敗，並將其轉換為需要人工干預的狀態。這確保了沒有請求會遺失或被遺忘，且任何問題都能得到適當的處理和解決。

AI 工作者作為流程管理器，提供了一個中央控制點，並實現了強大的流程報告和除錯功能。然而，需要注意的是，在應用程式的每個工作流程場景中都使用 AI 工作者作為流程管理器可能會過於複雜。

將 AI 工作者整合到您的應用程式架構中

在將 AI 工作者整合到您的應用程式架構時，需要解決幾個技術考量，以確保 AI 工作者與其他應用程式元件之間的順暢整合和有效通訊。本節將探討設計這些介面、處理資料流程和管理 AI 工作者生命週期的關鍵方面。

設計清晰的介面和通訊協議

為了促進 AI 工作者與其他應用程式元件之間的無縫整合，定義清晰的介面和通訊協議至關重要。考慮以下方法：

基於 API 的整合：透過明確定義的 API（如 RESTful 端點或 GraphQL 架構）公開 AI 工作者的功能。這允許其他元件使用標準的 HTTP 請求和回應與 AI 工作者互動。基於 API 的整合在 AI 工作者和使用元件之間提供了明確的契約，使整合點的開發、測試和維護更加容易。

基於訊息的通訊：實作訊息佇列或發布-訂閱系統等基於訊息的通訊模式，以實現 AI 工作者與其他元件之間的非同步互動。這種方法將 AI 工作者與應用程式的其餘部分解耦，實現更好的可擴展性、容錯性和鬆散耦合。當 AI 工作者執行的處理耗時或資源密集時，基於訊息的通訊特別有用，因為它允許應用程式的其他部分在不等待 AI 工作者完成任務的情況下繼續執行。

事件驅動架構：圍繞在特定條件滿足時啟動 AI 工作者的事件和觸發器設計您的系統。AI 工作者可以訂閱相關事件並做出相應反應，在事件發生時執行其指定的任務。事件驅動架構實現了即時處理，並允許按需調用 AI 工作者，減少不必要的資源消耗。這種方法特別適用於 AI 工作者需要響應特定操作或應用程式狀態變化的場景。

處理資料流程和同步

在將 AI 工作者整合到您的應用程式時，確保順暢的資料流程並維護 AI 工作者與其他元件之間的資料一致性至關重要。考慮以下方面：

資料準備：在將資料輸入 AI 工作者之前，您可能需要執行各種資料準備任務，如清理、格式化和/或轉換輸入資料。您不僅要確保 AI 工作者能夠有效處理，還要確保您不會浪費記號來關注工作者可能認為無用甚至分散注意力的資訊。資料準備可能涉及去除雜訊、處理缺失值或轉換資料類型等任務。

資料持久化：您將如何儲存和持久化流入和流出 AI 工作者的資料？考慮資料量、查詢模式和可擴展性等因素。您是否需要為了稽核或除錯目的而保存 AI 的對話記錄作為其「思考過程」的反映，或者只保留結果記錄就足夠了？

資料檢索：取得工作者所需的資料可能涉及查詢資料庫、讀取檔案或存取外部 API。請務必考慮延遲性，以及 AI 工作者如何存取最新的資料。他們是否需要完整存取您的資料庫，還是應該根據其工作內容來嚴格界定其存取範圍？擴展性又該如何？請考慮使用快取機制來提升效能並減輕底層資料來源的負載。

資料同步：當多個元件（包括 AI 工作者）存取和修改共享資料時，實作適當的同步機制以維持資料一致性就變得很重要。資料庫鎖定策略，如樂觀鎖定或悲觀鎖定，可以幫助您預防衝突並確保資料完整性。實作交易管理技術來組織相關的資料操作，並維持原子性、一致性、隔離性和持久性（ACID）特性。

錯誤處理與復原：實作強健的錯誤處理和復原機制，以處理資料流程中可能出現的資料相關問題。優雅地處理異常並提供有意義的錯誤訊息以協助除錯。實作重試機制和備用策略來處理暫時性故障或網路中斷。定義清楚的資料復原和還原程序，以應對資料損壞或遺失的情況。

透過仔細設計和實作資料流程和同步機制，您可以確保您的 AI 工作者能夠存取準確、一致且最新的資料。這使他們能夠有效執行任務並產生可靠的結果。

管理 AI 工作者的生命週期

開發一個標準化的程序來初始化和設定 AI 工作者。我偏好使用那些能標準化定義模型名稱、系統指令和函式定義等設定的框架。確保初始化程序是自動化且可重現的，以便於部署和擴展。

實作全面的監控和日誌記錄機制來追蹤 AI 工作者的健康狀況和效能。收集諸如資源使用率、處理時間、錯誤率和處理量等指標。使用集中式日誌系統如 ELK stack（Elasticsearch、Logstash、Kibana）來彙整和分析來自多個 AI 工作者的日誌。

在 AI 工作者架構中建立容錯和彈性。實作錯誤處理和復原機制以優雅地處理故障或異常。大型語言模型仍是最前沿的技術；提供者往往會在意想不到的時候發生故障。使用重試機制和斷路器來防止連鎖故障。

AI 工作者的可組合性和編排

AI 工作者架構的一個重要優勢是其可組合性，這允許您組合和編排多個 AI 工作者來解決複雜問題。通過將較大的任務分解成較小、更易管理的子任務，每個子任務由專門的 AI 工作者處理，您可以創建強大且靈活的系統。在本節中，我們將探討組合和編排「多個」AI 工作者的不同方法。

串接 AI 工作者以實現多步驟工作流程

在許多情況下，複雜的任務可以分解為一系列順序步驟，其中一個 AI 工作者的輸出成為下一個的輸入。這種 AI 工作者的串接創建了一個多步驟工作流程或管道。鏈中的每個 AI 工作者專注於特定的子任務，最終輸出是所有工作者共同努力的結果。

讓我們考慮一個 Ruby on Rails 應用程式中處理使用者生成內容的例子。工作流程包含以下步驟，這些步驟在現實生活中可能都太簡單而不值得這樣分解，但它們使示例更容易理解：

1. **文字清理**：負責移除 HTML 標籤、轉換文字為小寫，並處理 Unicode 正規化的 AI 工作者。
2. **語言偵測**：識別已清理文字語言的 AI 工作者。
3. **情緒分析**：基於偵測到的語言來判斷文字情緒（正面、負面或中性）的 AI 工作者。
4. **內容分類**：使用自然語言處理技術將文字分類到預定義類別的 AI 工作者。

以下是一個非常簡化的示例，展示如何使用 Ruby 來串接這些 AI 工作者：

```
1 class ContentProcessor
2   def initialize(text)
3     @text = text
4   end
5
6   def process
7     cleaned_text = TextCleanupWorker.new(@text).call
8     language = LanguageDetectionWorker.new(cleaned_text).call
9     sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
10    category = CategorizationWorker.new(cleaned_text, language).call
11
12    { cleaned_text:, language:, sentiment:, category: }
13  end
14 end
```

在此範例中，ContentProcessor 類別使用原始文字進行初始化，並在 process 方法中將 AI 工作器串接在一起。每個 AI 工作器執行其特定任務，並將結果傳遞給鏈中的下一個工作器。最終輸出是一個雜湊，其中包含已清理的文字、偵測到的語言、情緒分析結果和內容類別。

獨立 AI 工作器的平行處理

在先前的範例中，AI 工作器是按順序串接的，每個工作器處理文字後將結果傳遞給下一個工作器。然而，如果您有多個可以對相同輸入獨立運作的 AI 工作器，您可以透過平行處理來優化工作流程。

在給定的情境中，一旦 `TextCleanupWorker` 完成文字清理，`LanguageDetectionWorker`、`SentimentAnalysisWorker` 和 `CategorizationWorker` 都可以獨立處理已清理的文字。透過平行執行這些工作器，您可以潛在減少整體處理時間並提升工作流程的效率。

若要在 Ruby 中實現平行處理，您可以運用執行緒或非同步程式設計等並發技術。以下是一個示例，說明如何修改 `ContentProcessor` 類別，使用執行緒來平行處理最後三個工作器：

```
1  require 'concurrent'
2
3  class ContentProcessor
4    def initialize(text)
5      @text = text
6    end
7
8    def process
9      cleaned_text = TextCleanupWorker.new(@text).call
10
11      language_future = Concurrent::Future.execute do
12        LanguageDetectionWorker.new(cleaned_text).call
13      end
14
15      sentiment_future = Concurrent::Future.execute do
16        SentimentAnalysisWorker.new(cleaned_text).call
```

```
17     end
18
19     category_future = Concurrent::Future.execute do
20         CategorizationWorker.new(cleaned_text).call
21     end
22
23     language = language_future.value
24     sentiment = sentiment_future.value
25     category = category_future.value
26
27     { cleaned_text:, language:, sentiment:, category: }
28 end
29 end
```

在這個優化版本中，我們使用 `concurrent-ruby` 函式庫為每個獨立的 AI 工作器創建 `Concurrent::Future` 物件。`Future` 代表一個將在獨立執行緒中非同步執行的運算。

在文本清理步驟之後，我們創建了三個 `Future` 物件：`language_future`、`sentiment_future` 和 `category_future`。每個 `Future` 在獨立的執行緒中執行其對應的 AI 工作器（`LanguageDetectionWorker`、`SentimentAnalysisWorker` 和 `CategorizationWorker`），並將 `cleaned_text` 作為輸入。

通過調用每個 `Future` 的 `value` 方法，我們等待運算完成並獲取結果。`value` 方法會阻塞直到結果可用，確保所有並行工作器在程式繼續執行之前都已完成處理。

最後，我們像原始範例一樣，使用清理後的文本和並行工作器的結果構建輸出雜湊。

通過並行處理獨立的 AI 工作器，與順序執行相比，您可以潛在減少整體處理時間。當處理耗時的任務或處理大量數據時，這種優化特別有益。

然而，需要注意的是，實際的性能提升取決於多個因素，例如每個工作器的複雜度、可用的系統資源以及執行緒管理的開銷。建議始終對程式碼進行基準測試和效能分析，以確定特定使用案例的最佳並行程度。

此外，在實現並行處理時，要注意工作器之間的共享資源或依賴關係。確保工作器可以獨立運作而不會發生衝突或競態條件。如果存在依賴關係或共享資源，您可能需要實現適當的同步機制來維護數據完整性並避免死鎖或不一致結果等問題。

Ruby 的全域直譯器鎖與非同步處理

在考慮 Ruby 中基於執行緒的非同步處理時，理解 Ruby 的全域直譯器鎖 (GIL) 的影響很重要。

GIL 是 Ruby 直譯器中的一種機制，即使在多核心處理器上，也確保同一時間只有一個執行緒可以執行 Ruby 程式碼。這意味著雖然可以在 Ruby 行程中創建和管理多個執行緒，但在任何給定時刻只有一個執行緒可以主動執行 Ruby 程式碼。

GIL 的設計目的是簡化 Ruby 直譯器的實現並為 Ruby 的內部數據結構提供執行緒安全性。然而，它也限制了 Ruby 程式碼真正並行執行的可能性。

當您在 Ruby 中使用執行緒時，無論是使用 `concurrent-ruby` 函式庫還是內建的 `Thread` 類別，執行緒都受到 GIL 的限制。GIL 允許每個執行緒執行 Ruby 程式碼一小段時間，然後切換到另一個執行緒，創造並發執行的假象。

然而，由於 GIL 的存在，Ruby 程式碼的實際執行仍然是順序的。當一個執行緒在執行 Ruby 程式碼時，其他執行緒實際上處於暫停狀態，等待輪到它們獲取 GIL 並執行。

這意味著 Ruby 中基於執行緒的非同步處理對 I/O 密集型任務最有效，例如等待外部 API 響應（如第三方託管的大型語言模型）或執行檔案 I/O 操作。當執行緒遇到 I/O 操作時，它可以釋放 GIL，允許其他執行緒在等待 I/O 完

成時執行。

另一方面，對於 CPU 密集型任務，如密集計算或長時間運行的 AI 工作器處理，GIL 可能會限制基於執行緒的並行處理的潛在性能提升。由於同一時間只有一個執行緒可以執行 Ruby 程式碼，與順序處理相比，整體執行時間可能不會顯著減少。

要在 Ruby 中實現 CPU 密集型任務的真正並行執行，您可能需要探索其他方法，例如：

- 使用基於行程的並行處理，每個 Ruby 行程運行在單獨的 CPU 核心上。
- 利用提供原生擴展或接口的外部函式庫或框架，連接到沒有 GIL 的語言，如 C 或 Rust。
- 使用分散式計算框架或訊息佇列在多台機器或行程間分配任務。

在設計和實現 Ruby 中的非同步處理時，考慮任務的性質和 GIL 施加的限制至關重要。雖然基於執行緒的非同步處理可以為 I/O 密集型任務提供好處，但由於 GIL 的限制，它可能無法為 CPU 密集型任務提供顯著的性能改進。

提高準確度的集成技術

集成技術涉及組合多個 AI 工作器的輸出，以提高系統的整體準確度或穩健性。集成技術不是依賴單一 AI 工作器，而是利用多個工作器的集體智慧來做出更明智的決策。



集成系統在工作流程的不同部分需要不同 AI 模型時特別重要，這種情況比你想像的更為普遍。像 GPT-4 這樣強大的模型與較簡單的開源選項相比非常昂貴，而且可能不是應用程式中每個工作流程步驟都需要用到。

多數投票法是一種常見的集成技術，多個 AI 工作單元獨立處理相同的輸入，最

終輸出由多數共識決定。這種方法可以幫助減輕個別工作單元錯誤的影響，提高系統的整體可靠性。

讓我們考慮一個例子，我們有三個用於情感分析的 AI 工作單元，每個都使用不同的模型或提供不同的上下文。我們可以使用多數投票法來整合它們的輸出，從而確定最終的情感預測結果。

```
1  class SentimentAnalysisEnsemble
2    def initialize(text)
3      @text = text
4    end
5
6    def analyze
7      predictions = [
8        SentimentAnalysisWorker1.new(@text).analyze,
9        SentimentAnalysisWorker2.new(@text).analyze,
10       SentimentAnalysisWorker3.new(@text).analyze
11      ]
12
13      predictions
14        .group_by { |sentiment| sentiment }
15        .max_by { |_, votes| votes.size }
16        .first
17
18    end
19  end
```

在這個例子中，SentimentAnalysisEnsemble 類別在初始化時接收文本並調用三個不同的情感分析 analyze 方法收集每個工作器的預測結果，並使用 group_by 和 max_by 方法來決定多數情感傾向。最終輸出是在工作器集成中獲得最多票數的情感傾向。



集成系統顯然是值得花時間研究平行處理的案例。

AI 工作器的動態選擇和調用

在某些（如果不是大多數）情況下，具體要調用哪個 AI 工作器可能取決於執行時條件或使用者輸入。AI 工作器的動態選擇和調用為系統提供了靈活性和適應性。



你可能會想要嘗試在單個 AI 工作器中塞入大量功能，給它許多函數和一個複雜的提示來解釋如何調用它們。相信我，請克制這種誘惑。我們在本章討論的方法之所以被稱為“多工作器模式”，其中一個原因就是提醒我們：擁有許多專門的工作器是可取的，每個工作器都在為更大的目標做著自己的小任務。

例如，考慮一個聊天機器人應用程式，其中不同的 AI 工作器負責處理不同類型的使用者查詢。基於使用者的輸入，應用程式動態選擇合適的 AI 工作器來處理查詢。

```
1 class ChatbotController < ApplicationController
2   def process_query
3     query = params[:query]
4     query_type = QueryClassifierWorker.new(query).classify
5
6     case query_type
7     when 'greeting'
8       response = GreetingWorker.new(query).generate_response
9     when 'product_inquiry'
10      response = ProductInquiryWorker.new(query).generate_response
```



```
11     when 'order_status'
12       response = OrderStatusWorker.new(query).generate_response
13     else
14       response = DefaultResponseWorker.new(query).generate_response
15     end
16
17     render json: { response: response }
18   end
19 end
```

在這個例子中，ChatbotController 通過 `process_query` 動作接收用戶查詢。它首先使用 `QueryClassifierWorker` 來確定查詢的類型。根據分類後的查詢類型，控制器動態選擇合適的 AI 工作器來生成回應。這種動態選擇使聊天機器人能夠處理不同類型的查詢，並將它們路由到相關的 AI 工作器。



由於 `QueryClassifierWorker` 的工作相對簡單，不需要太多上下文或函數定義，你可能可以使用超快速的小型 LLM，如 `mistralai/mixtral-8x7b-instruct:nitro` 來實現它。在我寫這篇文章時，它在許多任務上的能力接近 GPT-4 的水平，而且 Groq 可以以驚人的 444 個詞元/秒的速度處理它。

將傳統 NLP 與 LLM 結合

雖然大型語言模型（LLM）徹底改變了自然語言處理（NLP）領域，在廣泛的任務中提供了無與倫比的多功能性和性能，但它們並不總是每個問題最有效或最具成本效益的解決方案。在許多情況下，將傳統的 NLP 技術與 LLM 結合可以帶來更優化、更有針對性和更經濟的方法來解決特定的 NLP 挑戰。

可以把 LLM 想像成 NLP 領域的瑞士軍刀——非常靈活和強大，但並不一定是每項工作的最佳工具。有時候，像開瓶器或開罐器這樣的專用工具可能對特定任

務更有效率。同樣，傳統的 NLP 技術，如文件聚類、主題識別和分類，通常可以為你的 NLP 流程的某些方面提供更有針對性和更具成本效益的解決方案。

傳統 NLP 技術的一個主要優勢是其計算效率。這些方法通常依賴於較簡單的統計模型或基於規則的方法，與 LLM 相比，可以更快速且計算開銷更低地處理大量文本數據。這使得它們特別適合於涉及分析和組織大量文檔的任務，例如對相似文章進行聚類或識別文本集中的關鍵主題。

此外，傳統的 NLP 技術在特定任務上通常可以達到較高的準確性和精確度，特別是在針對特定領域數據集進行訓練時。例如，使用支持向量機 (SVM) 或樸素貝葉斯等傳統機器學習算法的經過良好調整的文檔分類器，可以以最小的計算成本準確地將文檔分類到預定義的類別中。

然而，當涉及需要更深入理解語言、上下文和推理的任務時，LLM 真正展現出其優勢。它們生成連貫且符合上下文的文本、回答問題和總結長段落的能力是傳統 NLP 方法無法比擬的。LLM 可以有效處理複雜的語言現象，如歧義、共指和習語表達，這使它們在需要自然語言生成或理解的任務中變得非常寶貴。

真正的力量在於將傳統 NLP 技術與 LLM 結合，創建利用兩者優勢的混合方法。通過使用傳統 NLP 方法進行文檔預處理、聚類和主題提取等任務，你可以高效地組織和構建文本數據。然後，這些結構化信息可以輸入到 LLM 中進行更高級的任務，如生成摘要、回答問題或創建全面的報告。

例如，讓我們考慮一個用例，你想要根據大量個別趨勢文檔生成特定領域的趨勢報告。與其完全依賴 LLM（這對於處理大量文本來說可能在計算上昂貴且耗時），你可以採用混合方法：

1. 使用傳統的 NLP 技術，如主題建模（例如潛在狄利克雷分配）或聚類算法（例如 K-均值），將相似的趨勢文檔分組並識別語料庫中的關鍵主題。
2. 將聚類後的文檔和識別出的主題輸入到 LLM 中，利用其卓越的語言理解和生成能力為每個聚類或主題創建連貫且信息豐富的摘要。
3. 最後，使用 LLM 通過組合個別摘要來生成全面的趨勢報告，突出最重要的趨勢，並基於匯總的信息提供見解和建議。

通過這種方式將傳統 NLP 技術與 LLM 結合，你可以高效處理大量文本數據，提取有意義的見解，並生成高質量的報告，同時優化計算資源和成本。

當您開始進行自然語言處理項目時，至關重要的是要仔細評估每項任務的具體需求和限制條件，並考慮如何將傳統自然語言處理方法與大型語言模型結合使用，以達到最佳效果。通過將傳統技術的效率和精確度與大型語言模型的多功能性和強大功能相結合，您可以創建高效且經濟實惠的自然語言處理解決方案，為您的用戶和利益相關者帶來價值。

工具使用



在人工智慧驅動的應用程式開發領域中，「工具使用」或「函式呼叫」已經成為一項強大的技術，使您的大型語言模型（LLM）能夠連接到外部工具、API、函式、資料庫和其他資源。這種方法不僅能夠產生文字輸出，還能實現人工智慧元件與應用程式生態系統其他部分之間更動態的互動。正如我們將在本章中探討的，工具使用還為您提供了讓人工智慧模型以結構化方式生成資料的選項。

什麼是工具使用？

工具使用，也被稱為函式呼叫，是一種允許開發人員指定 LLM 在生成過程中可以互動的函式清單的技術。這些工具可以是簡單的工具函式，也可以是複雜的 API 或資料庫查詢。通過為 LLM 提供這些工具的存取權限，開發人員可以擴展模型的功能，使其能夠執行需要外部知識或操作的任務。

圖 8. 分析文件的 AI 工作者的函式定義示例

```
1  FUNCTION = {
2      name: "save_analysis",
3      description: "Save analysis data for document",
4      parameters: {
5          type: "object",
6          properties: {
7              title: {
8                  type: "string",
9                  maxLength: 140
10             },
11             summary: {
12                 type: "string",
13                 description: "comprehensive multi-paragraph summary with
14                             overview and list of sections (if applicable)"
15             },
16             tags: {
17                 type: "array",
18                 items: {
19                     type: "string",
20                     description: "lowercase tags representing main themes
21                             of the document"
22                 }
23             }
24         },
25         "required": %w[title summary tags]
26     }
27 }.freeze
```

工具使用的核心理念是賦予 LLM 根據使用者輸入或當前任務動態選擇和執行

適當工具的能力。工具使用不僅依賴模型的預訓練知識，還能運用外部資源來生成更準確、更相關且更具可操作性的回應。工具使用使得 RAG（檢索增強生成）等技術的實現變得比以往更加容易。

請注意，除非另有說明，本書假設您的 AI 模型沒有任何內建的伺服器端工具。任何您想要提供給 AI 使用的工具都必須由您在每個 API 請求中明確宣告，並在 AI 表示想要在其回應中使用該工具時提供執行方案。

工具使用的潛力

工具使用為 AI 驅動的應用程式開創了廣泛的可能性。以下是一些可以通過工具使用實現的例子：

1. **聊天機器人和虛擬助理**：通過將 LLM 連接到外部工具，聊天機器人和虛擬助理可以執行更複雜的任務，例如從資料庫檢索資訊、執行 API 呼叫或與其他系統互動。舉例來說，聊天機器人可以使用客戶關係管理工具根據使用者的要求更改交易狀態。
2. **資料分析和洞察**：LLM 可以連接到資料分析工具或函式庫來執行進階資料處理任務。這使應用程式能夠根據使用者查詢生成見解、進行比較分析或提供資料驅動的建議。
3. **搜尋和資訊檢索**：工具使用允許 LLM 與搜尋引擎、向量資料庫或其他資訊檢索系統互動。通過將使用者查詢轉換為搜尋查詢，LLM 可以從多個來源檢索相關資訊，並為使用者問題提供全面的答案。
4. **與外部服務整合**：工具使用實現了 AI 驅動的應用程式與外部服務或 API 的無縫整合。例如，LLM 可以與天氣 API 互動以提供即時天氣更新，或與翻譯 API 互動以生成多語言回應。

工具使用工作流程

工具使用工作流程通常包含四個關鍵步驟：

1. 在請求上下文中包含函數定義
2. 動態（或明確）工具選擇
3. 執行函數
4. 可選的原始提示詞延續

讓我們詳細檢視每個步驟。

在請求上下文中包含函數定義

AI 之所以知道它可以使用哪些工具，是因為您在完成請求中提供了一個工具列表（通常使用 JSON 架構的變體來定義函數）。

工具定義的具體語法因模型而異。

以下是在 Claude 3 中定義 `get_weather` 函數的方式：

```
1  {
2      "name": "get_weather",
3      "description": "Get the current weather in a given location",
4      "input_schema": {
5          "type": "object",
6          "properties": {
7              "location": {
8                  "type": "string",
9                  "description": "The city and state, e.g. San Francisco, CA"
10             },
11             "unit": {
```

```
12         "type": "string",
13         "enum": ["celsius", "fahrenheit"],
14         "description": "The unit of temperature"
15     }
16 },
17     "required": ["location"]
18 }
19 }
```

而對於 GPT-4 來說，要定義相同的函數，你需要將它作為 `tools` 參數的值來傳遞：

```
1  {
2      "name": "get_current_weather",
3      "description": "Get the current weather in a given location",
4      "parameters": {
5          "type": "object",
6          "properties": {
7              "location": {
8                  "type": "string",
9                  "description": "The city and state, e.g. San Francisco, CA",
10             },
11             "unit": {
12                 "type": "string",
13                 "enum": ["celsius", "fahrenheit"],
14                 "description": "The unit of temperature"
15             },
16         },
17         "required": ["location"],
18     },
19 }
```


19 }

幾乎一樣，卻莫名其妙地不同！真是惱人。

函式定義需要指定名稱、描述和輸入參數。輸入參數可以透過屬性進一步定義，例如使用列舉來限制可接受的值，以及指定參數是否為必要項目。

除了實際的函式定義之外，你還可以在系統指令中加入使用該函式的原因和方法的說明。

舉例來說，我在 Olympia 中的網路搜尋工具包含這個系統指令，它提醒 AI 可以使用這些提到的工具：

```
1 The `google_search` and `realtime_search` functions let you do research
2 on behalf of the user. In contrast to Google, realtime search is powered
3 by Perplexity and provides real-time information to curated current events
4 databases and news sources. Make sure to include URLs in your response so
5 user can do followup research.
```

提供詳細的描述被認為是工具效能最重要的因素。您的描述應該解釋有關工具的每個細節，包括：

- 工具的功能
- 何時應該使用（以及何時不該使用）
- 每個參數的含義以及如何影響工具的行為
- 任何適用於工具實作的重要注意事項或限制

您能為 AI 提供的工具相關上下文越多，AI 就越能更好地決定何時以及如何使用這些工具。例如，Anthropic 建議其 Claude 3 系列的每個工具描述至少要有 3-4 個句子，如果工具較為複雜，則需要更多說明。

這可能不太直觀，但描述被認為比範例更重要。雖然您可以在工具描述或附帶的提示中包含如何使用工具的範例，但這比起有一個清晰且全面的工具用途和參數說明來得次要。請在完整撰寫描述之後，才添加範例。

以下是一個類似 Stripe 的 API 函式規格範例：

```
1  {
2    "name": "createPayment",
3    "description": "Create a new payment request",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "transaction_amount": {
8          "type": "number",
9          "description": "The amount to be paid"
10       },
11       "description": {
12         "type": "string",
13         "description": "A brief description of the payment"
14       },
15       "payment_method_id": {
16         "type": "string",
17         "description": "The payment method to be used"
18       },
19       "payer": {
20         "type": "object",
21         "description": "Information about the payer, including their name,
22                        email, and identification number",
23         "properties": {
24           "name": {
```

```
25         "type": "string",
26         "description": "The payer's name"
27     },
28     "email": {
29         "type": "string",
30         "description": "The payer's email address"
31     },
32     "identification": {
33         "type": "object",
34         "description": "The payer's identification number",
35         "properties": {
36             "type": {
37                 "type": "string",
38                 "description": "Identification document (e.g. CPF, CNPJ)"
39             },
40             "number": {
41                 "type": "string",
42                 "description": "The identification number"
43             }
44         },
45         "required": [ "type", "number" ]
46     }
47 },
48 "required": [ "name", "email", "identification" ]
49 }
50 }
51 }
```



實際上，某些模型在處理巢狀函式規範和處理複雜的輸出資料類型時會遇到困難，例如陣列、字典等。但理論上，你應該能夠提供任意深度的 JSON Schema 規範！

動態工具選擇

當你執行包含工具定義的聊天完成時，LLM 會動態選擇最適合使用的工具，並為每個工具生成所需的輸入參數。

實際上，AI 要精確地呼叫正確的函式，並完全遵循你對輸入的規範，其成功率時好時壞。將溫度超參數完全調至 0.0 會有很大幫助，但根據我的經驗，你仍然會偶爾遇到錯誤。這些失敗包括幻想出的函式名稱、命名錯誤或完全缺失的輸入參數。參數是以 JSON 形式傳遞的，這意味著有時你會看到由於 JSON 被截斷、引號錯誤或其他破壞而導致的錯誤。



[自我修復資料](#)模式可以幫助[自動修復](#)因語法錯誤而失敗的函式呼叫。

強制（又稱顯式）工具選擇

某些模型讓你可以請求中以參數的方式強制呼叫特定函式。否則，是否呼叫函式完全取決於 AI 的判斷。

在某些情況下，強制函式呼叫的能力非常重要，因為你可能想要確保特定工具或函式被執行，而不考慮 AI 的動態選擇過程。這項功能重要的原因有以下幾點：

1. **明確控制**：你可能正在將 AI 作為離散元件使用，或在預定義的工作流程中需要在特定時間執行特定函式。通過強制呼叫，你可以確保所需的函式被執行，而不是要委婉地請求 AI 去執行。

2. **除錯和測試**：在開發和測試 AI 驅動的應用程式時，強制函式呼叫的能力對除錯來說非常寶貴。通過明確觸發特定函式，你可以隔離並測試應用程式的個別元件。這使你能夠驗證函式實作的正確性，驗證輸入參數，並確保返回預期的結果。
3. **處理邊緣案例**：可能會有一些邊緣案例或特殊情況，在這些情況下，AI 的動態選擇過程可能不會選擇執行應該執行的函式，而你基於外部流程知道應該執行。在這種情況下，具備強制函式呼叫的能力可以讓你明確處理這些情況。在你的應用程式邏輯中定義規則或條件，以決定何時要覆寫 AI 的判斷。
4. **一致性和可重現性**：如果你有一系列需要按特定順序執行的函式，強制呼叫可以確保每次都遵循相同的順序。這在需要一致性和可預測行為的應用程式中特別重要，例如金融系統或科學模擬。
5. **效能優化**：在某些情況下，強制函式呼叫可以帶來效能優化。如果你知道特定任務需要某個特定函式，而 AI 的動態選擇過程可能會引入不必要的開銷，你可以繞過選擇過程直接呼叫所需的函式。這可以幫助減少延遲並提高應用程式的整體效率。

總的來說，在 AI 驅動的應用程式中強制函式呼叫的能力提供了明確的控制，有助於除錯和測試，處理邊緣案例，確保一致性和可重現性。這是你工具箱中的一個強大工具，但我們還需要討論這個重要功能的另一個方面。



在許多決策用例中，我們總是希望模型進行函式呼叫，可能永遠不希望模型僅使用其內部知識來回應。例如，如果你正在多個專門處理不同任務的模型（多語言輸入、數學等）之間進行路由，你可能會使用函式呼叫模型將請求委派給其中一個輔助模型，而不是獨立回應。

工具選擇參數

GPT-4 和其他支援函式呼叫的語言模型提供了 `tool_choice` 參數，用於控制完成時是否需要使用工具。此參數有三個可能的值：

- `auto` 讓 AI 完全自行決定是否使用工具或直接回應
- `required` 告訴 AI 它必須呼叫工具而不是直接回應，但讓 AI 自行選擇使用哪個工具
- 第三個選項是設定你想要強制使用的 `name_of_function`。更多內容將在下一節討論。



請注意，如果您將 `tool choice` 設置為 `required`，模型將被迫從提供的函數中選擇最相關的一個來呼叫，即使沒有一個真正符合提示。在發布時，我還沒有看到有任何模型會返回空的 `tool_calls` 響應，或使用其他方式來告知它沒有找到合適的函數來呼叫。

強制函數呼叫以獲取結構化輸出

強制函數呼叫的能力為您提供了一種方法，可以從聊天完成中強制獲取結構化數據，而不必自己從純文本響應中提取數據。

為什麼強制函數獲取結構化輸出如此重要？簡單來說，因為從 LLM 輸出中提取結構化數據是一件非常麻煩的事。您可以通過要求以 XML 格式輸出數據來使生活變得稍微輕鬆一點，但之後您還得解析 XML。而且當您的 AI 回應：“對不起，由於某某原因，我無法生成您請求的數據...”時，您該怎麼辦？

在以這種方式使用工具時：

- 您可能應該在請求中只定義一個工具
- 記得使用 `tool_choice` 參數來強制使用其函數
- 記住模型會將輸入傳遞給工具，所以工具的名稱和描述應該從模型的角度出發，而不是您的角度

最後一點需要一個例子來說明。假設您正在要求 AI 對用戶文本進行情緒分析。函數的名稱不應該是 `analyze_sentiment`，而應該是類似 `save_sentiment_analysis` 這樣的名稱。進行情緒分析的是 AI，而不是工具。從 AI 的角度來看，工具所做的只是保存分析結果。

以下是使用 Claude 3 將圖像摘要記錄為結構良好的 JSON 的示例，這次是使用 `curl` 從命令行執行：

```
1  curl https://api.anthropic.com/v1/messages \
2      --header "content-type: application/json" \
3      --header "x-api-key: $ANTHROPIC_API_KEY" \
4      --header "anthropic-version: 2023-06-01" \
5      --header "anthropic-beta: tools-2024-04-04" \
6      --data \
7  '{
8      "model": "claude-3-sonnet-20240229",
9      "max_tokens": 1024,
10     "tools": [{
11         "name": "record_summary",
12         "description": "Record summary of image into well-structured JSON.",
13         "input_schema": {
14             "type": "object",
15             "properties": {
16                 "key_colors": {
17                     "type": "array",
18                     "items": {
19                         "type": "object",
20                         "properties": {
21                             "r": {
22                                 "type": "number",
```

```
23         "description": "red value [0.0, 1.0]"
24     },
25     "g": {
26         "type": "number",
27         "description": "green value [0.0, 1.0]"
28     },
29     "b": {
30         "type": "number",
31         "description": "blue value [0.0, 1.0]"
32     },
33     "name": {
34         "type": "string",
35         "description": "Human-readable color name
36                         in snake_case, e.g.
37                         \"olive_green\"or
38                         \"turquoise\""
39     },
40 },
41     "required": [ "r", "g", "b", "name" ]
42 },
43     "description": "Key colors in the image. Four or less."
44 },
45     "description": {
46         "type": "string",
47         "description": "Image description. 1-2 sentences max."
48     },
49     "estimated_year": {
50         "type": "integer",
51         "description": "Estimated year that the image was taken,
```



```
52             if is it a photo. Only set this if the
53             image appears to be non-fictional.
54             Rough estimates are okay!"
55         }
56     },
57     "required": [ "key_colors", "description" ]
58 }
59 ]],
60 "messages": [
61     {
62         "role": "user",
63         "content": [
64             {
65                 "type": "image",
66                 "source": {
67                     "type": "base64",
68                     "media_type": "'$IMAGE_MEDIA_TYPE'",
69                     "data": "'$IMAGE_BASE64'"
70                 }
71             },
72             {
73                 "type": "text",
74                 "text": "Use `record_summary` to describe this image."
75             }
76         ]
77     }
78 ]
79 }'
```

在提供的範例中，我們使用 Anthropic 的 Claude 3 模型來生成圖片的結構化

JSON 摘要。以下是其運作方式：

1. 我們在請求內容的 `tools` 陣列中定義了一個名為 `record_summary` 的工具。這個工具負責將圖片的摘要記錄為結構完善的 JSON。
2. `record_summary` 工具具有一個 `input_schema`，用於指定預期的 JSON 輸出結構。它定義了三個屬性：
 - `key_colors`：一個物件陣列，表示圖片中的主要顏色。每個顏色物件都包含紅、綠、藍值（範圍從 0.0 到 1.0）以及以 `snake_case` 格式表示的人類可讀顏色名稱。
 - `description`：字串屬性，用於簡短描述圖片，限制在 1-2 個句子內。
 - `estimated_year`：可選的整數屬性，用於估計照片拍攝的年份（如果它看起來是非虛構的照片）。
3. 在 `messages` 陣列中，我們提供了以 `base64` 編碼的圖片數據及其媒體類型。這使模型能夠將圖片作為輸入的一部分進行處理。
4. 我們還提示 Claude 使用 `record_summary` 工具來描述圖片。
5. 當請求發送到 Claude 3 模型時，它會分析圖片並根據指定的 `input_schema` 生成 JSON 摘要。模型提取主要顏色，提供簡短描述，並估計拍攝年份（如果適用）。
6. 生成的 JSON 摘要作為參數傳遞給 `record_summary` 工具，提供圖片主要特徵的結構化表示。

通過使用具有明確定義的 `input_schema` 的 `record_summary` 工具，我們可以獲得圖片的結構化 JSON 摘要，而無需依賴純文本提取。這種方法確保輸出遵循一致的格式，並且可以被應用程序的下游組件輕鬆解析和處理。

在人工智能驅動的應用程序中，強制函數呼叫並指定預期輸出結構的能力是工具使用的一個強大特性。它使開發人員能夠更好地控制生成的輸出，並簡化了將 AI 生成的數據整合到應用程序工作流程中的過程。

函數執行

你已經定義了函數，並提示了你的 AI，而 AI 決定它應該呼叫你的某個函數。現在是時候讓你的應用程式代碼或函式庫（如果你使用的是像 [raix-rails](#) 這樣的 Ruby gem）將函數呼叫及其參數分派到你的應用程式代碼中的相應實現。

你的應用程式代碼決定如何處理函數執行的結果。可能需要做的事情包括在 `lambda` 中的單行代碼，或者可能涉及呼叫外部 API。可能涉及呼叫另一個 AI 組件，或者可能涉及系統其餘部分中的數百甚至數千行代碼。這完全取決於你。

有時函數呼叫就是操作的終點，但如果結果代表了需要 AI 繼續處理的思維鏈中的信息，那麼你的應用程式代碼需要將執行結果插入到聊天記錄中，讓 AI 繼續處理。

例如，這裡是 Olympia 的 `AccountManager` 用於客戶服務智能工作流程編排中與客戶溝通的 [Raix](#) 函數聲明。

```
1  class AccountManager
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    # lots of other functions...
6
7    function :notify_account_owner,
8      "Don't share UUID. Mention dollars if subscription changed",
9      message: { type: "string" } do |arguments|
10      account.owner.freeform_notify(
11        subject: "Account Change Notification",
12        message: arguments[:message]
13      )
14      "Notified account owner"
15    end
```

這裡發生的事情可能不是很容易理解，所以讓我來分解說明。

1. `AccountManager` 類定義了許多與帳戶管理相關的功能。它可以更改您的方案、新增和移除團隊成員，以及其他功能。
2. 其頂層指令告訴 `AccountManager` 應該使用 `notify_account_owner` 函數將帳戶更改請求的結果通知帳戶擁有者。
3. 該函數的簡明定義包括：
 - 名稱
 - 描述
 - 參數 `message: { type: "string" }`
 - 當函數被調用時要執行的程式區塊

在使用函數區塊的結果更新對話記錄後，會再次調用 `chat_completion` 方法。此方法負責將更新後的對話記錄發送回 AI 模型以進行進一步處理。我們將此過程稱為**對話循環**。

當 AI 模型收到包含更新記錄的新聊天完成請求時，它可以訪問先前執行的函數結果。它可以分析這些結果，將其納入決策過程中，並根據對話的累積上下文生成下一個回應或動作。它可以根據更新後的上下文選擇執行額外的函數，或者如果確定不需要進一步的函數調用，就可以對原始提示生成最終回應。

原始提示的可選續處理

當您將工具結果發送回 LLM 並繼續處理原始提示時，AI 會使用這些結果來調用額外的函數或生成最終的純文本回應。



某些模型，如 Cohere 的 **Command-R**，可以在其回應中引用它們使用的具體工具，提供額外的透明度和可追蹤性。

根據使用的模型，函數調用的結果將存在於具有特殊角色的對話記錄消息中，或以其他語法形式反映。但重要的是這些數據要在對話記錄中，這樣 AI 在決定下一步行動時就能考慮到這些數據。



一個常見（且可能代價高昂）的錯誤情況是在繼續聊天之前忘記將函數結果添加到對話記錄中。結果，AI 會以與第一次調用函數之前基本相同的方式被提示。換句話說，就 AI 而言，它還沒有調用該函數。所以它會再次調用。然後又一次。如此反覆，直到你中斷它。希望你的上下文不要太大，你的模型不要太貴！

工具使用的最佳實踐

要充分利用工具，請考慮以下最佳實踐。

描述性定義

為每個工具及其輸入參數提供清晰且具描述性的名稱和說明。這有助於 LLM 更好地理解每個工具的用途和功能。

根據我的經驗，我可以告訴你“命名很難”這句常見的智慧在這裡也適用；我曾看到僅僅通過更改函數名稱或描述的措辭，就能讓 LLM 產生截然不同的結果。有時候移除描述反而能提升性能。

工具結果處理

將工具結果傳回 LLM 時，確保它們結構良好且全面。使用有意義的鍵和值來表示每個工具的輸出。嘗試不同的格式，看看哪種效果最好，從 JSON 到純文本都可以。

結果解釋器通過使用 AI 來分析結果並提供人性化的解釋、摘要或關鍵要點來解決這個挑戰。

錯誤處理

實施健全的錯誤處理機制，以處理 LLM 可能生成無效或不支援的工具調用輸入參數的情況。優雅地處理並從工具執行過程中可能發生的任何錯誤中恢復。

AI 的一個極其好的特質是它能理解錯誤訊息！這意味著如果你正在以快速且粗略的心態工作，你可以簡單地捕獲工具實現中產生的任何異常，並將其傳回給 AI，這樣它就知道發生了什麼！

例如，這是 Olympia 中 Google 搜索實現的精簡版本：

```
1  def google_search(conversation, params)
2      conversation.update_cstatus("Searching Google...")
3      query = params[:query]
4      search = GoogleSearch.new(query).get_hash
5
6      conversation.update_cstatus("Summarizing results...")
7      SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
8  rescue StandardError => e
9      Honeybadger.notify(e)
10     { error: e.message }.inspect
11 end
```

在 Olympia 中的 Google 搜尋是一個兩步驟的過程。首先執行搜尋，然後總結結果。如果發生任何失敗，無論是什麼原因，異常訊息都會被打包並送回給 AI。這個技術是幾乎所有智能錯誤處理模式的基礎

舉例來說，假設GoogleSearch API 呼叫因為 503 服務無法使用異常而失敗。這個異常會冒泡到最上層的異常處理，然後錯誤描述會作為函數調用的結果返回給

AI。AI 不會只是給用戶一個空白畫面或技術性錯誤，而是會說類似「抱歉，我目前無法使用 Google 搜尋功能。如果您願意的話，我可以稍後再試。」這樣的話。

這可能看起來只是個巧妙的技巧，但考慮另一種錯誤情況：當 AI 呼叫外部 API 且能直接控制傳遞給 API 的參數時。也許它在生成這些參數時出錯了？只要外部 API 提供的錯誤訊息足夠詳細，將錯誤訊息傳回給呼叫方的 AI 就意味著它可以重新考慮這些參數並再試一次。這是自動的過程，不管錯誤是什麼。

現在想想在一般程式碼中實現這種強大的錯誤處理需要做多少工作。這幾乎是不可能的。

迭代優化

如果LLM沒有推薦適當的工具或產生次優的回應，請對工具定義、描述和輸入參數進行迭代。根據觀察到的行為和期望的結果，持續優化和改進工具設置。

1. 從簡單的工具定義開始：首先定義具有清晰簡潔的名稱、描述和輸入參數的工具。初期避免過度複雜化工具設置，專注於核心功能。例如，如果您想要保存情感分析的結果，可以從這樣的基本定義開始：

```
1  {
2    "name": "save_sentiment_score",
3    "description": "Analyze user-provided text and generate sentiment score",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "score": {
8          "type": "float",
9          "description": "sentiment score from -1 (negative) to 1 (positive)"
10       }
11   }
```

```
11     },  
12     "required": ["score"]  
13 }  
14 }
```

2. 測試和觀察：一旦完成初步的工具定義設置，請使用不同的提示詞來測試並觀察大型語言何與工具互動。密切關注生成回應的品質和相關性。如果大型語言模型產生次優回應，就是時候改進工具定義了。
3. 改進描述：如果大型語言模型誤解了工具的用途，試著改進工具的描述。提供更多上下文、範例或說明，以引導大型語言模型有效使用工具。例如，你可以更新情緒分析工具的描述，使其更具體地針對文本的情感基調進行分析：

```
1 {  
2     "name": "save_sentiment_score",  
3     "description": "Determine the overall emotional tone of a piece of text,  
4     such as customer reviews, social media posts, or feedback comments.",  
5     ...  
6 }
```

4. 調整輸入參數：如果大型語言模型正在生成無效或不相關的輸入參數，請考慮調整參數定義。添加更具體的限制條件、驗證規則或示例，以闡明預期的輸入格式。
5. 基於反饋進行迭代：持續監控工具的性能並收集用戶或利益相關者的反饋。利用這些反饋來識別需要改進的領域，並對工具定義進行迭代改進。例如，如果用戶反映分析無法很好地處理諷刺語氣，您可以在描述中添加說明：


```
1 {  
2   "name": "save_sentiment_score",  
3   "description": "Analyze the sentiment of a given text and return a sentiment  
4     score between -1 (negative) and 1 (positive). Note: Sarcasm should be  
5     considered negative.",  
6   ...  
7 }
```

通過根據觀察到的行為和反饋迭代改進您的工具定義，您可以逐步提升 AI 驅動應用程式的性能和效果。請記住要保持工具定義清晰、簡潔，並專注於特定的任務。定期測試和驗證工具互動，確保它們符合您期望的結果。

組合和鏈接工具

到目前為止僅提及的工具使用最強大的特性之一是能夠將多個工具組合和鏈接在一起以完成複雜任務。通過仔細設計工具定義及其輸入/輸出格式，您可以創建可以以各種方式組合的可重用構建塊。

讓我們考慮一個例子，您正在為 AI 驅動的應用程式建立數據分析管道。您可能會有以下工具：

1. **DataRetrieval**：一個根據指定條件從數據庫或 API 獲取數據的工具。
2. **DataProcessing**：一個對檢索到的數據執行計算、轉換或聚合的工具。
3. **DataVisualization**：一個以用戶友好的格式（如圖表或圖形）呈現處理後數據的工具。

通過將這些工具鏈接在一起，您可以創建一個強大的工作流程，用於檢索相關數據、處理數據並以有意義的方式呈現結果。以下是工具使用工作流程的可能樣貌：

1. 大型語言模型（LLM）接收用戶查詢，要求提供特定產品類別的銷售數據洞察。
2. LLM 選擇 `DataRetrieval` 工具並生成適當的輸入參數，以從數據庫中獲取相關的銷售數據。
3. 檢索到的數據被“傳遞”給 `DataProcessing` 工具，該工具計算總收入、平均銷售價格和增長率等指標。
4. 處理後的數據隨後由 `DataVisualization` 工具處理，該工具創建一個視覺吸引力的圖表或圖形來表示這些洞察，並將圖表的 URL 傳回給 LLM。
5. 最後，LLM 使用 `markdown` 生成格式化的用戶查詢回應，整合視覺化數據並提供關鍵發現的摘要。

通過組合這些工具，您可以創建一個可以輕鬆整合到應用程式中的無縫數據分析工作流程。這種方法的優點在於每個工具都可以獨立開發和測試，然後以不同方式組合來解決各種問題。

為了實現工具的順暢組合和鏈接，明確定義每個工具的輸入和輸出格式很重要。

例如，`DataRetrieval` 工具可能接受數據庫連接詳細信息、表名和查詢條件等參數，並將結果集作為結構化 JSON 對象返回。然後，`DataProcessing` 工具可以將此 JSON 對象作為輸入，並產生轉換後的 JSON 對象作為輸出。通過標準化工具之間的數據流，您可以確保兼容性和可重用性。

在設計您的工具生態系統時，要思考如何組合不同的工具來應對應用程式中的常見用例。考慮創建封裝常見工作流程或業務邏輯的高層級工具，使 LLM 更容易選擇和有效使用它們。

請記住，工具使用的力量在於其提供的靈活性和模組化。通過將複雜任務分解為更小的、可重用的工具，您可以創建一個強大且適應性強的 AI 驅動應用程式，能夠應對各種挑戰。

未來方向

隨著 AI 驅動應用程式開發領域的發展，我們可以期待工具使用能力的進一步提升。一些潛在的未來方向包括：

1. **多跳工具使用**：LLM 可能能夠決定需要使用工具的次數，以生成令人滿意的回應。這可能涉及基於中間結果的多輪工具選擇和執行。
2. **預定義工具**：AI 平台可能提供一組開發者可以開箱即用的預定義工具，如 Python 解釋器、網絡搜索工具或常用實用函數。
3. **無縫整合**：隨著工具使用變得更加普遍，我們可以期待 AI 平台與流行開發框架之間的更好整合，使開發者更容易將工具使用整合到他們的應用程式中。

工具使用是一種強大的技術，使開發者能夠在 AI 驅動的應用程式中充分發揮 LLM 的潛力。通過將 LLM 連接到外部工具和資源，您可以創建更具動態性、智能性和情境感知能力的系統，能夠適應用戶需求並提供有價值的洞察和行動。

雖然工具使用提供了巨大的可能性，但重要的是要意識到潛在的挑戰和考慮因素。一個關鍵方面是管理工具互動的複雜性，並確保整個系統的穩定性和可靠性。您需要處理工具調用可能失敗、返回意外結果或影響性能的情況。此外，您應該考慮安全和訪問控制措施，以防止未經授權或惡意使用工具。適當的錯誤處理、日誌記錄和監控機制對於維護 AI 驅動應用程式的完整性和性能至關重要。

在探索在您自己的專案中運用工具的可能性時，請記住要從明確的目標開始，設計結構完善的工具定義，並根據反饋和結果進行迭代。採用正確的方法和思維模式，工具使用可以為您的人工智能驅動的應用程式開啟創新和價值的新境界。

串流處理



透過 HTTP 進行串流資料處理，也稱為伺服器推送事件 (SSE)，是一種伺服器在資料可用時持續向客戶端發送資料的機制，無需客戶端明確請求。由於 AI 的回應是逐步生成的，透過即時顯示 AI 的輸出來提供響應式的使用者體驗是很合理的做法。事實上，據我所知，所有 AI 提供商的 API 都在其完成端點中提供了串流回應作為選項。

本章出現在[使用工具](#)之後的原因，是因為將工具的使用與即時 AI 回應相結合能夠產生強大的效果。這樣的組合允許動態和互動式的體驗，其中 AI 可以處理使用者輸入、自行運用各種工具和函式，然後提供即時回應。

要實現這種無縫互動，你需要編寫串流處理器，它們能夠分派 AI 調用的工具函式以及向終端使用者輸出純文字。在處理工具函式後需要進行循環這一點為這項工作增添了一個有趣的挑戰。

實作 ReplyStream

為了展示如何實現串流處理，本章將深入探討 Olympia 中使用的 ReplyStream 類別的簡化版本。這個類別的實例可以作為 [ruby-openai](#) 和 [openrouter](#) 等 AI 客戶端函式庫中的 stream 參數傳遞。

以下是我在 Olympia 的 PromptSubscriber 中使用 ReplyStream 的方式，它通過 Wisper 監聽新使用者訊息的建立。

```
1  class PromptSubscriber
2    include Raix::ChatCompletion
3    include Raix::PromptDeclarations
4
5    # many other declarations omitted...
6
7    prompt text: -> { user_message.content },
8                stream: -> { ReplyStream.new(self) },
9                until: -> { bot_message.complete? }
10
11   def message_created(message) # invoked by Wisper
12     return unless message.role.user? && message.content?
13
14     # rest of the implementation omitted...
```

除了有一個指向建立它的提示訂閱者的 context 參考外，ReplyStream 類別還具有用於儲存接收資料緩衝區的實例變數，以及用於追蹤串流處理期間所呼叫的函式名稱和參數的陣列。

```
1 class ReplyStream
2   attr_accessor :buffer, :f_name, :f_arguments, :context
3
4   delegate :bot_message, :dispatch, to: :context
5
6   def initialize(context)
7     self.context = context
8     self.buffer = []
9     self.f_name = []
10    self.f_arguments = []
11  end
12
13  def call(chunk, bytesize = nil)
14    # ...
15  end
16
17  # ...
18 end
```

`initialize` 方法設置 `ReplyStream` 實例的初始狀態，初始化緩衝區、上下文和其他變數。

`call` 方法是處理串流資料的主要入口點。它接收一個資料區塊（以雜湊形式表示）和一個可選的 `bytesize` 參數，在我們的例子中這個參數未被使用。在這個方法內部，類別使用模式匹配來根據接收到的資料區塊結構處理不同的情況。



對資料區塊呼叫 `deep_symbolize_keys` 可以讓模式匹配更優雅，因為這樣我們就能操作符號而不是字串。

```
1 def call(chunk, _bytesize)
2     case chunk.deep_symbolize_keys
3
4     in { # match function name
5         choices: [
6             {
7                 delta: {
8                     tool_calls: [
9                         { index: index, function: {name: name} }
10                    ]
11                }
12            }
13        ] }
14
15     f_name[index] = name
```

我們要匹配的第一個模式是工具呼叫及其相關的函式名稱。如果偵測到一個工具呼叫，我們就將它存入 `f_name` 陣列中。我們將函式名稱存儲在索引陣列中，因為模型能夠執行平行函式呼叫，同時發送多個函式來執行。

平行函式呼叫是 AI 模型同時執行多個函式呼叫的能力，允許這些函式呼叫的效果和結果能平行處理。這在函式執行時間較長的情況下特別有用，並且可以減少與 API 的往返通訊次數，進而可以節省大量的權杖用量。

接下來我們需要匹配對應於函式呼叫的引數。

```
1  in { # match arguments
2    choices: [
3      {
4        delta: {
5          tool_calls: [
6            {
7              index: index, function: {arguments: argument }
8            }
9          ]
10       }
11     ]
12   ]}
13
14   f_arguments[index] ||= "" # initialize if not already
15   f_arguments[index] << argument
```

與我們處理函數名稱的方式類似，我們將參數存儲在索引陣列中。

接下來，我們關注面向用戶的一般訊息，這些訊息將從伺服器一個字符標記 (token) 一個字符標記地到達，並被指派給 `new_content` 變數。我們還需要留意 `finish_reason`。在輸出序列的最後一個區塊之前，它都將保持為 `nil`。


```
1   in {
2     choices: [
3       { delta: {content: new_content}, finish_reason: finish_reason }
4     ]}
5
6     # you could transmit every chunk to the user here...
7     buffer << new_content.to_s
8
9     if finish_reason.present?
10      finalize
11    elsif new_content.to_s.match?(/\n\n/)
12      send_to_client # ...or buffer and transmit once per paragraph
13    end
```

重要的是，我們添加了一個模式匹配表達式來處理 AI 模型提供者發送的錯誤訊息。在本地開發環境中，我們會拋出異常，但在生產環境中，我們則會記錄錯誤並完成處理。

```
1   in { error: { message: } }
2     if Rails.env.local?
3       raise message
4     else
5       Honeybadger.notify("AI Error: #{message}")
6       finalize
7     end
```

case 語句中最後的 else 子句會在前面所有的模式都不匹配時執行。這只是一個保護機制，這樣如果人工智慧模型開始發送我們無法識別的區塊時，我們就能夠察覺。

```
1  else
2      Honeybadger.notify("Unrecognized Chunk: #{chunk}")
3  end
4  end
```

`send_to_client` 方法負責將緩衝區的內容發送給客戶端。它會檢查緩衝區是否為空、更新機器人訊息內容、渲染機器人訊息，並將內容儲存在資料庫中以確保資料持久性。

```
1  def send_to_client
2      # no need to process pure whitespace
3      return if buffer.join.squish.blank?
4
5      # set the buffer content on the bot message
6      content = buffer.join
7      bot_message.content = content
8
9      # save to database so that we never lose data
10     # even if the stream doesn't terminate correctly
11     bot_message.update_column(:content, content)
12
13     # update content via websocket
14     ConversationRenderer.update(bot_message)
15 end
```

當串流處理完成時，會呼叫 `finalize` 方法。如果在串流期間收到任何函式呼叫，該方法會進行函式派發，使用最終內容和其他相關資訊更新機器人訊息，並重置函式呼叫歷史記錄。

```
1  def finalize
2    if f_name.any?
3      f_name.each_with_index do |name, index|
4        # takes care of calling the function wherever it's implemented
5        dispatch(name:, arguments: JSON.parse(f_arguments[index]))
6      end
7
8      # reset the function call history
9      f_name.clear
10     f_arguments.clear
11   else
12     content = buffer.join.presence
13     bot_message.update!(content:, complete: true)
14     ConversationRenderer.update(bot_message)
15   end
16 end
```

如果模型決定要呼叫函式，你需要「分派」該函式呼叫（名稱和參數），使其得以執行，並且將`function_call` 和`function_result`訊息加入到對話紀錄中。

根據我的經驗，最好在程式碼庫的同一個地方處理函式訊息的建立，而不是依賴於工具的實作。這樣不僅更整潔，還有一個非常重要的實務原因：如果人工智慧模型呼叫了一個函式，但在你進行下一輪對話時，在紀錄中看不到相應的呼叫和結果訊息，那麼它會重複呼叫同一個函式。這可能會無限循環下去。請記住，人工智慧是完全無狀態的，所以除非你將這些函式呼叫的訊息回傳給它，否則這些呼叫就像從未發生過一樣。

```
1  # PromptSubscriber#dispatch
2
3  def dispatch(name:, arguments:)
4      # adds a function_call message to the conversation transcript
5      # plus dispatches to tool and returns result
6      conversation.function_call!(name, arguments).then do |result|
7          # add function result message to the transcript
8          conversation.function_result!(name, result)
9      end
10 end
```



在派發後清除函式呼叫歷史記錄與確保呼叫和結果都記錄在對話記錄中同樣重要，這樣你就不會在每次循環時重複呼叫相同的函式。

「對話循環」

在 PromptSubscriber 類別中，我們使用來自 PromptDeclarations 模組的 prompt 方法來定義對話循環的行為。until 參數被設定為 -> { bot_message.complete? }，這表示循環會持續到 bot_message 被標記為完成為止。

```
1  prompt text: -> { user_message.content },
2      stream: -> { ReplyStream.new(self) },
3      until: -> { bot_message.complete? }
```



但是 bot_message 是在什麼時候被標記為完成的呢？如果你忘記了，請回顧 finalize 方法的第 13 行。

讓我們回顧整個串流處理邏輯。

1. `PromptSubscriber` 通過 `message_created` 方法接收新的使用者訊息，這個方法是由 `Wisper` 發佈/訂閱系統在終端使用者建立新提示時觸發的。
2. `prompt` 類別方法以宣告式方式定義了 `PromptSubscriber` 的聊天完成邏輯。AI 模型將使用使用者的訊息內容執行聊天完成，並將新的 `ReplyStream` 實例作為串流參數，以及指定的循環條件。
3. AI 模型處理提示並開始生成回應。當回應以串流方式傳輸時，`ReplyStream` 實例的 `call` 方法會針對每個資料區塊被呼叫。
4. 如果 AI 模型決定呼叫工具函數，函數名稱和參數會從區塊中提取並分別存儲在 `f_name` 和 `f_arguments` 陣列中。
5. 如果 AI 模型生成面向使用者的內容，這些內容會被緩衝並通過 `send_to_client` 方法發送給客戶端。
6. 一旦串流處理完成，`finalize` 方法就會被呼叫。如果在串流期間有任何工具函數被呼叫，它們會使用 `PromptSubscriber` 的 `dispatch` 方法進行分派。
7. `dispatch` 方法會在對話紀錄中添加一個 `function_call` 訊息，執行相應的工具函數，並在紀錄中添加一個包含函數呼叫結果的 `function_result` 訊息。
8. 在分派工具函數後，函數呼叫歷史記錄會被清除，以防止在後續循環中重複呼叫函數。
9. 如果沒有呼叫工具函數，`finalize` 方法會使用最終內容更新 `bot_message`，將其標記為完成，並將更新後的訊息發送給客戶端。
10. 循環條件 -> `{ bot_message.complete? }` 會被評估。如果 `bot_message` 未被標記為完成，循環會繼續，並且原始提示會與更新後的對話紀錄一起重新提交。
11. 步驟 3-10 會重複執行，直到 `bot_message` 被標記為完成，表示 AI 模型已完成生成回應，且不需要執行更多工具函數。

通過實現這個對話循環，你能讓 AI 模型與應用程式進行來回互動，根據需要執行工具函數，並生成面向使用者的回應，直到對話自然結束。

串流處理和對話循環的結合實現了動態且互動式的 AI 驅動體驗，其中 AI 模型可以處理使用者輸入、使用各種工具和函數，並根據不斷演變的對話上下文提供即時回應。

自動續接

重要的是要了解 AI 輸出的限制。大多數模型在單次回應中能生成的權杖數量都有上限，這個上限由 `max_tokens` 參數決定。如果 AI 模型在生成回應時達到這個限制，它會突然停止並指出輸出被截斷。

在來自 AI 平台 API 的串流回應中，你可以通過檢查區塊中的 `finish_reason` 變數來偵測這種情況。如果 `finish_reason` 被設置為 "length"（或特定於模型的其他關鍵值），這表示模型在生成過程中達到了最大權杖限制，輸出被提前切斷。

處理這種情況的一種優雅方式，同時提供無縫的使用者體驗，是在串流處理邏輯中實現自動續接機制。通過為長度相關的完成原因添加模式匹配，你可以選擇循環並自動從中斷處繼續輸出。

這裡是一個刻意簡化的例子，展示如何修改 `ReplyStream` 類別中的 `call` 方法以支援自動續接：

```
1  LENGTH_STOPS = %w[length MAX_TOKENS]
2
3  def call(chunk, _bytesize)
4    case chunk.deep_symbolize_keys
5      # ...
6
7    in {
8      choices: [
9        { delta: {content: new_content},
10          finish_reason: finish_reason } ] }
```

```
11
12     buffer << new_content.to_s
13
14     if finish_reason.blank?
15         send_to_client if new_content.to_s.match?(/\n\n/)
16     elsif LENGTH_STOPS.include?(finish_reason)
17         continue_cutoff
18     else
19         finalize
20     end
21
22     # ...
23 end
24 end
25
26 private
27
28 def continue_cutoff
29     conversation.bot_message!(buffer.join, visible: false)
30     conversation.user_message!("please continue", visible: false)
31     bot_message.update_column(:created_at, Time.current)
32 end
```

在這個修改版本中，當 `finish_reason` 顯示輸出被截斷時，我們不會結束串流，而是在不完成對話記錄的情況下添加一對訊息，通過更新原始面向用戶回應訊息的 `created_at` 屬性將其移至對話記錄的“底部”，然後讓循環繼續進行，使 AI 能夠從中斷處繼續生成內容。

請記住，AI 完成端點是無狀態的。它只“知道”你通過對話記錄告訴它的內容。在這種情況下，我們通過向對話記錄添加（對終端用戶）“不可見”的訊息來告

訴 AI 它被截斷了。不過請記住，這是一個刻意簡化的示例。真實的實現需要進行更多的對話記錄管理，以確保我們不會浪費權杖數，和/或因對話記錄中重複的助理訊息而使 AI 混淆。

自動續接的真實實現還應該包含所謂的“斷路器邏輯”，以防止失控循環。原因是，在某些類型的用戶提示和較低的 `max_tokens` 設置下，AI 可能會無限循環生成面向用戶的輸出。

請記住，每次循環都需要一個單獨的請求，而且每個請求都會再次消耗你的整個對話記錄。在決定是否在應用程式中實現自動續接時，你應該認真考慮用戶體驗和 API 使用量之間的權衡。自動續接特別容易產生危險的高額費用，尤其是在使用高級商業模型時。

結論

串流處理是構建 AI 驅動應用程式的關鍵方面，它將工具使用與即時 AI 回應相結合。通過高效處理來自 AI 平台 API 的串流數據，你可以提供無縫和互動的用戶體驗，處理大型回應，優化資源使用，並優雅地處理錯誤。

提供的 `Conversation::ReplyStream` 類別展示了如何在 Ruby 應用程式中使用模式匹配和事件驅動架構實現串流處理。通過理解和利用串流處理技術，你可以釋放 AI 整合在應用程式中的全部潛力，並提供強大而吸引人的用戶體驗。

自我修復數據



自我修復數據是一種強大的方法，通過利用大型語言模型（LLMs）的能力來確保應用程式中數據的完整性、一致性和品質。這類模式的重點在於使用 AI 自動檢測、診斷和修正數據異常、不一致或錯誤，從而減輕開發人員的負擔並維持高水準的數據可靠性。

從本質上來說，自我修復數據模式認識到數據是任何應用程式的命脈，確保其準確性和完整性對於應用程式的正常運作和用戶體驗至關重要。然而，隨著應用程式規模和複雜度的增長，管理和維護數據品質可能是一項複雜且耗時的任務。這就是 AI 發揮作用的地方。

在自我修復數據模式中，AI 工作器被用來持續監控和分析您應用程式的數據。這些模型具有理解和解讀數據中的模式、關係和異常的能力。通過利用其自然語言處理和理解能力，它們可以識別數據中的潛在問題或不一致，並採取適當的措施來糾正它們。

自我修復數據的過程通常包含幾個關鍵步驟：

1. **數據監控：** AI 工作器持續監控應用程式的數據流、數據庫或儲存系統，尋找任何異常、不一致或錯誤的跡象。或者，您可以在遇到異常時啟動 AI 組件。
2. **異常檢測：** 當發現問題時，AI 工作器詳細分析數據以確定問題的具體性質和範圍。這可能包括檢測缺失值、不一致的格式，或違反預定規則或約束的數據。
3. **診斷和修正：** 一旦確定問題，AI 工作器利用其對數據領域的知識和理解來決定適當的行動方案。這可能包括自動修正數據、填補缺失值，或在必要時標記問題以供人工干預。
4. **持續學習（可選，取決於使用案例）：** 當您的 AI 工作器遇到並解決各種數據問題時，它可以輸出描述發生了什麼以及如何回應的資訊。這些元數據可以被輸入到學習過程中，使您（以及可能通過微調的底層模型）能夠隨著時間推移在識別和解決數據異常方面變得更加有效和高效。

通過自動檢測和修正數據問題，您可以確保您的應用程式運行在高品質、可靠的數據基礎上。這減少了錯誤、不一致或數據相關錯誤影響應用程式功能或用戶體驗的風險。

一旦您有了 AI 工作器處理數據監控和修正的任務，您就可以將精力集中在應用程式的其他關鍵方面。這節省了原本需要花在手動數據清理和維護上的時間和資源。事實上，隨著您的應用程式規模和複雜度的增長，手動管理數據品質變得越來越具有挑戰性。“自我修復數據”模式通過利用 AI 的力量來處理大量數據並實時檢測問題，可以有效地擴展。



由於其本質，AI 模型可以隨著時間推移適應變化的數據模式、架構或需求，幾乎不需要監督。只要他們的指令提供足夠的指導，特別是關於預期結果的指導，您的應用程式可能能夠演化並處理新的數據場景，而無需大量的手動干預或代碼更改。

自我修復數據模式與我們討論過的其他模式類別（如“多重工作器”）很好地協調。自我修復數據功能可以被視為一種專門專注於確保數據品質和完整性的特殊工作器。這種工作器與其他 AI 工作器一起運作，各自為應用程式的不同功能方面做出貢獻。

在實踐中實施自我修復數據模式需要仔細設計並將 AI 模型整合到應用程式架構中。由於數據丟失和損壞的風險，您應該為如何使用這項技術定義明確的指導原則。您還應該考慮性能、可擴展性和數據安全性等因素。

實際案例研究：修復損壞的 JSON

利用自我修復數據最實用和最方便的方法之一也很容易解釋：修復損壞的 JSON。

這種技術可以應用於處理 LLMs 生成的不完美或不一致數據（如損壞的 JSON）這一常見挑戰，並提供了自動檢測和修正這些問題的方法。

在 Olympia 中，我經常遇到大型語言模型生成的 JSON 資料不完全有效的情況。這種情況可能由多種原因造成，例如大型語言模型在實際的 JSON 程式碼前後添加註解，或引入像是遺漏逗號或未轉義雙引號等語法錯誤。這些問題可能導致解析錯誤，並造成應用程式功能的中斷。

為了解決這個問題，我實作了一個實用的解決方案，以 JsonFixer 類別的形式呈現。這個類別體現了「自我修復資料」模式，它接收損壞的 JSON 作為輸入，並運用大型語言模型來修復它，同時盡可能保留原有的資訊和意圖。

```
1  class JsonFixer
2    include Raix::ChatCompletion
3
4    def call(bad_json, error_message)
5      raise "No data provided" if bad_json.blank? || error_message.blank?
6
7      transcript << {
8        system: "Consider user-provided JSON that generated a parse
9                exception. Do your best to fix it while preserving the
10               original content and intent as much as possible." }
11      transcript << { user: bad_json }
12      transcript << { assistant: "What is the error message?"}
13      transcript << { user: error_message }
14      transcript << { assistant: "Here is the corrected JSON\n```\njson\n" }
15
16      self.stop = ["```\n"]
17
18      chat_completion(json: true)
19    end
20
21    def model
22      "mistralai/mixtral-8x7b-instruct:nitro"
23    end
24  end
```



請注意 `JsonFixer` 如何使用 `Ventriloquist` 來引導 AI 的回應。

JSON 資料的自我修復程序如下：

1. **JSON 生成：**使用 LLM 根據特定提示或要求生成 JSON 資料。然而，由於 LLM 的特性，生成的 JSON 可能並非總是完全有效。當然，如果您提供無效的 JSON 給解析器，它會引發 `ParserError`。

```
1 begin
2   JSON.parse(llm_generated_json)
3 rescue JSON::ParserError => e
4   JsonFixer.new.call(llm_generated_json, e.message)
5 end
```

請注意，異常訊息也會傳遞給 `JSONFixer` 呼叫，這樣它就不需要完全假設數據出了什麼問題，特別是因為解析器通常會準確地告訴你哪裡出錯了。

2. **基於大語言模型的修正：**`JSONFixer` 類別會將損壞的 JSON 連同特定的提示或指令一起發送回大語言模型，要求在盡可能保留原始資訊和意圖的同時修復 JSON。這個經過大量數據訓練且理解 JSON 語法的大語言模型，會嘗試修正錯誤並生成有效的 JSON 字串。我們使用[回應圍欄](#)來限制大語言模型的輸出，並選擇 Mixtral 8x7B 作為 AI 模型，因為它特別適合這類任務。
3. **驗證和整合：**由於呼叫了 `chat_completion(json: true)`，大語言模型返回的修復後的 JSON 字串會由 `JSONFixer` 類別本身進行解析。如果修復後的 JSON 通過驗證，它就會被整合回應用程式的工作流程中，使應用程式能夠無縫地繼續處理數據。損壞的 JSON 已經被“修復”了。

雖然我已經多次編寫和重寫自己的 `JSONFixer` 實現，但我懷疑所有這些版本的總投入時間不會超過一兩個小時。

請注意，保持意圖是任何自我修復數據模式的關鍵要素。基於大語言模型的修正過程旨在盡可能保留生成的 JSON 的原始資訊和意圖。這確保了修復後的 JSON 能保持其語義含義，並能在應用程式的上下文中有效使用。

這個在 Olympia 中實現的“自我修復數據”方法的實際應用，清楚地展示了如何利用 AI（特別是大語言模型）來解決現實世界的數據挑戰。它展現了將傳統程式設計技術與 AI 功能相結合來建構強大且高效應用程式的力量。

波斯特爾法則與“自我修復數據”模式

“自我修復數據”（如 JSONFixer 類別所展示的）與波斯特爾法則（也稱為魯棒性原則）有著很好的契合。波斯特爾法則指出：

“對於自己要做的事要保守，對於接受他人的東西要寬容。”

這個原則最初由互聯網先驅 Jon Postel 提出，強調了建立系統時要能容忍多樣化甚至略有錯誤的輸入，同時在發送輸出時要嚴格遵守指定協議的重要性。

在“自我修復數據”的背景下，JSONFixer 類別體現了波斯特爾法則，它能寬容地接受由大語言模型生成的損壞或不完善的 JSON 數據。當遇到不完全符合預期格式的 JSON 時，它不會立即拒絕或失敗。相反，它採取了一種包容的方法，嘗試使用大語言模型的能力來修復 JSON。

通過寬容地接受不完善的 JSON，JSONFixer 類別展示了其魯棒性和靈活性。它承認現實世界中的數據經常以各種形式出現，可能並不總是符合嚴格的規範。通過優雅地處理和修正這些偏差，該類別確保了應用程式能夠在面對不完善數據時仍然順暢運行。

另一方面，JSONFixer 類別在輸出方面也遵守了波斯特爾法則的保守原則。在使用大語言模型修復 JSON 之後，該類別會驗證修正後的 JSON，以確保它嚴格符合預期格式。在將數據傳遞給應用程式的其他部分之前，它維護了數據的完整性和正確性。這種保守的方法保證了 JSONFixer 類別的輸出是可靠和一致的，促進了互操作性並防止錯誤的傳播。

關於 Jon Postel 的有趣知識：

- Jon Postel (1943-1998) 是一位美國計算機科學家，在互聯網的發展中發揮了關鍵作用。由於他對底層協議和標準的重大貢獻，他被稱為“互聯網之神”。
- Postel 是請求評議文件 (RFC) 系列的編輯，這是一系列關於互聯網的技術和組織說明。他撰寫或共同撰寫了超過 200 份 RFC，包括基礎協議如 TCP、IP 和 SMTP。
- 除了技術貢獻外，Postel 以其謙遜和協作的方式而聞名。他相信達成共識和共同努力建立強大且可互操作網絡的重要性。
- Postel 從 1977 年直到 1998 年英年早逝期間，擔任南加州大學 (USC) 信息科學研究所 (ISI) 計算機網絡部門主任。
- 為表彰他的巨大貢獻，Postel 在 1998 年 posthumously 獲得了享有盛譽的圖靈獎，這個獎項常被稱為“計算機界的諾貝爾獎”。

JSONFixer 類別促進了健壯性、靈活性和互操作性，這些都是 Postel 在其職業生涯中始終堅持的核心價值。通過建立能夠容忍缺陷同時又嚴格遵守協議的系統，我們可以創建出在面對現實世界挑戰時更具彈性和適應性的應用程序。

考量因素與禁忌症

自我修復數據方法的適用性完全取決於您的應用程序所處理的數據類型。這就是為什麼您可能不想簡單地使用猴子補丁來修改 `JSON.parse` 以自動修正應用程序中的所有 JSON 解析錯誤的原因：並非所有錯誤都能或應該被自動修正。

當涉及到與數據處理和處理相關的監管或合規要求時，自我修復特別令人擔憂。某些行業，如醫療保健和金融業，對數據完整性和可審核性有如此嚴格的規定，以至於在沒有適當監督或日誌記錄的情況下進行任何“黑盒”數據修正可能會違反這些規定。確保您設計的任何自我修復數據技術都符合適用的法律和監管框架至關重要。

應用自我修復數據技術，特別是涉及 AI 模型的技術，也可能對應用程序性能和資源利用率產生重大影響。通過 AI 模型處理大量數據以進行錯誤檢測和修正可能會消耗大量計算資源。評估自我修復數據的效益與相關的性能和資源成本之間的權衡非常重要。

話雖如此，讓我們深入探討決定何時何地應用這種強大方法所涉及的因素。

數據關鍵性

在考慮應用自我修復數據技術時，評估所處理數據的關鍵性至關重要。關鍵性水平指的是數據在您的應用程序和業務領域中的重要性和敏感性。

在某些情況下，自動修正數據錯誤可能並不適當，特別是當數據高度敏感或具有法律影響時。例如，考慮以下情況：

1. **金融交易：**在金融應用程序中，如銀行系統或交易平台，數據準確性至關重要。即使是金融數據中的微小錯誤也可能產生重大後果，如帳戶餘額錯誤、資金錯誤轉帳或錯誤的交易決策。在這些情況下，在沒有徹底驗證和審核的情況下進行自動修正可能會帶來不可接受的風險。
2. **醫療記錄：**醫療保健應用程序處理高度敏感和機密的患者數據。醫療記錄中的不準確可能會對患者安全和治療決策產生嚴重影響。在沒有合格醫療專業人員適當監督和驗證的情況下自動修改醫療數據可能違反監管要求並危及患者福祉。
3. **法律文件：**處理法律文件的應用程序，如合同、協議或法院文件，需要嚴格的準確性和完整性。即使是法律數據中的微小錯誤也可能產生重大法律影響。在這個領域進行自動修正可能並不適當，因為數據通常需要法律專家進行手動審查和驗證，以確保其有效性和可執行性。

在這些關鍵數據場景中，自動修正相關的風險通常超過潛在的效益。錯誤引入或不正確修改數據的後果可能很嚴重，導致財務損失、法律責任，甚至對個人造成傷害。

在處理高度關鍵的數據時，優先考慮手動驗證和確認流程至關重要。人工監督和專業知識對確保數據的準確性和完整性至關重要。自動自我修復技術仍然可以用於標記潛在的錯誤或不一致，但最終的修正決定應該包含人工判斷和批准。

然而，重要的是要注意，應用程序中並非所有數據都具有相同的關鍵性水平。在同一個應用程序中，可能存在敏感度較低或錯誤影響較小的數據子集。在這種情況下，自我修復數據技術可以選擇性地應用於這些特定的數據子集，而關鍵數據仍然需要進行手動驗證。

關鍵是要仔細評估應用程序中每個數據類別的關鍵性，並根據相關風險和影響定義明確的修正指導原則和流程。通過區分關鍵數據（如分類帳、醫療記錄）和非關鍵數據（如郵寄地址、資源警告），您可以在適當利用自我修復數據技術的效益和在必要時維持嚴格控制和監督之間取得平衡。

最終，是否對關鍵數據應用自我修復數據技術的決定應該與領域專家、法律顧問和其他相關利益相關者協商後做出。考慮應用程序數據的具體要求、法規和風險，並相應地調整數據修正策略至關重要。

錯誤嚴重程度

在應用自我修復數據技術時，評估數據錯誤的嚴重程度和影響很重要。並非所有錯誤都是同等的，適當的處理方式可能會根據問題的嚴重程度而有所不同。

輕微的不一致或格式問題可能適合自動修正。例如，負責修復損壞的 JSON 的自我修復數據工作器可以處理缺失的逗號或未轉義的雙引號，而不會顯著改變數據的含義或結構。這些類型的錯誤通常容易修正，對整體數據完整性的影響最小。

然而，對於那些從根本上改變數據含義或完整性的更嚴重錯誤，可能需要採取不同的處理方法。在這些情況下，自動修正可能並不足夠，需要人工干預來確保數據的準確性和有效性。

這就是運用人工智能本身來幫助判定錯誤嚴重程度的理念。通過利用人工智能模型的能力，我們可以設計自我修復數據工作器，不僅能夠糾正錯誤，還能評估這些錯誤的嚴重程度，並就如何處理這些錯誤做出明智的決策。

例如，讓我們考慮一個負責修正流入客戶數據庫的數據不一致問題的自我修復數據工作器。該工作器可以被設計用來分析數據並識別潛在的錯誤，如缺失或衝突的信息。不過，該工作器不會自動修正所有錯誤，而是可以配備額外的工具調用功能，使其能夠將嚴重錯誤標記出來供人工審查。

以下是實施這種方法的示例：

```
1  class CustomerDataReviewer
2      include Raix::ChatCompletion
3      include Raix::FunctionDeclarations
4
5      attr_accessor :customer
6
7      function :flag_for_review, reason: { type: "string" } do |params|
8          AdminNotifier.review_request(customer, params[:reason])
9      end
10
11     def initialize(customer)
12         self.customer = customer
13     end
14
15     def call(customer_data)
16         transcript << {
17             system: "You are a customer data reviewer. Your task is to identify
18                 and correct inconsistencies in customer data.
19
20                 < additional instructions here... >
```

```
21
22     If you encounter severe errors that require human review, use the
23     `flag_for_review` tool to flag the data for manual intervention." }
24
25     transcript << { user: customer.to_json }
26     transcript << { assistant: "Reviewed/corrected data:\n```\njson\n" }
27
28     self.stop = ["```\n"]
29
30     chat_completion(json: true).then do |result|
31       return if result.blank?
32
33       customer.update(result)
34     end
35   end
36 end
```

在這個例子中，CustomerDataHealer worker 被設計用來識別和修正客戶數據中的不一致性。我們再次使用**回應圍欄**和**傀儡師**來獲取結構化輸出。重要的是，worker 的系統指令包含了在遇到嚴重錯誤時使用 flag_for_review 函數的說明。當 worker 處理客戶數據時，它會分析數據並嘗試修正任何不一致之處。如果 worker 判定錯誤嚴重且需要人工干預，它可以使用 flag_for_review 工具來標記數據，並提供標記的原因。

chat_completion 方法被調用時設置了 json: true 以將修正後的客戶數據解析為 JSON。由於函數調用後沒有循環處理的機制，如果調用了 flag_for_review，結果將為空。否則，客戶數據將使用經過審查和可能修正的數據進行更新。

通過納入錯誤嚴重性評估和標記數據供人工審查的選項，自我修復數據 worker 變得更加智能和適應性強。它可以自動處理輕微錯誤，同時將嚴重錯誤升級給人類專家進行手動干預。

判定錯誤嚴重性的具體標準可以根據領域知識和業務需求在 worker 的指令中定義。在評估嚴重性時，可以考慮諸如對數據完整性的影響、數據丟失或損壞的可能性，以及不正確數據可能造成的後果等因素。

通過利用 AI 來評估錯誤嚴重性並提供人工干預的選項，自我修復數據技術可以在自動化和維護數據準確性之間取得平衡。這種方法確保了輕微錯誤能夠被高效修正，而嚴重錯誤則能得到人工審查者必要的關注和專業處理。

領域複雜性

在考慮應用自我修復數據技術時，評估數據領域的複雜性以及支配其結構和關係的規則非常重要。領域的複雜性可能會顯著影響自動數據修正方法的有效性和可行性。

當數據遵循明確定義的模式和約束時，自我修復數據技術效果最佳。在數據結構相對簡單且數據元素之間關係直接的領域中，自動修正可以以較高的置信度進行應用。例如，修正格式問題或強制執行基本數據類型約束通常可以由自我修復數據 worker 有效處理。

然而，隨著數據領域複雜性的增加，與自動數據修正相關的挑戰也隨之增長。在具有複雜業務邏輯、數據實體之間複雜關係，或特定領域規則和例外的領域中，自我修復數據技術可能無法完全把握其中的細微差別，並可能引入意外後果。

讓我們考慮一個複雜領域的例子：金融交易系統。在這個領域中，數據涉及各種金融工具、市場數據、交易規則和監管要求。不同數據元素之間的關係可能錯綜複雜，而支配數據有效性和一致性的規則可能高度特定於該領域。

在如此複雜的領域中，負責修正交易數據不一致性的自我修復數據 worker 需要深入理解特定領域的規則和約束。它需要考慮諸如市場法規、交易限制、風險計算和結算程序等因素。在這種情況下，自動修正可能無法完全把握領域的全部複雜性，並可能無意中引入錯誤或違反特定領域的規則。

為了應對領域複雜性帶來的挑戰，自我修復數據技術可以通過將特定領域的知識和規則納入 AI 模型和 worker 中來加強。這可以通過以下技術實現：

1. **領域特定訓練：**用於自我修復數據的 AI 模型可以在捕捉特定領域的細節和規則的數據集上進行指導甚至微調。通過接觸具有代表性的數據和場景，它們可以學習該領域特有的模式、約束和例外情況。
2. **基於規則的約束：**自我修復數據 worker 可以通過明確的基於規則的約束來增強，這些約束編碼了特定領域的知識。這些規則可以由領域專家定義並整合到數據修正過程中。AI 模型可以使用這些規則來指導其決策並確保符合特定領域的要求。
3. **與領域專家合作：**在複雜領域中，在設計和開發自我修復數據技術時讓領域專家參與進來至關重要。領域專家可以提供關於數據細節、業務規則和潛在邊界情況的寶貴見解。他們的知識可以通過**人機協作**模式納入 AI 模型和 worker 中，以提高自動數據修正的準確性和可靠性。
4. **漸進式和迭代方法：**在處理複雜領域時，採用漸進式和迭代方法進行自我修復數據通常是有益的。與其試圖一次性自動修正整個領域，不如專注於規則和約束明確的特定子領域或數據類別。隨著對領域理解的加深和技術的有效性得到證實，逐步擴大自我修復技術的範圍。

通過考慮數據領域的複雜性並將領域特定知識整合到自我修復數據技術中，您可以在自動化和準確性之間取得平衡。重要的是要認識到自我修復數據並非放之四海而皆準的解決方案，而是應該根據每個領域的具體要求和挑戰量身定制適當的方法。

在複雜的領域中，將自我修復數據技術與人類專業知識和監督相結合的混合方法可能最為有效。自動化修正可以處理常規和明確定義的情況，而複雜的場景或例外情況則可以標記出來供人工審查和干預。這種協作方法確保了在保持複雜數據領域必要控制和準確性的同時，實現自動化的效益。

可解釋性和透明度

可解釋性指的是理解和解讀人工智能模型決策背後推理的能力，而透明度則涉及為數據修正過程提供清晰的可見性。

在許多情況下，數據修改需要可審計且合理。包括業務用戶、審計師和監管機構在內的利益相關者可能需要解釋為什麼進行某些數據修正，以及人工智能模型如何得出這些決策。這在數據準確性和完整性具有重要影響的領域中尤為重要，例如金融、醫療保健和法律事務。

為了滿足可解釋性和透明度的需求，自我修復數據技術應該包含能夠提供人工智能模型決策過程洞察的機制。這可以通過以下各種方法實現：

1. **思維鏈：**在對數據進行更改之前要求模型“大聲”解釋其思維過程，這可能有助於更容易理解決策過程，並能為所做的修正生成人類可讀的解釋。權衡之處在於在將解釋與結構化數據輸出分離時會增加一些複雜性，這可以通過...來解決
2. **解釋生成：**自我修復數據工作器可以具備為其所做修正生成人類可讀解釋的能力。這可以通過要求模型將其決策過程作為易於理解的解釋_整合到數據本身_中來實現。例如，自我修復數據工作器可以生成一份報告，突出顯示它識別出的具體數據不一致性、應用的修正以及這些修正背後的理由。
3. **特徵重要性：**人工智能模型可以在其指令中被告知數據修正過程中不同特徵或屬性的重要性。這些指令反過來可以向人類利益相關者展示。通過識別影響模型決策的關鍵因素，利益相關者可以深入了解修正背後的推理並評估其有效性。
4. **日誌記錄和審計：**實施全面的日誌記錄和審計機制對於維護自我修復數據過程的透明度至關重要。人工智能模型所做的每一個數據修正都應該被記錄下來，包括原始數據、修正後的數據以及採取的具體行動。這種審計追蹤允許進行回顧性分析，並為數據修改提供清晰的記錄。

5. **人機協作方法：**納入人機協作方法可以增強自我修復數據技術的可解釋性和透明度。通過讓人類專家參與審查和驗證人工智能生成的修正，組織可以確保修正與領域知識和業務需求保持一致。人工監督增加了一層問責制，並允許識別人工智能模型中的任何潛在偏見或錯誤。
6. **持續監控和評估：**定期監控和評估自我修復數據技術的性能對於維護透明度和信任至關重要。通過隨時間評估人工智能模型的準確性和有效性，組織可以識別任何偏差或異常並採取糾正措施。持續監控有助於確保自我修復數據過程保持可靠並與預期結果保持一致。

可解釋性和透明度是實施自我修復數據技術時的關鍵考慮因素。通過為數據修正提供清晰的解釋、維護全面的審計追蹤以及涉及人工監督，組織可以建立對自我修復數據過程的信任，並確保對數據的修改是合理的，並與業務目標保持一致。

在自動化的好處和透明度需求之間取得平衡很重要。雖然自我修復數據技術可以顯著提高數據質量和效率，但不應以失去對數據修正過程的可見性和控制為代價。通過在設計自我修復數據工作器時考慮到可解釋性和透明度，組織可以在保持必要的問責制和數據信任度的同時利用人工智能的力量。

意外後果

雖然自我修復數據技術旨在提高數據質量和一致性，但意識到潛在的意外後果至關重要。如果沒有經過仔細設計和監控，自動修正可能會無意中改變數據的含義或上下文，導致下游問題。

自我修復數據的主要風險之一是在數據修正過程中引入偏差或錯誤。人工智能模型，像任何其他軟件系統一樣，可能受到訓練數據中存在的偏見或通過算法設計引入的偏見的影響。如果這些偏差沒有被識別和緩解，它們可能會通過自我修復數據過程傳播，並導致偏斜或錯誤的數據修改。

例如，考慮一個負責糾正客戶人口統計數據中不一致性的自我修復數據工作器。如果 AI 模型從歷史數據中學習到了偏見，比如將某些職業或收入水平與

特定性別或族裔相關聯，它可能會做出錯誤的假設，並以強化這些偏見的方式修改數據。這可能導致不準確的客戶檔案、錯誤的商業決策，以及潛在的歧視性結果。

另一個潛在的意外後果是在數據糾正過程中損失有價值的信息或上下文。自我修復數據技術通常專注於標準化和規範化數據以確保一致性。然而，在某些情況下，原始數據可能包含對於理解全局至關重要的細微差別、例外情況或上下文信息。盲目執行標準化的自動糾正可能會無意中移除或模糊這些有價值的信息。

例如，想像一個負責糾正醫療記錄中不一致性的自我修復數據工作器。如果工作器遇到一個患者的罕見病症或不尋常的治療方案的病歷，它可能會嘗試將數據規範化以符合更常見的模式。然而，在這樣做的過程中，它可能會丟失準確表示患者獨特情況所必需的具體細節和上下文。這種信息的損失可能對患者護理和醫療決策產生嚴重影響。

為了降低意外後果的風險，在設計和實施自我修復數據技術時採取主動方法至關重要：

1. **** 徹底的測試和驗證：**** 在將自我修復數據工作器部署到生產環境之前，必須針對各種不同場景徹底測試和驗證其行為。這包括使用涵蓋各種邊緣情況、例外情況和潛在偏見的代表性數據集進行測試。嚴格的測試有助於在影響實際數據之前識別和解決任何意外後果。
2. **** 持續監控和評估：**** 實施持續監控和評估機制對於及時發現和緩解意外後果至關重要。定期審查自我修復數據流程的結果、分析對下游系統和決策的影響，以及收集利益相關者的反饋，可以幫助識別任何不利影響並及時採取糾正措施。如果你的組織有操作儀表板，添加與自動數據更改相關的清晰可見的指標可能是個好主意。添加與正常數據更改活動的大幅偏差相關的警報可能是更好的主意！
3. **** 人工監督和干預：**** 保持人工監督和干預自我修復數據流程的能力至關重要。雖然自動化可以大大提高效率，但讓人類專家審查和驗證 AI 模型

所做的糾正很重要，特別是在關鍵或敏感領域。人類的判斷和領域專業知識可以幫助識別和解決可能出現的任何意外後果。

4. ** 可解釋人工智能（XAI）和透明度：** 如前一小節所討論的，納入可解釋人工智能技術並確保自我修復數據流程的透明度可以幫助緩解意外後果。通過為數據糾正提供清晰的解釋並維護全面的審計追蹤，組織可以更好地理解和追溯 AI 模型所做修改背後的推理。
5. ** 漸進和迭代方法：** 採用漸進和迭代的自我修復數據方法可以幫助最小化意外後果的風險。與其一次性對整個數據集應用自動糾正，不如從數據子集開始，隨著技術證明有效和可靠而逐步擴大範圍。這允許在過程中進行仔細監控和調整，減少任何意外後果的影響。
6. ** 協作和反饋：** 在整個自我修復數據流程中讓不同領域的利益相關者參與並鼓勵協作和反饋，可以幫助識別和解決意外後果。定期尋求領域專家、數據消費者和最終用戶的意見，可以提供關於數據糾正實際影響的寶貴見解，並突顯可能被忽視的任何問題。

通過主動應對意外後果的風險並實施適當的保護措施，組織可以利用自我修復數據技術的優勢，同時最小化潛在的不利影響。重要的是要將自我修復數據視為一個迭代和協作的過程，持續監控、評估和改進技術，以確保它們與預期結果保持一致，並維護數據的完整性和可靠性。

在考慮使用自我修復數據模式時，必須仔細評估這些因素，並權衡收益與潛在風險和限制。在某些情況下，將自動糾正與人工監督和干預相結合的混合方法可能是最合適的解決方案。

還值得注意的是，自我修復數據技術不應被視為替代健全的數據驗證、輸入淨化和錯誤處理機制。這些基礎實踐對於確保數據完整性和安全性仍然至關重要。自我修復數據應被視為一種補充方法，可以增強和改進這些現有措施。

最終，使用自我修復數據模式的決定取決於您的應用程序的具體要求、約束和優先事項。通過仔細考慮上述考慮因素，並將它們與您的應用程序目標和架構

保持一致，您可以就何時以及如何有效利用自我修復數據技術做出明智的決定。

情境內容生成



情境內容生成模式運用大型語言模型（LLM）的力量，在應用程式中生成動態且具有情境特性的內容。這類模式認識到根據使用者的特定需求、偏好，甚至是與應用程式之前和當前的互動來提供個人化且相關內容的重要性。

在這種方法中，「內容」既指主要內容（即部落格文章、文章等），也指元內容，例如對主要內容的推薦。

情境內容生成模式在提升使用者參與度、提供客製化體驗，以及為您和您的使用者自動化內容創建任務方面可以發揮關鍵作用。透過運用本章所描述的模式，您可以創建能夠動態生成內容的應用程式，即時適應情境和輸入。

這些模式通過將 LLM 整合到應用程式的輸出中來運作，範圍涵蓋從使用者介面

(有時稱為「chrome」) 到電子郵件和其他形式的通知，以及任何內容生成管道。

當使用者與應用程式互動或觸發特定內容請求時，應用程式會捕捉相關情境，例如使用者偏好、先前互動或特定提示。這些情境資訊連同任何必要的範本或指南一起輸入到 LLM 中，用於生成原本必須硬編碼、儲存在資料庫中或透過演算法生成的文字輸出。

LLM 生成的內容可以採取各種形式，例如個人化推薦、動態產品描述、客製化電子郵件回覆，甚至是完整的文章或部落格文章。我在一年多前開創的最具突破性的應用之一是動態生成 UI 元素，如表單標籤、工具提示和其他類型的解釋文字。

個人化

情境內容生成模式的主要優勢之一是能夠為使用者提供高度個人化的體驗。透過基於使用者特定情境生成內容，這些模式使應用程式能夠根據個別使用者的興趣、偏好和互動來客製化內容。

個人化不僅僅是將使用者的名字插入通用內容中。它涉及利用每個使用者可用的豐富情境來生成能夠引起他們特定需求和慾望共鳴的內容。這種情境可以包括多種因素，例如：

1. **使用者檔案資訊：**在應用這種技術的最一般層面上，人口統計資料、興趣、偏好和其他檔案屬性可用於生成與使用者背景和特徵相符的內容。
2. **行為數據：**使用者過去與應用程式的互動，如瀏覽過的頁面、點擊過的連結或購買過的產品，可以提供有關其行為和興趣的寶貴見解。這些數據可用於生成反映其參與模式並預測其未來需求的內容建議。
3. **情境因素：**使用者的當前情境，如位置、裝置、一天中的時間，甚至天氣，都可能影響內容生成過程。例如，旅遊應用程式可能擁有一個 AI 工作器，能夠根據使用者的當前位置和當時的天氣狀況生成個人化推薦。

透過利用這些情境因素，情境內容生成模式使應用程式能夠提供看似為每個使用者量身打造的内容。這種程度的個人化具有幾個重要優勢：

1. **提高參與度：**個人化內容能夠吸引使用者的注意力，並使他們持續與應用程式互動。當使用者感覺內容與其需求相關且直接針對其需求時，他們更可能花更多時間與應用程式互動並探索其功能。
2. **提升使用者滿意度：**個人化內容展示了應用程式理解並關心使用者的獨特需求。透過提供有幫助、資訊豐富且符合其興趣的內容，應用程式可以提升使用者滿意度並與使用者建立更強的連結。
3. **更高的轉換率：**在電子商務或行銷應用程式的情境中，個人化內容可以顯著影響轉換率。透過向使用者展示根據其偏好和行為量身打造的产品、優惠或推薦，應用程式可以增加使用者採取期望行動的可能性，例如購買或註冊服務。

生產力

情境內容生成模式可以透過減少創意過程中手動內容生成和編輯的需求來顯著提升某些類型的生產力。透過利用 LLM 的力量，您可以大規模生成高品質內容，節省內容創作者和開發人員原本必須花在繁瑣手動工作上的時間和精力。

傳統上，內容創作者需要研究、撰寫、編輯和格式化內容，以確保其符合應用程式的要求和使用者的期望。這個過程可能相當耗時且需要大量資源，尤其是當內容量不斷增長時。

然而，透過情境內容生成模式，內容創作過程可以大幅自動化。大型語言模型可以根據提供的提示和指導方針，生成連貫、語法正確且具有情境相關性的內容。這種自動化帶來了幾項生產力優勢：

1. **減少人工工作：**透過將內容生成任務委託給大型語言模型，內容創作者可以專注於更高層次的任務，如內容策略、構思和品質保證。他們可以為大型語言模型提供必要的情境、範本和指導方針，讓它處理實際的內容生成。這減少了寫作和編輯所需的人工工作，使內容創作者能更有效率地工作。

2. **更快的內容創作：**大型語言模型能比人類寫作者更快地生成內容。只要有適當的提示和指導方針，大型語言模型就能在幾秒或幾分鐘內產生多個內容。這種速度使應用程式能以更快的步調生成內容，跟上使用者需求和不斷變化的數位環境。

更快的內容創作是否正在導致「公地悲劇」的情況，讓網際網路充斥著無人閱讀的內容？可惜的是，我認為答案是肯定的。

3. **一致性和品質：**大型語言模型可以輕鬆修改內容，使其在風格、語氣和品質上保持一致。在清晰的指導方針和範例下，某些類型的應用（如新聞編輯室、公關等）可以確保其人工生成的內容符合品牌聲音並達到所需的品質標準。這種一致性減少了大量編輯和修訂的需求，節省了內容創作過程中的時間和精力。
4. **迭代和優化：**情境內容生成模式使內容能夠快速迭代和優化。透過調整提供給大型語言模型的提示、範本或指導方針，您的應用程式可以快速生成內容的變體，並以過去無法實現的自動化方式測試不同的方法。這種迭代過程允許更快速地實驗和改進內容策略，隨著時間推移產生更有效和更具吸引力的內容。這種特定技術對於依賴跳出率和參與度生存的電子商務應用程式來說可能是徹底的遊戲改



重要的是要注意，雖然情境內容生成模式可以大大提高生產力，但並不能完全消除人類參與的需求。內容創作者和編輯在定義整體內容策略、為大型語言模型提供指導，以及確保生成內容的品質和適當性方面仍然發揮著關鍵作用。

透過自動化更多重複性和耗時的內容創作方面，情境內容生成模式釋放了寶貴的人力時間和資源，這些可以重新導向更高價值的任務。這種提高的生產力使您能夠為使用者提供更個人化和吸引人的內容，同時優化內容創作工作流程。

快速迭代和實驗

情境內容生成模式使您能夠快速迭代和實驗不同的內容變體，從而更快地優化和改進您的內容策略。只需調整提供給模型的情境、範本或指導方針，您就可以在幾秒鐘內生成多個版本的內容。

這種快速迭代能力提供了幾個關鍵優勢：

1. **測試和優化：**有了快速生成內容變體的能力，您可以輕鬆測試不同的方法並衡量其效果。例如，您可以生成產品描述或行銷訊息的多個版本，每個版本都針對特定的使用者群體或情境量身定制。透過分析使用者參與度指標，如點擊率或轉換率，您可以識別最有效的內容變體並相應地優化您的內容策略。
2. **A/B 測試：**情境內容生成模式實現了內容的無縫 A/B 測試。您可以生成兩個或更多的內容變體，並隨機向不同的使用者群體展示。透過比較每個變體的表現，您可以確定哪些內容最能引起目標受眾的共鳴。這種數據驅動的方法使您能夠做出明智的決策，並持續改進您的內容以最大化使用者參與度並實現預期的成果。
3. **個人化實驗：**快速迭代和實驗在個人化方面特別有價值。使用情境內容生成模式，您可以根據不同的使用者群體、偏好或行為快速生成個人化的內容變體。透過實驗不同的個人化策略，您可以識別最有效的方法來吸引個別使用者並提供量身定制的體驗。
4. **** 適應不斷變化的趨勢：**** 快速迭代和實驗的能力使您能夠保持靈活並適應不斷變化的趨勢和使用者偏好。隨著新主題、關鍵字或使用者行為的出現，您可以快速生成與這些趨勢相符的內容。通過持續實驗和改進您的內容，您可以在不斷演進的數位環境中保持相關性並維持競爭優勢。
5. **** 成本效益實驗：**** 傳統的內容實驗通常需要大量的時間和資源，因為內容創作者需要手動開發和測試不同的變體。然而，使用情境式內容生成模式，實驗的成本大大降低。大型語言模型可以快速且大規模地生成內容變體，讓您能夠在不產生巨大成本的情況下探索各種想法和方法。

為了充分利用快速迭代和實驗，重要的是要建立一個明確定義的實驗框架。這個框架應該包括：

- 每個實驗的明確目標和假設
- 適當的指標和追蹤機制來衡量內容表現
- 分類和目標策略，確保相關的內容變體能夠傳遞給正確的使用者
- 從實驗數據中獲取見解的分析和報告工具
- 將學習成果和優化整合到您的內容策略中的流程

通過採用快速迭代和實驗，您可以持續改進和優化您的內容，確保它保持吸引力、相關性，並有效實現應用程式的目標。這種敏捷的內容創作方法使您能夠保持領先地位並提供卓越的使用者體驗。

可擴展性和效率

隨著應用程式的成長和個人化內容需求的增加，情境式內容生成模式能夠實現內容創作的高效擴展。大型語言模型可以同時為大量使用者和情境生成內容，而無需相應增加人力資源。這種可擴展性使應用程式能夠在不影響其內容創作能力的情況下，為不斷增長的使用者群提供個人化體驗。



請注意，情境式內容生成可以有效地用於「即時」國際化您的應用程式。事實上，這正是我使用 `Instant18n` Gem 來讓 `Olympia` 支援超過半打語言的方式，儘管我們成立還不到一年。

AI 驅動的本地化

如果允許我自誇一下，我認為我的 Rails 應用程式 `Instant18n` 函式庫是「情境式內容生成」模式的一個開創性範例，展示了 AI 在應用程式開發中的變革潛力。

這個 gem 運用 OpenAI 的 GPT 大型語言模型的力量，徹底改革了 Rails 應用程式中處理國際化和本地化的方式。

傳統上，國際化 Rails 應用程式需要手動定義翻譯鍵值並為每種支援的語言提供相應的翻譯。這個過程可能耗時、資源密集且容易出現不一致。然而，使用 Instant18n gem，本地化的範式被完全重新定義。

通過整合大型語言模型，Instant18n gem 使您能夠根據文本的上下文和含義即時生成翻譯。不需要依賴預定義的翻譯鍵值和靜態翻譯，該 gem 使用 AI 的力量動態翻譯文本。這種方法提供了幾個關鍵優勢：

1. **** 無縫本地化：**** 使用 Instant18n gem，開發人員不再需要為每種支援的語言手動定義和維護翻譯文件。該 gem 根據提供的文本和目標語言自動生成翻譯，使本地化過程變得輕鬆和無縫。
2. **** 上下文準確性：**** AI 可以獲得足夠的上下文來理解被翻譯文本的細微差別。它可以考慮周圍的上下文、慣用語和文化參考，生成準確、自然且符合上下文的翻譯。
3. **** 廣泛的語言支援：**** Instant18n gem 利用 GPT 的廣泛知識和語言能力，能夠翻譯成範圍廣泛的語言。從常見的西班牙語和法語到更罕見或虛構的語言如克林貢語和精靈語，該 gem 都能處理各種翻譯需求。
4. **** 靈活性和創造力：**** 該 gem 超越了傳統的語言翻譯，允許創意和非常規的本地化選項。開發人員可以將文本翻譯成各種風格、方言，甚至虛構語言，為獨特的使用者體驗和引人入勝的內容開闢新的可能性。
5. **** 效能優化：**** Instant18n gem 整合了快取機制以提高效能並減少重複翻譯的開銷。翻譯後的文本會被快取，使後續對相同翻譯的請求能夠快速提供服務，無需重複的 API 呼叫。

Instant18n gem 通過利用 AI 動態生成本地化內容，展示了「情境式內容生成」模式的力量。它展示了 AI 如何能夠整合到 Rails 應用程式的核心功能中，徹底改變開發人員處理國際化和本地化的方式。

透過消除手動管理翻譯的需求，並實現基於情境的即時翻譯，Instant18n 套件為開發人員節省了大量的時間和精力。這使他們能夠專注於建構應用程式的核心功能，同時確保在地化方面能夠順暢且準確地處理。

使用者測試與回饋的重要性

最後，請始終謹記使用者測試與回饋的重要性。驗證情境式內容生成是否符合使用者期望並與應用程式目標一致是至關重要的。要根據使用者見解和分析數據持續改進和優化所生成的內容。如果您正在大規模生成動態內容，而這些內容不可能由您和您的團隊手動驗證，請考慮添加回饋機制，讓使用者能夠回報奇怪或錯誤的內容，並說明原因。這些寶貴的回饋甚至可以提供給負責調整內容生成組件的 AI 工作程序！

生成式使用者介面



在現今這個時代，注意力如此珍貴，要有效地吸引使用者，不僅需要順暢且直覺的軟體體驗，還必須高度配合個人需求、偏好和情境。因此，設計師和開發人員越來越面臨著一個挑戰：如何創造能夠大規模適應並滿足每個使用者獨特需求的使用者介面。

生成式使用者介面（GenUI）是一種革命性的使用者介面設計方法，它運用大型語言模型（LLMs）的力量，即時創造高度個人化和動態的使用者體驗。我想要在本書中至少為您介紹 GenUI 的基礎知識，因為我相信這是目前應用程式設計和框架領域中最具發展潛力的機會之一。我確信在這個特定領域中將會出現數十個或更多成功的商業和開源專案。

從本質上來說，GenUI 將**情境內容生成**的原則與進階人工智慧技術相結合，根據對使用者情境、偏好和目標的深入理解，動態生成使用者介面元素，如文字、圖像和版面配置。GenUI 使設計師和開發人員能夠創造出能根據使用者互動而調適和演進的介面，提供前所未有的個人化程度。

GenUI 代表了我們在使用者介面設計方法上的根本性改變。我們不再為大眾設計，而是為個人設計。個人化的內容和介面有潛力創造出能與每個使用者產生更深層共鳴的使用者體驗，從而提高參與度、滿意度和忠誠度。

作為一項前沿技術，轉向 GenUI 充滿了概念和實務上的挑戰。將人工智慧整合到設計過程中，確保生成的介面不僅個人化，還要易於使用、無障礙，並與整體品牌和使用者的體驗保持一致，這些挑戰使得 GenUI 成為少數人而非多數人的追求。此外，人工智慧的參與也引發了關於資料隱私、透明度，甚至倫理影響的問題。

儘管存在這些挑戰，大規模的個人化體驗有能力徹底改變我們與數位產品和服務互動的方式。它為創造包容性和無障礙的介面開創了可能性，能夠滿足使用者的多樣化需求，無論其能力、背景或偏好如何。

在本章中，我們將探討 GenUI 的概念，檢視其一些定義特徵、主要優點和潛在挑戰。我們從 GenUI 最基本和最容易實現的形式開始：為傳統設計和實作的使用者介面生成文字內容。

為使用者介面生成文案

存在於應用程式介面中的文字元素，如表單標籤、工具提示和說明文字，通常都是硬編碼在模板或 UI 元件中，為所有使用者提供一致但通用的體驗。使用情境內容生成模式，您可以將這些靜態元素轉換為動態的、具有情境感知能力的個人化元件。

個人化表單

表單是網頁和行動應用程式中無所不在的部分，是收集使用者輸入的主要方式。然而，傳統表單往往呈現出一種通用且不夠個人化的體驗，其標準標籤和欄位可能並不總是符合使用者的特定情境或需求。使用者更有可能完成那些感覺針對其需求和偏好量身打造的表單，這樣可以提高轉換率和使用者的滿意度。

然而，在個人化和一致性之間取得平衡很重要。雖然讓表單適應個別使用者可能有益，但維持一定程度的熟悉感和可預測性至關重要。即使有個人化元素，使用者仍應能輕易識別和導覽表單。

以下是一些個人化表單的靈感：

情境欄位建議

GenUI可以分析使用者的過往互動、偏好和資料，以預測方式提供智慧欄位建議。例如，如果使用者之前輸入過送貨地址，表單可以自動使用其儲存的資訊填充相關欄位。這不僅節省時間，還展示了應用程式理解並記住使用者偏好的能力。

等一下，這種技術不是不用 AI 也能做到嗎？當然可以，但是用 AI 來驅動這類功能的美妙之處有兩個：1) 實作起來可以非常簡單，2) 隨著使用者介面的改變和演進，它能保持高度的靈活性。

讓我們為我們假設的訂單處理系統建立一個服務，它會主動地試著為使用者填入正確的送貨地址。

```
1  class OrderShippingAddressSubscriber
2      include Raix::ChatCompletion
3
4      attr_accessor :order
5
6      delegate :customer, to: :order
7
8      DIRECTIVE = "You are a smart order processing assistant. Given the
9      customer's order history, guess the most likely shipping address
10     for the current order."
11
12     def order_created(order)
```

```
13     return unless order.pending? && order.shipping_address.blank?
14
15     self.order = order
16
17     transcript.clear
18     transcript << { system: DIRECTIVE }
19     transcript << { user: "Order History: #{order_history.to_json}" }
20     transcript << { user: "Current Order: #{order.to_json}" }
21
22     response = chat_completion
23     apply_predicted_shipping_address(order, response)
24 end
25
26 private
27
28 def apply_predicted_shipping_address(order, response)
29   # extract the shipping address from the response...
30   # ...and assume there's some sort of live update of the address fields
31   order.update(shipping_address:)
32 end
33
34 def order_history
35   customer.orders.successful.limit(100).map do |order|
36     {
37       date: order.date,
38       description: order.description,
39       shipping_address: order.shipping_address
40     }
41   end
42 end
```

42 **end**

43 **end**

這個例子雖然非常簡化，但適用於大多數情況。其概念是要讓人工智慧像人類一樣進行推測。為了說明清楚我所談論的內容，讓我們來看看一些範例資料：

```
1  Order History:
2  [
3    {"date": "2024-01-03", "description": "garden soil mix",
4      "shipping_address": "123 Country Lane, Rural Town"},
5    {"date": "2024-01-15", "description": "hardcover fiction novels",
6      "shipping_address": "456 City Apt, Metroville"},
7    {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
8      "789 Suburb St, Quietville"},
9    {"date": "2024-02-01", "description": "organic vegetables",
10     "shipping_address": "123 Country Lane, Rural Town"},
11   {"date": "2024-02-17", "description": "mystery thriller book set",
12     "shipping_address": "456 City Apt, Metroville"},
13   {"date": "2024-02-25", "description": "baby wipes",
14     "shipping_address": "789 Suburb St, Quietville"},
15   {"date": "2024-03-05", "description": "flower seeds",
16     "shipping_address": "123 Country Lane, Rural Town"},
17   {"date": "2024-03-20", "description": "biographies",
18     "shipping_address": "456 City Apt, Metroville"},
19   {"date": "2024-03-30", "description": "baby formula",
20     "shipping_address": "789 Suburb St, Quietville"},
21   {"date": "2024-04-12", "description": "lawn fertilizer",
22     "shipping_address": "123 Country Lane, Rural Town"},
23   {"date": "2024-04-22", "description": "science fiction novels",
24     "shipping_address": "456 City Apt, Metroville"},
```

```
25  {"date": "2024-05-02", "description": "infant toys",  
26    "shipping_address": "789 Suburb St, Quietville"},  
27  {"date": "2024-05-14", "description": "outdoor grill",  
28    "shipping_address": "123 Country Lane, Rural Town"},  
29  {"date": "2024-05-29", "description": "literary classics",  
30    "shipping_address": "456 City Apt, Metroville"},  
31  {"date": "2024-06-11", "description": "baby clothes",  
32    "shipping_address": "789 Suburb St, Quietville"},  
33  {"date": "2024-07-01", "description": "watering can",  
34    "shipping_address": "123 Country Lane, Rural Town"},  
35  {"date": "2024-07-18", "description": "non-fiction essays",  
36    "shipping_address": "456 City Apt, Metroville"},  
37  {"date": "2024-07-28", "description": "baby bath items",  
38    "shipping_address": "789 Suburb St, Quietville"},  
39  {"date": "2024-08-09", "description": "herb garden kit",  
40    "shipping_address": "123 Country Lane, Rural Town"},  
41  {"date": "2024-08-24", "description": "children's books",  
42    "shipping_address": "456 City Apt, Metroville"}  
43 ]
```

你注意到資料中的模式了嗎？我可以保證，對大型語言模型來說，這些都是簡單的任務。讓我們來問問 GPT-4，「溫度計」最可能的送貨地址是什麼。

1 From the order history you've provided, it looks like the purchases are
2 generally clustered into three main types based on the shipping addresses:
3
4 123 Country Lane, Rural Town - This address often orders garden and
5 outdoor-related items like soil mix, vegetables, flower seeds, lawn
6 fertilizer, an outdoor grill, a watering can, and an herb garden kit.
7
8 456 City Apt, Metroville - This address tends to order various types of
9 books, indicating an interest in reading across genres like fiction,
10 thrillers, biographies, science fiction, literary classics, non-fiction
11 essays, and children's books.
12
13 789 Suburb St, Quietville - The purchases for this address are consistently
14 baby-related products, such as diapers, wipes, baby formula, infant toys,
15 baby clothes, and baby bath items.
16
17 Given these patterns, a digital thermometer could potentially be a fit for
18 any of the addresses based on different needs: outdoor temperature tracking,
19 a tool for a passionate reader interested in science gadgets, or a health
20 tool for a baby. However, considering its common use in monitoring health,
21 especially for babies, it's most likely to be purchased by someone who's
22 already buying baby care products.
23
24 Therefore, the most likely shipping address for a digital thermometer, based
25 on the purchasing patterns, would be 789 Suburb St, Quietville.

如果你認為使用像 GPT-4 這樣昂貴的模型來執行這項任務有點大材小用，你說得對！我在 Mistral 7B Instruct 上使用相同的提示，它以每秒 75 個權重的速度產生了回應，而成本僅為 0.000218 美元。

1 Based on the given order history, it appears that the customer has received
2 orders for baby-related items (diapers, wipes, formula, clothes, bath items)
3 at the address "789 Suburb St, Quietville" and garden-related items (soil,
4 seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
5 Town". The customer has also received books (fiction, non-fiction, literary
6 classics, mystery-thriller, science fiction, biographies, children's books)
7 at the address "456 City Apt, Metroville".
8
9 Given that the new order is for a "thermometer", which doesn't seem to fit
10 into any of the categories that the customer has previously ordered, it's
11 hard to accurately predict the shipping address based solely on the order
12 history. However, given the 50%-50% split between baby-related and
13 garden-related items, it could somewhat lean towards the Baby-related items
14 address ("789 Suburb St, Quietville"). But remember, this is an assumption
15 and cannot be definitively confirmed without more context or information.

對於許多線上零售商來說，這項技術的開銷和成本是否值得用來打造更神奇的結帳體驗？答案是絕對值得。從目前情況來看，AI 運算的成本只會持續下降，特別是在那些競相降價的商用開源模型託管服務供應商之間。



使用提示詞模板和結構化輸入輸出，再搭配回應圍欄來優化這類聊天完成功能。

自適應欄位排序

表單欄位的呈現順序會顯著影響使用者體驗和完成率。透過 GenUI，你可以根據使用者的情境和各個欄位的重要性來動態調整欄位順序。例如，如果使用者正在填寫健身應用程式的註冊表單，系統可以優先顯示與其健身目標和偏好相關的欄位，使整個過程更加貼切且引人入勝。

個人化微文案

表單相關的說明文字、錯誤訊息和其他微文案也可以使用 GenUI 來個人化。與其顯示像「電子郵件地址無效」這樣的通用錯誤訊息，你可以產生更有幫助且更具情境性的訊息，例如「請輸入有效的電子郵件地址以接收訂單確認信」。這些個人化的細節可以使表單體驗更加友善，減少使用者的挫折感。

個人化驗證

循著個人化微文案的思路，你可以使用 AI 以看似神奇的方式來驗證表單。想像讓 AI 在語意層面驗證使用者資料表單，尋找潛在的錯誤。

Create your account

Full name

Obie Fernandez

Email

obiefernandez@gmail.com

Did you mean obiefernandez@gmail.com? [Yes, update.](#)

Country ⓘ

 United States

Password

.....

✓ Nice work. This is an excellent password.

圖 9. 你能發現這裡的語意驗證嗎？

漸進式揭露

GenUI 可以根據使用者的情境智慧地判斷哪些表單欄位是必要的，並逐步顯示額外的欄位。這種漸進式揭露技術有助於減輕認知負荷，使填寫表單的過程更容易管理。例如，當使用者註冊基本訂閱時，表單最初只顯示必要欄位，隨著使用者的進展或選擇特定選項，可以動態引入其他相關欄位。

情境感知說明文字

工具提示通常用於當使用者懸停或互動特定元素時提供額外資訊或指引。透過「情境內容生成」方法，你可以產生根據使用者情境調整的工具提示，提供相關資訊。例如，當使用者在探索複雜功能時，工具提示可以根據他們先前的互動或技能水平提供個人化提示或範例。

說明文字，如指示、描述或幫助訊息，可以根據使用者的情境動態生成。你可以使用 LLM 產生針對使用者特定需求或問題量身打造的文字，而不是呈現通用的說明。例如，如果使用者在某個步驟遇到困難，說明文字可以提供個人化的指導或故障排除提示。

微文案指的是引導使用者使用應用程式的小段文字，如按鈕標籤、錯誤訊息或確認提示。透過將**情境內容生成**方法應用於微文案，你可以創建回應使用者行為並提供相關且有幫助文字的自適應 UI。例如，當使用者即將執行重要操作時，確認提示可以動態生成，提供清晰且個人化的訊息。

個人化的說明文字和工具提示可以大幅改善新使用者的入門體驗。透過提供具有情境特性的指導和範例，你可以幫助使用者快速理解和導覽應用程式，減少學習曲線並提高採用率。

動態且具有情境感知的介面元素也能使應用程式感覺更直覺且引人入勝。當附帶的文字是針對使用者特定需求和興趣量身打造時，使用者更可能與功能互動並進行探索。

到目前為止，我們已經討論了用 AI 來增強現有使用者介面範式的想法，但是否可以用更激進的方式重新思考使用者介面的設計和實作方式呢？

定義生成式使用者介面

與傳統的使用者介面設計不同，設計師不再創造固定且靜態的介面，生成式使用者介面暗示了一個軟體具有靈活、個人化且能即時演化和適應的體驗的未來。每當我們使用 AI 驅動的對話式介面時，我們都在讓 AI 適應使用者的特定需求。生成式使用者介面更進一步，將這種適應性應用到軟體的視覺介面上。

今天之所以能夠嘗試生成式使用者介面的想法，是因為大型語言模型 已經理解程式設計，且其基礎知識包含了使用者介面技術和框架。現在的問題是，是否可以使用大型語言模型來生成為每個使用者量身打造的使用者介面元素，如文字、圖像、版面配置，甚至整個介面。模型可以被指示考慮各種因素，如使用者過去的互動、明確的偏好、人口統計資訊，以及當前使用的情境，以創建高度個人化且相關的介面。

生成式使用者介面在幾個關鍵方面與傳統使用者介面設計不同：

1. **動態且適應性強：**傳統的使用者介面設計涉及創建對所有使用者都相同的固定、靜態介面。相比之下，生成式使用者介面能根據使用者需求和情境動態適應和改變。這意味著同一個應用程式可以為不同使用者呈現不同的介面，甚至可以為同一使用者在不同情況下呈現不同的介面。
2. **規模化個人化：**在傳統設計中，由於時間和資源的限制，為每個使用者創建個人化體驗通常是不切實際的。而生成式使用者介面則允許規模化的個人化。透過運用 AI，設計師可以創建自動適應每個使用者獨特需求和偏好的介面，而無需手動為每個使用者群體設計和開發單獨的介面。
3. **注重成果：**傳統的使用者介面設計通常著重於創建視覺吸引力和功能性的介面。雖然這些方面在生成式使用者介面中仍然重要，但主要焦點轉向實現期望的使用者成果。生成式使用者介面旨在創建針對每個使用者特定目標和任務優化的介面，將可用性和效能置於純粹美學考慮之上。
4. **持續學習和改進：**生成式使用者介面系統可以根據使用者互動和回饋持續學習和改進。當使用者與生成的介面互動時，AI 模型可以收集使用者行為、偏好和成果的資料，使用這些資訊來改進和優化未來的介面生成。這

種反覆學習過程使生成式使用者介面系統能夠隨時間越來越有效地滿足使用者需求。

值得注意的是，生成式使用者介面與 AI 輔助設計工具不同，後者提供建議或自動化某些設計任務。雖然這些工具有助於簡化設計過程，但它們仍然依賴設計師做出最終決定並創建靜態介面。相比之下，生成式使用者介面涉及 AI 系統在基於使用者資料和情境的實際介面生成和適應中扮演更積極的角色。

生成式使用者介面代表了我們處理使用者介面設計方式的重大轉變，從一體適用的解決方案轉向高度個人化、適應性的體驗。透過運用 AI 的力量，生成式使用者介面有潛力革新我們與數位產品和服務互動的方式，為每個使用者創建更直觀、更吸引人且更有效的介面。

範例

為了說明生成式使用者介面的概念，讓我們考慮一個假設的健身應用程式「FitAI」。這個應用程式旨在根據使用者的個人目標、健身水平和偏好提供個人化的運動計劃和營養建議。

在傳統的使用者介面設計方法中，FitAI 可能會有一組對所有使用者都相同的固定畫面和元素。然而，使用生成式使用者介面，應用程式的介面可以動態適應每個使用者的獨特需求和情境。

這種方法在 2024 年可能有點難以想像實現，而且可能甚至沒有足夠的投資回報率，但這是可能的。

以下是它可能的運作方式：

1. 新用戶引導：

- 不是標準的問卷調查，FitAI 使用對話式 AI 來收集有關使用者目標、當前健身水平和偏好的資訊。

- 基於這初步互動，AI 生成個人化的儀表板版面，突出顯示與使用者目標最相關的功能和資訊。
- 目前的 AI 技術可能有一系列可用於組合個人化儀表板的畫面元件。
- 未來的 AI 技術可能會扮演經驗豐富的使用者介面設計師的角色，從頭開始創建儀表板。

2. 運動計畫器：

- AI 會根據使用者的經驗水平和可用設備來調整運動計畫介面。
- 對於沒有設備的初學者，它可能會顯示簡單的徒手運動，並附上詳細說明和影片。
- 對於能夠使用健身房的進階使用者，它可能會顯示更複雜的運動程序，並減少說明內容。
- 運動計畫的內容並非只是從大型資料集中過濾而來，而是可以根據包含使用者所有已知資訊的上下文，即時從知識庫中生成。

3. 進度追蹤：

- 進度追蹤介面會根據使用者的目標和參與模式而演變。
- 如果使用者主要專注於減重，介面可能會突出顯示體重趨勢圖表和卡路里消耗統計。
- 對於增肌的使用者，它可能會強調力量增長和身體組成的變化。
- AI 可以根據使用者的實際進度來調整應用程式的這部分。如果進度在一段時間內停滯，應用程式可以轉換到一種模式，試圖引導使用者透露停滯的原因，以便加以改善。

4. 營養建議：

- 營養部分會根據使用者的飲食偏好和限制進行調整。
- 對於素食者，它可能會顯示植物性餐點建議和蛋白質來源。
- 對於麩質不耐症的使用者，它會自動從建議中過濾掉含麩質的食物。
- 同樣地，內容並非從適用於所有使用者的龐大餐點資料集中抽取，而是根據使用者的具體情況和限制，從知識庫中綜合生成。

- 例如，食譜會根據使用者的體能水平和身體數據的持續變化，生成符合其不斷變化的卡路里需求的食材規格。

5. 激勵元素：

- 應用程式的激勵內容和通知會根據使用者的性格類型和對不同激勵策略的反應進行個人化。
- 某些使用者可能會收到鼓勵性的訊息，而其他人則會得到更多數據導向的回饋。

在這個例子中，生成式使用者介面使 FitAI 能為每位使用者創造高度客製化的體驗，可能提高參與度、滿意度和達成健身目標的可能性。介面元素、內容，甚至應用程式的「個性」都會調整以最好地服務每個使用者的需求和偏好。

轉向結果導向設計

生成式使用者介面代表了使用者介面設計方法的根本轉變，從專注於創建特定介面元素轉向更全面的、結果導向的方法。這種轉變有幾個重要含義：

1. 專注於使用者目標：

- 設計師需要更深入地思考使用者目標和期望的結果，而不是具體的介面組件。
- 重點將放在創建能生成幫助使用者有效率且有效地達成目標的介面系統上。
- 新的使用者介面框架將會出現，為基於 AI 的設計師提供所需工具，使其能即時且從頭開始生成使用者體驗，而不是基於預定義的畫面規格。

2. 設計師角色的改變：

- 設計師將從創建固定版面配置轉變為定義 AI 系統在生成介面時需要遵循的規則、限制和指導方針。
- 他們需要在資料分析、AI 提示工程和系統思維等領域發展技能，以有效指導生成式使用者介面系統。

3. 使用者研究的重要性：

- 在生成式使用者介面環境中，使用者研究變得更加重要，因為設計師不僅需要了解使用者偏好，還需要了解這些偏好和需求在不同情境下如何變化。
- 持續的使用者測試和回饋迴圈對於改進 AI 生成有效介面的能力至關重要。

4. 為可變性設計：

- 設計師不再是創建單一「完美」介面，而是需要考慮多種可能的變化，確保系統能為不同使用者需求生成適當的介面。
- 這包括為邊緣案例設計，並確保生成的介面在不同配置下都能保持可用性和無障礙性。
- 產品差異化在使用者心理學的不同觀點上呈現新的面向，並利用競爭對手無法取得的獨特資料集和知識庫。

挑戰與考量

雖然生成式使用者介面提供了令人興奮的可能性，但它也帶來了幾個挑戰和需要考慮的問題：

1. 技術限制：

- 目前的 AI 技術雖然先進，但在理解複雜的使用者意圖和生成真正符合情境的介面方面仍有限制。

- 在效能較低的裝置上，即時生成介面元素相關的效能問題。

2. 資料需求：

- 根據使用情境，有效的生成式使用者介面系統可能需要大量使用者資料來生成個人化介面。
- 在道德範圍內取得真實使用者資料的挑戰引發了對資料隱私和安全的顧慮，以及用於訓練生成式使用者介面模型的資料可能存在偏差的問題。

3. 可用性與一致性：

- 至少在這項技術普及之前，介面持續變動的應用程式可能會導致可用性問題，因為使用者可能難以找到熟悉的元素或有效率地導覽。
- 在個人化與維持一致性、可學習的介面之間取得平衡將是關鍵。

4. 過度依賴人工智慧：

- 存在將設計決策過度委託給人工智慧系統的風險，可能導致缺乏創意、有問題或單純失效的介面選擇。
- 在可預見的未來，人為監督和能夠推翻人工智慧生成設計的能力仍將維持重要。

5. 無障礙設計考量：

- 確保動態生成的介面對身心障礙使用者保持無障礙使用，帶來全新的挑戰，考慮到一般系統在無障礙合規性方面表現不佳，這點令人擔憂。
- 另一方面，人工智慧設計師可能會內建對無障礙設計的考量，並具備即時建立無障礙介面的能力，就像為一般使用者建立使用者介面一樣。
- 無論如何，生成式使用者介面系統應該要以健全的無障礙設計準則和測試流程為基礎。

6. 使用者信任與透明度：

- 使用者可能會對「過於了解」他們或以難以理解的方式改變的介面感到不安。
- 提供介面個人化方式和原因的透明度，對建立使用者信任至關重要。

未來展望與機會

生成式使用者介面（GenUI）的未來在革新我們與數位產品和服務互動方式方面蘊含巨大潛力。隨著這項技術持續發展，我們可以預期使用者介面的設計、實作和體驗方式將發生重大轉變。我認為生成式使用者介面是最終將推動我們的軟體邁入現今被視為科幻領域的現象。

生成式使用者介面最令人興奮的前景之一是其提升無障礙設計的潛力，這超越了僅確保重度身心障礙者不被完全排除在軟體使用之外的範疇。透過自動調適介面以符合個別使用者需求，生成式使用者介面可能使數位體驗比以往更具包容性。想像介面能無縫地為年輕或視障使用者提供更大的文字，或為認知障礙者提供簡化的版面配置，這一切都無需手動配置或分離的「無障礙版本」應用程式。

生成式使用者介面的個人化功能很可能會提高各類數位產品的使用者參與度、滿意度和忠誠度。隨著介面更加適應個人偏好和行為，使用者將發現數位體驗更直覺且愉悅，可能導致與科技之間更深入且有意義的互動。

生成式使用者介面也有潛力改變新使用者的入門過程。透過建立直覺且個人化的首次使用者體驗，並快速適應每位使用者的專業程度，生成式使用者介面可能顯著降低學習新應用程式的門檻。這可能導致更快的採用率，並增加使用者探索新功能的信心。

另一個令人興奮的可能性是生成式使用者介面能在不同裝置和平台間維持一致的使用者體驗，同時針對每個特定使用情境進行最佳化。這

可能解決長期以來在日益分散的裝置生態系統中提供連貫體驗的挑戰，從智慧型手機和平板電腦到桌上型電腦以及新興技術如擴增實境眼鏡。

生成式使用者介面的資料導向特性為使用者介面設計的快速迭代和改進開啟了機會。透過收集使用者如何與生成介面互動的即時資料，設計師和開發人員可以獲得前所未有的使用者行為和偏好洞察。這個回饋迴圈可能導致使用者介面設計的持續改進，這些改進是由實際使用模式而非假設或有限的使用者測試所驅動。

為了準備這個轉變，設計師需要發展他們的技能組合和思維模式。重點將從建立固定版面配置轉向開發全面的設計系統和準則，以指導人工智慧驅動的介面生成。設計師需要培養對資料分析、人工智慧技術和系統思維的深入理解，以有效指導生成式使用者介面系統。

此外，隨著生成式使用者介面模糊了設計和技術之間的界線，設計師需要與開發人員和資料科學家更緊密合作。這種跨領域方法在創建不僅視覺吸引且使用者友善，同時也技術穩健且符合倫理的生成式使用者介面系統時將至關重要。

隨著生成式使用者介面技術的成熟，其倫理影響將會成為重要議題。設計師在發展負責任的人工智慧介面設計框架方面將扮演關鍵角色，確保個人化能增強使用者體驗的同時，不會侵犯隱私或以不道德的方式操縱使用者行為。

展望未來，生成式使用者介面既帶來令人興奮的機會，也伴隨著重大挑戰。它有潛力為全球使用者創造更直覺、更有效率且更令人滿意的數位體驗。儘管這項技術要求設計師必須適應並掌握新技能，但同時也提供了一個前所未有的機會，能以深遠且有意義的方式塑造人機互動的未來。邁向完全實現的生成式使用者介面系統的道路無疑是複雜的，但就改善使用者體驗和數位無障礙而言，這樣的未來值得我們為之努力。

智能工作流程編排



在應用程式開發領域中，工作流程在定義任務、流程和使用者的互動結構與執行方式上扮演著關鍵角色。隨著應用程式變得更加複雜，以及使用者期望不斷提升，對智能且適應性強的工作流程編排的需求變得越來越明顯。

“智能工作流程編排”方法著重於運用人工智能元件來動態編排和優化應用程式中的複雜工作流程。其目標是創建更有效率、更具響應性，且能根據即時數據和情境進行調適的應用程式。

在本章中，我們將探討支撐智能工作流程編排方法的關鍵原則和模式。我們將思考如何運用人工智能來智能化路由任務、自動化決策制定，並根據各種因素（如使用者行為、系統效能和業務規則）來動態調整工作流程。通過實際範例和真實場景，我們將展示人工智能在簡化和優化應用程式工作流程方面的變革潛力。

無論您是在建構具有複雜業務流程的企業應用程式，還是具有動態使用者旅程

的面向消費者的應用程式，本章討論的模式和技術都將為您提供知識和工具，以創建能夠提升整體使用者體驗並帶來商業價值的智能高效工作流程。

業務需求

傳統的工作流程管理方法通常依賴於預定義規則和靜態決策樹，這可能會顯得僵化、缺乏靈活性，且無法應對現代應用程式的動態特性。

考慮一個電子商務應用程式需要處理複雜訂單履行流程的場景。工作流程可能涉及多個步驟，如訂單驗證、庫存檢查、付款處理、配送和客戶通知。每個步驟可能都有其自身的規則集、依賴關係、外部整合和異常處理機制。通過手動或硬編碼邏輯來管理這樣的工作流程很快就會變得繁瑣、容易出錯且難以維護。

此外，隨著應用程式規模的擴大和併發使用者數量的增長，工作流程可能需要根據即時數據和系統效能進行自我調適和優化。例如，在高峰流量期間，應用程式可能需要動態調整工作流程以優先處理某些任務、有效分配資源並確保流暢的使用者體驗。

這就是“智能工作流程編排”方法發揮作用的地方。通過利用人工智能元件，開發人員可以創建智能、適應性強且能自我優化的工作流程。人工智能可以分析大量數據，從過往經驗中學習，並即時做出明智的決策來有效編排工作流程。

主要優勢

1. **提高效率：**人工智能可以優化任務分配、資源利用和工作流程執行，從而縮短處理時間並提高整體效率。
2. **適應性：**由人工智能驅動的工作流程可以動態適應不斷變化的條件，如使用者需求、系統效能或業務需求的波動，確保應用程式保持響應性和彈性。

3. **自動化決策：**人工智能可以自動化工作流程中的複雜決策過程，減少人工干預並最小化人為錯誤的風險。
4. **個人化：**人工智能可以分析使用者行為、偏好和情境，以個人化工作流程並為個別使用者提供量身定制的體驗。
5. **可擴展性：**人工智能驅動的工作流程可以無縫擴展以處理不斷增加的數據量 and 使用者互動，同時不影響效能或可靠性。

在接下來的章節中，我們將探討實現智能工作流程的關鍵模式和技術，並展示人工智能如何在現代應用程式中改變工作流程管理。

關鍵模式

為了在應用程式中實現智能工作流程編排，開發人員可以利用幾個關鍵模式來發揮人工智能的力量。這些模式為設計和管理工作流程提供了結構化方法，使應用程式能夠根據即時數據和情境進行調適、優化和自動化流程。讓我們探討智能工作流程編排中的一些基本模式。

動態任務路由

這種模式涉及使用人工智能根據各種因素（如任務優先級、資源可用性和系統效能）來智能地路由工作流程中的任務。人工智能算法可以分析每個任務的特徵，考慮系統的當前狀態，並做出明智的決策，將任務分配給最適合的資源或處理路徑。動態任務路由確保任務得到高效分配和執行，優化整體工作流程效能。


```
1  class TaskRouter
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      attr_accessor :task
6
7      # list of functions that can be called by the AI entirely at its
8      # discretion depending on the task received
9
10     function :analyze_task_priority do
11         TaskPriorityAnalyzer.perform(task)
12     end
13
14     function :check_resource_availability, # ...
15     function :assess_system_performance, # ...
16     function :assign_task_to_resource, # ...
17
18     DIRECTIVE = "You are a task router, responsible for intelligently
19         assigning tasks to available resources based on priority, resource
20         availability, and system performance..."
21
22     def initialize(task)
23         self.task = task
24         transcript << { system: DIRECTIVE }
25         transcript << { user: task.to_json }
26     end
27
28     def perform
29         while task.unassigned?
```

```
30         chat_completion
31
32         # todo: add max loop counter and break
33     end
34
35     # capture the transcript for later analysis
36     task.update(routing_transcript: transcript)
37 end
38 end
```

請注意第 29 行的 `while` 表達式所創建的循環，它會持續提示 AI 直到任務被分配。在第 35 行，我們保存任務的記錄以供後續分析和除錯使用（如有必要）。

情境決策制定

你可以使用非常相似的程式碼來在工作流程中進行情境感知決策。通過分析相關數據點（如用戶偏好、歷史模式和即時輸入），AI 組件可以在工作流程的每個決策點確定最適當的行動方案。根據每個用戶或場景的具體情境調整工作流程的行為，提供個性化和優化的體驗。

自適應工作流程組合

這種模式著重於基於變化的需求或條件動態組合和調整工作流程。AI 可以分析工作流程的當前狀態，識別瓶頸或低效之處，並自動修改工作流程結構以優化性能。自適應工作流程組合使應用程式能夠在無需人工干預的情況下持續發展和改進其流程。

異常處理和恢復

異常處理和恢復是智能工作流程協調的關鍵方面。在使用 AI 組件和複雜工作流程時，預期並優雅地處理異常對於確保系統的穩定性和可靠性至關重要。

以下是智能工作流程中異常處理和恢復的一些關鍵考慮因素和技術：

1. **異常傳播：**實施一個一致的方法來跨工作流程組件傳播異常。當組件內發生異常時，應當捕獲、記錄並將其傳播給協調器或負責處理異常的獨立組件。這樣做的目的是集中異常處理並防止異常被靜默吞噬，同時為[智能錯誤處理](#)開關可能性。
2. **重試機制：**重試機制有助於提高工作流程的彈性並優雅地處理間歇性故障。對於暫時性或可恢復的異常（如網路連接或資源不可用），一定要實施重試機制，這些異常可以在指定延遲後自動重試。擁有 AI 驅動的協調器或異常處理器意味著你的重試策略不必是機械性的，依賴指數回退等固定算法。你可以將重試的處理交給負責決定如何處理異常的 AI 組件的“判斷”。
3. **備用策略：**如果 AI 組件無法提供有效響應或遇到錯誤（考慮到其前沿特性，這是常見情況），要有備用機制確保工作流程能夠繼續。這可能涉及使用預設值、替代算法或[人在環路中](#)來做出決策並保持工作流程向前推進。
4. **補償動作：**協調器的指令應包括關於處理無法自動解決的異常的補償動作的說明。補償動作是為了撤銷或緩解失敗操作影響而採取的步驟。例如，如果支付處理步驟失敗，補償動作可以是回滾交易並通知用戶。補償動作有助於在面對異常時維持數據一致性和完整性。
5. **異常監控和警報：**設置監控和警報機制以檢測並通知相關利益相關者關於重要異常的情況。協調器可以意識到閾值和規則，在異常超過特定限制或發生特定類型的異常時觸發警報。這允許在問題影響整體系統之前主動識別和解決問題。

以下是 Ruby 工作流程組件中異常處理和恢復的示例：

```
1  class InventoryManager
2    def check_availability(order)
3      begin
4        # Perform inventory check logic
5        inventory = Inventory.find_by(product_id: order.product_id)
6        if inventory.available_quantity >= order.quantity
7          return true
8        else
9          raise InsufficientInventoryError,
10             "Insufficient inventory for product #{order.product_id}"
11        end
12      rescue InsufficientInventoryError => e
13        # Log the exception
14        logger.error("Inventory check failed: #{e.message}")
15
16        # Retry the operation after a delay
17        retry_count ||= 0
18        if retry_count < MAX_RETRIES
19          retry_count += 1
20          sleep(RETRY_DELAY)
21          retry
22        else
23          # Fallback to manual intervention
24          NotificationService.admin("Inventory check failed: Order #{order.id}")
25          return false
26        end
27      end
28    end
29  end
```

在這個例子中，InventoryManager 元件會檢查特定訂單的產品庫存情況。如果可用數量不足，它會拋出 InsufficientInventoryError。這個異常會被捕獲並記錄，同時實施重試機制。如果超過重試次數限制，元件會轉為人工介入，通知管理員處理。

通過實施健全的異常處理和恢復機制，您可以確保您的智能工作流程具有彈性、可維護性，並能夠優雅地處理意外情況。

這些模式構成了智能工作流程編排的基礎，可以根據不同應用程序的具體需求進行組合和調整。通過運用這些模式，開發人員可以創建具有靈活性、彈性，並針對性能和用戶體驗進行優化的工作流程。

在下一節中，我們將探討如何在實踐中實施這些模式，使用真實世界的例子和程式碼片段來說明如何將 AI 元件整合到工作流程管理中。

在實踐中實施智能工作流程編排

現在我們已經探討了智能工作流程編排中的關鍵模式，讓我們深入了解如何在實際應用程序中實施這些模式。我們將提供實用的例子和程式碼片段，以說明如何將 AI 元件整合到工作流程管理中。

智能訂單處理器

讓我們深入探討一個在 Ruby on Rails 電子商務應用程序中實施智能工作流程編排的實際例子，使用 AI 驅動的 OrderProcessor 元件。這個 OrderProcessor 實現了我們在第三章討論[多重工作者](#)時首次遇到的[流程管理器企業整合](#)概念。該元件將負責管理訂單履行工作流程，根據中間結果做出路由決策，並協調各個處理步驟的執行。

訂單履行流程包括多個步驟，如訂單驗證、庫存檢查、付款處理和配送。每個步驟都作為獨立的工作進程實施，執行特定任務並將結果返回給

OrderProcessor。這些步驟不是強制性的，甚至不一定要按照精確的順序執行。

以下是 OrderProcessor 的實施示例。它包含了來自 Raix 的兩個混入模組。第一個（ChatCompletion）賦予它聊天完成的能力，這使其成為一個 AI 元件。第二個（FunctionDispatch）啟用了 AI 的函數調用功能，允許它用函數調用而不是文本消息來響應提示。

工作者函數（validate_order、check_inventory 等）委託給各自的工作者類別，這些類別可以是 AI 或非 AI 元件，唯一的要求是它們返回的工作結果能夠以字符串形式表示。



與本書這一部分的所有其他示例一樣，這段程式碼實際上是偽代碼，僅用於傳達模式的含義並啟發您自己的創作。完整的模式描述和完整的程式碼示例包含在第二部分中。

```
1  class OrderProcessor
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."
6
7      def initialize(order)
8          self.order = order
9          transcript << { system: SYSTEM_DIRECTIVE }
10         transcript << { user: order.to_json }
11     end
12
13     def perform
14         # will continue looping until `stop_looping!` is called
15         chat_completion(loop: true)
```

```
16  end
17
18  # list of functions available to be called by the AI
19  # truncated for brevity
20
21  def functions
22    [
23      {
24        name: "validate_order",
25        description: "Invoke to check validity of order",
26        parameters: {
27          ...
28        },
29        ...
30      ]
31  end
32
33  # implementation of functions that can be called by the AI
34  # entirely at its discretion, depending on the needs of the order
35
36  def validate_order
37    OrderValidationWorker.perform(@order)
38  end
39
40  def check_inventory
41    InventoryCheckWorker.perform(@order)
42  end
43
44  def process_payment
```

```
45     PaymentProcessingWorker.perform(@order)
46   end
47
48   def schedule_shipping
49     ShippingSchedulerWorker.perform(@order)
50   end
51
52   def send_confirmation
53     OrderConfirmationWorker.perform(@order)
54   end
55
56   def finished_processing
57     @order.update!(transcript:, processed_at: Time.current)
58     stop_looping!
59   end
60 end
```

在此示例中，OrderProcessor 通過訂單對象進行初始化，並維護著工作流程執行的紀錄文本，採用大型語言模型原生的對話紀錄格式。AI 被賦予完全的控制權來協調各種處理步驟的執行，例如訂單驗證、庫存檢查、支付處理和配送。

每次調用 chat_completion 方法時，紀錄文本都會被發送給 AI 以獲取作為函數調用的完成結果。完全由 AI 來分析前一步驟的結果並確定採取適當的行動。例如，如果庫存檢查顯示庫存量低，OrderProcessor 可以安排補貨任務。如果支付處理失敗，它可以發起重試或通知客戶支援。

上面的示例中並沒有定義補貨或通知客戶支援的函數，但這絕對是可以實現的。

每次調用函數時，紀錄文本都會增長，並作為工作流程執行的記錄，包括每個步驟的結果和 AI 生成的下一步指示。這份紀錄文本可用於調試、審計，以及提供訂單履行過程的可見性。

通過在 `OrderProcessor` 中運用 AI，電子商務應用程式可以根據即時數據動態調整工作流程，並智能地處理異常情況。AI 組件可以做出明智的決策，優化工作流程，確保在複雜場景下也能順暢地處理訂單。

工作進程的唯一要求就是返回一些 AI 可以理解的輸出，用於決定下一步該做什麼，這可能會讓你開始意識到這種方法如何能夠減少在整合不同系統時通常涉及的輸入/輸出映射工作。

智能內容審核器

社交媒體應用程式通常需要至少最基本的內容審核，以確保社區的安全和健康。這個示例 `ContentModerator` 組件利用 AI 來智能地協調審核工作流程，根據內容特徵和各種審核步驟的結果做出決策。

審核過程涉及多個步驟，如文本分析、圖像識別、用戶信譽評估和人工審核。每個步驟都作為獨立的工作進程實現，執行特定任務並將結果返回給 `ContentModerator`。

以下是 `ContentModerator` 的示例實現：

```
1  class ContentModerator
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are a content moderator process manager,
6          tasked with the workflow involved in moderating user-generated content..."
7
8      def initialize(content)
9          @content = content
10         @transcript = [
11             { system: SYSTEM_DIRECTIVE },
12             { user: content.to_json }
13         ]
14     end
15
16     def perform
17         complete(@transcript)
18     end
19
20     def model
21         "openai/gpt-4"
22     end
23
24     # list of functions available to be called by the AI
25     # truncated for brevity
26
27     def functions
28         [
29             {
```

```
30     name: "analyze_text",
31     # ...
32 },
33 {
34     name: "recognize_image",
35     description: "Invoke to describe images...",
36     # ...
37 },
38 {
39     name: "assess_user_reputation",
40     # ...
41 },
42 {
43     name: "escalate_to_manual_review",
44     # ...
45 },
46 {
47     name: "approve_content",
48     # ...
49 },
50 {
51     name: "reject_content",
52     # ...
53 }
54 ]
55 end
56
57 # implementation of functions that can be called by the AI
58 # entirely at its discretion, depending on the needs of the order
```

```
59
60 def analyze_text
61     result = TextAnalysisWorker.perform(@content)
62     continue_with(result)
63 end
64
65 def recognize_image
66     result = ImageRecognitionWorker.perform(@content)
67     continue_with(result)
68 end
69
70 def assess_user_reputation
71     result = UserReputationWorker.perform(@content.user)
72     continue_with(result)
73 end
74
75 def escalate_to_manual_review
76     ManualReviewWorker.perform(@content)
77     @content.update!(status: 'pending', transcript: @transcript)
78 end
79
80 def approve_content
81     @content.update!(status: 'approved', transcript: @transcript)
82 end
83
84 def reject_content
85     @content.update!(status: 'rejected', transcript: @transcript)
86 end
87
```

```
88   private
89
90   def continue_with(result)
91     @transcript << { function: result }
92     complete(@transcript)
93   end
94 end
```

在這個例子中，ContentModerator 通過內容物件進行初始化，並以對話格式維護審核記錄。AI 組件完全控制審核工作流程，根據內容特徵和每個步驟的結果來決定執行哪些步驟。

AI 可以調用的工作函數包括 `analyze_text`、`recognize_image`、`assess_user_reputation` 和 `escalate_to_manual_review`。每個函數都會將任務委派給相應的工作處理程序（`TextAnalysisWorker`、`ImageRecognitionWorker` 等），並將結果添加到審核記錄中，其中升級函數作為終止狀態除外。最後，`approve_content` 和 `reject_content` 函數也作為終止狀態。

AI 組件分析內容並確定採取適當的行動。如果內容包含圖像引用，它可以調用 `recognize_image` 工作程序來協助進行視覺審查。如果任何工作程序警告潛在的有害內容，AI 可能決定將內容升級進行人工審核或直接拒絕。但根據警告的嚴重程度，AI 可能會選擇使用使用者信譽評估的結果來決定如何處理它不太確定的內容。視使用情況而定，也許受信任的使用者在發布內容時可以有更大的彈性。諸如此類...

與之前的流程管理器示例一樣，審核記錄作為工作流程執行的記錄，包括每個步驟的結果和 AI 生成的決策。這個記錄可用於審計、透明度和隨時間改進審核流程。

通過在 ContentModerator 中運用 AI，社交媒體應用程式可以根據內容特徵動態調整審核工作流程，並智能地處理複雜的審核場景。AI 組件可以做出明智的決策，優化工作流程，確保安全和健康的社群體驗。

讓我們探討另外兩個示例，這些示例展示了在智能工作流程編排環境中的預測性任務調度以及異常處理和恢復。

客戶支援系統中的預測性任務調度

在使用 Ruby on Rails 構建的客戶支援應用程式中，有效管理和優先處理支援工單對於及時為客戶提供協助至關重要。SupportTicketScheduler 組件利用 AI 根據工單緊急程度、客服人員專業知識和工作負荷等各種因素，預測性地調度並分配支援工單給可用的客服人員。

```
1  class SupportTicketScheduler
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
6      tasked with intelligently assigning tickets to available agents..."
7
8    def initialize(ticket)
9      @ticket = ticket
10     @transcript = [
11       { system: SYSTEM_DIRECTIVE },
12       { user: ticket.to_json }
13     ]
14   end
15
16   def perform
17     complete(@transcript)
18   end
19
20   def model
```

```
21     "openai/gpt-4"
22 end
23
24 def functions
25   [
26     {
27       name: "analyze_ticket_urgency",
28       # ...
29     },
30     {
31       name: "list_available_agents",
32       description: "Includes expertise of available agents",
33       # ...
34     },
35     {
36       name: "predict_agent_workload",
37       description: "Uses historical data to predict upcoming workloads",
38       # ...
39     },
40     {
41       name: "assign_ticket_to_agent",
42       # ...
43     },
44     {
45       name: "reschedule_ticket",
46       # ...
47     }
48   ]
49 end
```

```
50
51  # implementation of functions that can be called by the AI
52  # entirely at its discretion, depending on the needs of the order
53
54  def analyze_ticket_urgency
55      result = TicketUrgencyAnalyzer.perform(@ticket)
56      continue_with(result)
57  end
58
59  def list_available_agents
60      result = ListAvailableAgents.perform
61      continue_with(result)
62  end
63
64  def predict_agent_workload
65      result = AgentWorkloadPredictor.perform
66      continue_with(result)
67  end
68
69  def assign_ticket_to_agent
70      TicketAssigner.perform(@ticket, @transcript)
71  end
72
73  def delay_assignment(until)
74      until = DateTimeStandardizer.process(until)
75      SupportTicketScheduler.delay(@ticket, @transcript, until)
76  end
77
78  private
```



```
79
80     def continue_with(result)
81         @transcript << { function: result }
82         complete(@transcript)
83     end
84 end
```

在此範例中，SupportTicketScheduler 在初始化時會帶入一個支援票據物件並維護一個排程記錄。AI 組件會分析票據細節，並根據票據緊急程度、客服專員專業度以及預測的專員工作負載等因素，預測性地安排票據分配。

AI 可以調用的功能包括 analyze_ticket_urgency、list_available_agents、predict_agent_workload 和 assign_ticket_to_agent。每個功能都會將任務委派給相對應的分析器或預測器組件，並將結果附加到排程記錄中。AI 也可以使用 delay_assignment 功能來延遲分配。

AI 組件會檢查排程記錄並對票據分配做出明智的決定。它會考慮票據的緊急程度、可用客服專員的專業程度，以及每位專員預測的工作負載，以確定最適合處理該票據的專員。

透過運用預測性任務排程，客戶支援應用程式可以優化票據分配、縮短回應時間，並提升整體客戶滿意度。主動且有效率地管理支援票據，確保正確的票據在適當的時機分配給適當的客服專員。

資料處理管道中的異常處理與復原

處理異常狀況和從故障中復原對於確保資料完整性和防止資料遺失至關重要。DataProcessingOrchestrator 組件運用 AI 來智能處理異常狀況，並協調資料處理管道中的復原程序。

```
1  class DataProcessingOrchestrator
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."
6
7      def initialize(data_batch)
8          @data_batch = data_batch
9          @transcript = [
10              { system: SYSTEM_DIRECTIVE },
11              { user: data_batch.to_json }
12          ]
13      end
14
15      def perform
16          complete(@transcript)
17      end
18
19      def model
20          "openai/gpt-4"
21      end
22
23      def functions
24          [
25              {
26                  name: "validate_data",
27                  # ...
28              },
29              {
```

```
30         name: "process_data",
31         # ...
32     },
33     {
34         name: "request_fix",
35         # ...
36     },
37     {
38         name: "retry_processing",
39         # ...
40     },
41     {
42         name: "mark_data_as_failed",
43         # ...
44     },
45     {
46         name: "finished",
47         # ...
48     }
49 ]
50 end
51
52 # implementation of functions that can be called by the AI
53 # entirely at its discretion, depending on the needs of the order
54
55 def validate_data
56     result = DataValidator.perform(@data_batch)
57     continue_with(result)
58 rescue ValidationException => e
```

```
59     handle_validation_exception(e)
60 end
61
62 def process_data
63     result = DataProcessor.perform(@data_batch)
64     continue_with(result)
65 rescue ProcessingException => e
66     handle_processing_exception(e)
67 end
68
69 def request_fix(description_of_fix)
70     result = SmartDataFixer.new(description_of_fix, @data_batch)
71     continue_with(result)
72 end
73
74 def retry_processing(timeout_in_seconds)
75     wait(timeout_in_seconds)
76     process_data
77 end
78
79 def mark_data_as_failed
80     @data_batch.update!(status: 'failed', transcript: @transcript)
81 end
82
83 def finished
84     @data_batch.update!(status: 'finished', transcript: @transcript)
85 end
86
87 private
```

```
88
89     def continue_with(result)
90         @transcript << { function: result }
91         complete(@transcript)
92     end
93
94     def handle_validation_exception(exception)
95         @transcript << { exception: exception.message }
96         complete(@transcript)
97     end
98
99     def handle_processing_exception(exception)
100         @transcript << { exception: exception.message }
101         complete(@transcript)
102     end
103 end
```

在此範例中，`DataProcessingOrchestrator` 透過資料批次物件進行初始化，並維護一個處理記錄。AI 組件負責協調資料處理管道，處理異常狀況並在需要時從故障中恢復。

AI 可以調用的功能包括 `validate_data`、`process_data`、`request_fix`、`retry_processing` 和 `mark_data_as_failed`。每個功能都會將任務委派給相對應的資料處理組件，並將結果或異常詳情附加到處理記錄中。

如果在 `validate_data` 步驟中發生驗證異常，`handle_validation_exception` 功能會將異常資料附加到記錄中，並將控制權交回給 AI。同樣地，如果在 `process_data` 步驟中發生處理異常，AI 可以決定採取何種恢復策略。

根據遇到的異常性質，AI 可以自行決定是否呼叫 `request_fix`，該功能會委派給 AI 驅動的 `SmartDataFixer` 組件（參見自我修復資料章節）。資料修復器會獲得一個以簡單英文描述的指示，說明應該如何修改 `@data_batch` 以重試處理。

也許成功的重試可能需要從資料批次中移除未通過驗證的記錄，並/或將它們複製到另一個處理管道以供人工審查？可能性幾乎是無限的。

透過整合 AI 驅動的異常處理和恢復機制，資料處理應用程式變得更具彈性和容錯能力。DataProcessingOrchestrator 智能地管理異常，最小化資料損失，並確保資料處理工作流程的順暢執行。

監控與日誌記錄

監控與日誌記錄為 AI 驅動的工作流程組件的進度、效能和健康狀況提供可視性，使開發人員能夠追蹤和分析系統行為。實施有效的監控和日誌記錄機制對於除錯、稽核和持續改進智能工作流程至關重要。

監控工作流程進度和效能

為確保智能工作流程的順暢執行，監控每個工作流程組件的進度和效能非常重要。這包括在整個工作流程生命週期中追蹤關鍵指標和事件。

需要監控的重要方面包括：

- 1. 工作流程執行時間：**測量每個工作流程組件完成其任務所需的時間。這有助於識別效能瓶頸並優化整體工作流程效率。
- 2. 資源使用率：**監控每個工作流程組件對系統資源的使用情況，如 CPU、記憶體和儲存空間。這有助於確保系統在其容量範圍內運作，並能有效處理工作負載。
- 3. 錯誤率和異常：**追蹤工作流程組件中的錯誤和異常發生情況。這有助於識別潛在問題，並實現主動的錯誤處理和恢復。
- 4. 決策點和結果：**監控工作流程中的決策點和 AI 驅動決策的結果。這提供了對 AI 組件行為和效能的深入了解。

監控過程捕獲的資料可以透過儀表板顯示，或用作排程報告的輸入，以告知系統管理員系統的健康狀況。



監控資料可以輸入到 AI 驅動的系統管理員程序中進行審查和採取可能的行動！

記錄關鍵事件和決策

日誌記錄是一項重要的實踐，涉及捕獲和儲存工作流程執行期間發生的關鍵事件、決策和異常的相關資訊。

需要記錄的重要方面包括：

1. **工作流程啟動和完成：**記錄每個工作流程實例的開始和結束時間，以及任何相關的元數據，如輸入資料和使用者上下文。
2. **組件執行：**記錄每個工作流程組件的執行詳情，包括輸入參數、輸出結果和產生的任何中間資料。
3. **AI 決策和推理：**記錄 AI 組件做出的決策，以及相關的推理或信心分數。這提供了透明度，並使 AI 驅動的決策能夠被稽核。
4. **異常和錯誤訊息：**記錄工作流程執行期間遇到的任何異常或錯誤訊息，包括堆疊追蹤和相關的上下文資訊。

日誌記錄可以使用各種技術來實現，例如寫入日誌檔案、將日誌儲存在資料庫中，或將日誌發送到集中式日誌服務。選擇一個提供靈活性、可擴展性，並且易於與應用程式架構整合的日誌框架很重要。

以下是在 Ruby on Rails 應用程式中使用 ActiveSupport::Logger 類別實現日誌記錄的範例：

```
1 class WorkflowLogger
2   def self.log(message, severity = :info)
3     @logger ||= ActiveSupport::Logger.new('workflow.log')
4     @logger.formatter ||= proc do |severity, datetime, progname, msg|
5       "#{datetime} [{severity}] #{msg}\n"
6     end
7     @logger.send(severity, message)
8   end
9 end
10
11 # Usage example
12 WorkflowLogger.log("Workflow initiated for order #{@order.id}")
13 WorkflowLogger.log("Payment processing completed successfully")
14 WorkflowLogger.log("Inventory check failed for item #{item.id}", :error)
```

透過在工作流程元件和 AI 決策點中策略性地放置日誌記錄語句，開發人員可以擷取寶貴的資訊用於除錯、稽核和分析。

監控和日誌記錄的優點

在智慧工作流程編排中實施監控和日誌記錄具有以下幾個優點：

- 1. 除錯和故障排除：**詳細的日誌和監控數據幫助開發人員快速識別和診斷問題。它們提供了工作流程執行流程、元件互動以及遇到的任何錯誤或異常的洞察。
- 2. 效能優化：**監控效能指標使開發人員能夠識別瓶頸並優化工作流程元件以提高效率。透過分析執行時間、資源使用率和其他指標，開發人員可以做出明智的決策來改善系統的整體效能。
- 3. 稽核和合規：**記錄關鍵事件和決策為法規遵循和問責提供了稽核追蹤。這使組織能夠追蹤和驗證 AI 元件所採取的行動，並確保遵守業務規則和法律要求。

4. 持續改進：監控和日誌記錄數據為智慧工作流程的持續改進提供了寶貴的輸入。透過分析歷史數據、識別模式和衡量 AI 決策的效果，開發人員可以反覆細化和增強工作流程編排邏輯。

考量因素和最佳實踐

在實施智慧工作流程編排的監控和日誌記錄時，請考慮以下最佳實踐：

- 1. 定義明確的監控指標：**根據工作流程的具體要求，確定需要監控的關鍵指標和事件。專注於能夠提供系統效能、健康狀況和行為有意義洞察的指標。
- 2. 實施精細的日誌記錄：**確保在工作流程元件和 AI 決策點的適當位置放置日誌記錄語句。擷取相關的上下文資訊，如輸入參數、輸出結果和產生的任何中間數據。
- 3. 使用結構化日誌記錄：**採用結構化日誌記錄格式，以便於解析和分析日誌數據。結構化日誌記錄可以更好地搜尋、過濾和彙總日誌條目。
- 4. 管理日誌保留和輪替：**實施日誌保留和輪替政策以管理日誌檔案的儲存和生命週期。根據法律要求、儲存限制和分析需求確定適當的保留期限。如果可能，將日誌記錄轉移到第三方服務，如 [Papertrail](#)。
- 5. 保護敏感資訊：**在記錄敏感資訊時要謹慎，例如個人身份資訊（PII）或機密業務數據。實施適當的安全措施，如數據遮罩或加密，以保護日誌檔案中的敏感資訊。
- 6. 整合監控和警報工具：**利用監控和警報工具來集中收集、分析和視覺化監控和日誌記錄數據。這些工具可以提供即時洞察，根據預定義的閾值生成警報，並促進主動問題檢測和解決。我最喜歡的這類工具是 [Datadog](#)。

透過實施全面的監控和日誌記錄機制，開發人員可以獲得對智慧工作流程行為和效能的寶貴洞察。這些洞察使 AI 驅動的工作流程編排系統能夠進行有效的除錯、優化和持續改進。

可擴展性和效能考量

可擴展性和效能是設計和實施智慧工作流程編排系統時需要考慮的關鍵方面。隨著並行工作流程的數量和 AI 驅動元件的複雜性增加，確保系統能夠有效處理工作負載並無縫擴展以滿足不斷增長的需求變得至關重要。

處理大量並行工作流程

智慧工作流程編排系統通常需要處理大量並行工作流程。為確保可擴展性，請考慮以下策略：

- 1. 非同步處理：**實施非同步處理機制以解耦工作流程元件的執行。這使系統能夠同時處理多個工作流程，而無需阻塞或等待每個元件完成。非同步處理可以使用消息佇列、事件驅動架構或背景作業處理框架（如 Sidekiq）來實現。
- 2. 分散式架構：**設計系統架構以使用無伺服器元件（如 AWS Lambda）或簡單地將工作負載分散到多個節點或伺服器上，與主應用程式伺服器並行。這實現了水平可擴展性，可以添加額外的節點來處理增加的工作流程數量。
- 3. 平行執行：**識別工作流程中的平行執行機會。某些工作流程元件可能彼此獨立，可以同時執行。透過利用平行處理技術（如多執行緒或分散式任務佇列），系統可以優化資源使用並減少整體工作流程執行時間。

AI 驅動元件的效能優化

AI 驅動元件，例如機器學習模型或自然語言處理引擎，可能會耗費大量計算資源，並影響工作流程編排系統的整體效能。為了優化 AI 元件的效能，請考慮以下技術：

- 1. 快取：**如果您的 AI 處理純粹是生成式的，且在生成對話回應時不需要即時資訊查詢或外部整合，那麼您可以考慮使用快取機制來儲存和重複使用那些經常被存取或計算成本高昂的操作結果。

2. 模型優化：持續優化工作流程元件中 AI 模型的使用方式。這可能涉及諸如提示詞提煉 等技術，或者可能僅僅是在新模型推出時進行測試。

3. 批次處理：如果您正在使用 GPT-4 級別的模型，您可能可以利用批次處理技術來一次性處理多個數據點或請求，而不是個別處理它們。透過批次處理數據，系統可以優化資源利用率並減少重複模型請求的開銷。

監控和效能分析

為了識別效能瓶頸並優化智慧工作流程編排系統的可擴展性，實施監控和分析機制至關重要。請考慮以下方法：

1. 效能指標：定義並追蹤關鍵效能指標，如響應時間、吞吐量、資源使用率和延遲。這些指標能夠提供系統效能的洞察，並協助識別需要優化的領域。熱門的 AI 模型聚合器 [OpenRouter](#) 在每個 API 響應中都包含主機¹和速度²指標，使追蹤這些關鍵指標變得輕而易舉。

2. 分析工具：使用分析工具來分析個別工作流程元件和 AI 操作的效能。分析工具可以協助識別效能熱點、低效的程式碼路徑或資源密集型操作。常用的分析工具包括 New Relic、Scout，或程式語言或框架提供的內建分析器。

3. 負載測試：進行負載測試以評估系統在不同並發工作負載下的效能。負載測試有助於識別系統的可擴展性限制、檢測效能降級，並確保系統能夠在不影響效能的情況下處理預期的流量。

4. 持續監控：實施持續監控和警報機制，以主動檢測效能問題和瓶頸。設置監控儀表板和警報來追蹤關鍵效能指標（KPI），並在超過預定閾值時接收通知。這能夠實現效能問題的及時識別和解決。

¹主機時間是指從模型主機接收到串流生成的第一個位元組所需的時間，也就是所謂的「首位元組時間」。

²速度是以完成標記數除以總生成時間計算得出。對於非串流請求，延遲被視為生成時間的一部分。

擴展策略

為了處理不斷增加的工作負載並確保智慧工作流程編排系統的可擴展性，請考慮以下擴展策略：

1. **垂直擴展**：垂直擴展涉及增加個別節點或伺服器的資源（例如 CPU、記憶體）以處理更高的工作負載。當系統需要更多處理能力或記憶體來處理複雜的工作流程或 AI 操作時，這種方法是適當的。
2. **水平擴展**：水平擴展涉及向系統添加更多節點或伺服器以分散工作負載。當系統需要處理大量並發工作流程，或當工作負載可以輕易地分配到多個節點時，這種方法是有效的。水平擴展需要分散式架構和負載平衡機制來確保流量的均勻分配。
3. **自動擴展**：實施自動擴展機制，根據工作負載需求自動調整節點數量或資源。自動擴展允許系統根據傳入流量動態擴展或縮減，確保最佳的資源利用率和成本效益。像 Amazon Web Services (AWS) 或 Google Cloud Platform (GCP) 等雲端平台提供了可用於智慧工作流程編排系統的自動擴展功能。

效能優化技術

除了擴展策略外，請考慮以下效能優化技術來提升智慧工作流程編排系統的效率：

1. **高效的數據儲存和檢索**：優化工作流程元件使用的數據儲存和檢索機制。使用高效的資料庫索引、查詢優化技術和數據快取，以最小化延遲並提高數據密集型操作的效能。
2. **非同步 I/O**：利用非同步 I/O 操作來防止阻塞並提高系統的回應能力。非同步 I/O 允許系統在不等待 I/O 操作完成的情況下同時處理多個請求，從而最大化資源利用率。
3. **高效序列化和反序列化**：優化工作流程組件之間用於數據交換的序列化和反序列化過程。使用高效的序列化格式，如 Protocol Buffers 或 MessagePack，以減

少數據序列化的開銷並提高組件間通信的性能。



對於基於 Ruby 的應用程序，可以考慮使用 **Universal ID**。Universal ID 利用 MessagePack 和 Brotli（一個專為速度和最佳數據壓縮而設計的組合）。當結合使用時，這些庫比 Protocol Buffers 快 30%，且壓縮率相差僅在 2-5% 之間。

4. 壓縮和編碼：應用壓縮和編碼技術來減少工作流程組件之間傳輸的數據大小。壓縮算法，如 gzip 或 Brotli，可以顯著減少網絡頻寬使用並提高系統的整體性能。

在設計和實現智能工作流程編排系統時考慮可擴展性和性能方面，可以確保您的系統能夠處理大量並發工作流程，優化 AI 驅動組件的性能，並無縫擴展以滿足不斷增長的需求。持續監控、分析和優化工作對於隨著工作負載和複雜性隨時間增加而維持系統的性能和響應能力至關重要。

工作流程的測試和驗證

測試和驗證是開發和維護智能工作流程編排系統的關鍵方面。鑑於 AI 驅動工作流程的複雜性，確保每個組件按預期運行、整體工作流程行為正確，以及 AI 決策準確可靠是至關重要的。在本節中，我們將探討測試和驗證智能工作流程的各種技術和注意事項。

工作流程組件的單元測試

單元測試涉及獨立測試個別工作流程組件，以驗證其正確性和穩健性。在對 AI 驅動的工作流程組件進行單元測試時，請考慮以下幾點：

1. 輸入驗證：測試組件處理不同類型輸入的能力，包括有效和無效數據。驗證組件是否能夠優雅地處理邊界情況並提供適當的錯誤訊息或異常。

2. **輸出驗證**：斷言組件對給定的輸入集產生預期的輸出。將實際輸出與預期結果進行比較以確保正確性。

3. **錯誤處理**：通過模擬各種錯誤場景來測試組件的錯誤處理機制，如無效輸入、資源不可用或意外異常。驗證組件是否適當地捕獲和處理錯誤。

4. **邊界條件**：測試組件在邊界條件下的行為，如空輸入、最大輸入大小或極端值。確保組件能夠優雅地處理這些條件而不會崩潰或產生錯誤結果。

以下是使用 RSpec 測試框架對工作流程組件進行單元測試的示例：

```
1  RSpec.describe OrderValidator do
2    describe '#validate' do
3      context 'when order is valid' do
4        let(:order) { build(:order) }
5
6        it 'returns true' do
7          expect(subject.validate(order)).to be true
8        end
9      end
10
11     context 'when order is invalid' do
12       let(:order) { build(:order, total_amount: -100) }
13
14       it 'returns false' do
15         expect(subject.validate(order)).to be false
16       end
17     end
18   end
19 end
```

在這個例子中，OrderValidator 元件使用兩個測試案例進行測試：一個用於有

效訂單，另一個用於無效訂單。這些測試案例驗證 `validate` 方法是否根據訂單的有效性返回預期的布林值。

整合測試工作流程互動

整合測試著重於驗證不同工作流程元件之間的互動和資料流。它確保元件之間能夠無縫協作並產生預期的結果。在對智慧工作流程進行整合測試時，需要考慮以下幾點：

1. **元件互動**：測試工作流程元件之間的通訊和資料交換。驗證一個元件的輸出是否正確地作為工作流程中下一個元件的輸入。
2. **資料一致性**：確保資料在工作流程中流動時保持一致且準確。驗證資料轉換、計算和彙總是否正確執行。
3. **異常傳播**：測試異常和錯誤如何在工作流程元件之間傳播和處理。驗證異常是否被適當地捕獲、記錄和處理，以防止工作流程中斷。
4. **非同步行為**：如果工作流程涉及非同步元件或平行執行，測試協調和同步機制。確保工作流程在並發和非同步場景下能夠正確運作。

以下是使用 RSpec 測試框架對 Ruby 工作流程進行整合測試的例子：

```
1 RSpec.describe OrderProcessingWorkflow do
2
3   let(:order) { build(:order) }
4
5   it 'processes the order successfully' do
6     expect(OrderValidator).to receive(:validate).and_return(true)
7     expect(InventoryManager).to receive(:check_availability).and_return(true)
8     expect(PaymentProcessor).to receive(:process_payment).and_return(true)
9     expect(ShippingService).to receive(:schedule_shipping).and_return(true)
10
```

```
11     workflow = OrderProcessingWorkflow.new(order)
12     result = workflow.process
13
14     expect(result).to be true
15     expect(order.status).to eq('processed')
16 end
17
18 end
```

在此範例中，通過驗證不同工作流程組件之間的互動來測試 `OrderProcessingWorkflow`。測試案例為每個組件的行為設定預期，並確保工作流程能夠成功處理訂單，相應地更新訂單狀態。

測試 AI 決策點

測試 AI 決策點對於確保 AI 驅動的工作流程的準確性和可靠性至關重要。在測試 AI 決策點時，請考慮以下幾點：

1. **決策準確性：**驗證 AI 組件能根據輸入數據和訓練模型做出準確的決策。將 AI 決策與預期結果或真實標準數據進行比較。
2. **邊界案例：**測試 AI 組件在邊界案例和異常情況下的行為。驗證 AI 組件能夠妥善處理這些情況並做出合理的決策。
3. **偏見與公平性：**評估 AI 組件是否存在潛在偏見，並確保其做出公平且無偏見的決策。使用多樣化的輸入數據測試組件，並分析結果中是否存在任何歧視性模式。
4. **可解釋性：**如果 AI 組件為其決策提供解釋或推理，請驗證解釋的正確性和清晰度。確保解釋與底層決策過程相符。

以下是使用 RSpec 測試框架在 Ruby 中測試 AI 決策點的示例：


```
1  RSpec.describe FraudDetector do
2    describe '#detect_fraud' do
3      context 'when transaction is fraudulent' do
4        let(:tx) do
5          build(:transaction, amount: 10_000, location: 'High-Risk Country')
6        end
7
8        it 'returns true' do
9          expect(subject.detect_fraud(tx)).to be true
10        end
11      end
12
13      context 'when transaction is legitimate' do
14        let(:tx) do
15          build(:transaction, amount: 100, location: 'Low-Risk Country')
16        end
17
18        it 'returns false' do
19          expect(subject.detect_fraud(tx)).to be false
20        end
21      end
22    end
23  end
```

在此範例中，`FraudDetector` AI 元件使用兩個測試案例進行測試：一個用於欺詐交易，另一個用於合法交易。這些測試案例驗證 `detect_fraud` 方法是否根據交易的特徵回傳預期的布林值。

端對端測試

端對端測試涉及從開始到結束測試整個工作流程，模擬真實世界的情境和使用者互動。它確保工作流程運作正確並產生預期的結果。在執行智慧工作流程的端對端測試時，請考慮以下幾點：

1. **使用者情境：**識別常見的使用者情境，並測試工作流程在這些情境下的行為。驗證工作流程是否正確處理使用者輸入、做出適當的決策並產生預期的輸出。
2. **資料驗證：**確保工作流程驗證和淨化使用者輸入，以防止資料不一致或安全漏洞。使用各種類型的輸入資料測試工作流程，包括有效和無效的資料。
3. **錯誤恢復：**測試工作流程從錯誤和異常中恢復的能力。模擬錯誤情境並驗證工作流程是否能妥善處理這些情況、記錄錯誤並採取適當的恢復措施。
4. **效能和擴展性：**評估工作流程在不同負載條件下的效能和擴展性。使用大量並發請求測試工作流程，並測量回應時間、資源使用情況和整體系統穩定性。

以下是使用 Ruby 的 RSpec 測試框架和用於模擬使用者互動的 Capybara 函式庫進行工作流程端對端測試的範例：

```
1 RSpec.describe 'Order Processing Workflow' do
2   scenario 'User places an order successfully' do
3     visit '/orders/new'
4     fill_in 'Product', with: 'Sample Product'
5     fill_in 'Quantity', with: '2'
6     fill_in 'Shipping Address', with: '123 Main St'
7     click_button 'Place Order'
8
9     expect(page).to have_content('Order Placed Successfully')
10    expect(Order.count).to eq(1)
11    expect(Order.last.status).to eq('processed')
```

12 **end**

13 **end**

在這個例子中，端對端測試模擬使用者透過網頁界面下訂單的過程。它會填寫必要的表單欄位、提交訂單，並驗證訂單是否成功處理，包括顯示適當的確認訊息以及更新資料庫中的訂單狀態。

持續整合與部署

為了確保智慧工作流程的可靠性和可維護性，建議將測試和驗證整合到持續整合與部署（CI/CD）管道中。這能夠在工作流程部署到正式環境之前，自動執行測試和驗證的程序。請考慮以下實務做法：

1. **自動化測試執行：**設定 CI/CD 管道，在工作流程程式碼發生變更時自動執行測試套件。這確保能在開發過程的早期發現任何回歸錯誤或故障。
2. **測試覆蓋率監控：**測量並監控工作流程組件和 AI 決策點的測試覆蓋率。以達到高測試覆蓋率為目標，確保關鍵路徑和場景都經過徹底測試。
3. **持續反饋：**將測試結果和程式碼品質指標整合到開發工作流程中。在 CI/CD 過程中，持續向開發人員提供有關測試狀態、程式碼品質和發現問題的反饋。
4. **預備環境：**將工作流程部署到與正式環境相近的預備環境中。在預備環境中執行額外的測試和驗證，以發現任何與基礎架構、配置或資料整合相關的問題。
5. **回滾機制：**實作回滾機制，以應對部署失敗或在正式環境中發現的重大問題。確保工作流程能夠快速回復到先前的穩定版本，以減少系統停機時間和對使用者的影響。

透過在智慧工作流程的開發生命週期中納入測試和驗證，組織可以確保其 AI 驅動系統的可靠性、準確性和可維護性。定期的測試和驗證有助於發現錯誤、防止回歸問題，並建立對工作流程行為和結果的信心。

第二部分：模式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

提示工程

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

思維鏈

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

工作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

內容生成

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

結構化實體建立

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

LLM 代理指導

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

優點與考慮因素

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

模式切換

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

角色分配

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

提示物件

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

提示詞模板

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

優點和注意事項

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機：

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

結構化輸入輸出

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

擴展結構化輸入輸出

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

優點與注意事項

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

提示鏈接

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例：Olympia 的註冊流程

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

提示重寫器

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

工作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

回應圍欄

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

優點和注意事項

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

錯誤處理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

查詢分析器

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

實作

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

詞性標記（POS）和命名實體識別（NER）

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

意圖分類

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

關鍵字提取

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

優點

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

查詢重寫器

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

優點

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

腹語術師模式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

工作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

離散元件

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

謂詞

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

API 外觀模式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

主要優點

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

範例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

認證和授權

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

請求處理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

響應格式化

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

錯誤處理和邊緣情況

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

可擴展性和性能考慮

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

與其他設計模式的比較

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

結果解譯器

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

範例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

虛擬機器

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

使用時機

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

範例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

魔法背後的原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

規格與測試

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

定義行為規格

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

編寫測試案例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

範例：測試翻譯元件

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

HTTP 互動重播

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

人機協作系統 (HITL)

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

高層次模式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

混合智能

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

自適應響應

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

人機角色轉換

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

升級處理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

主要優點

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

現實應用：醫療保健

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

反饋循環

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

工作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

應用和示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

人工反饋整合的進階技術

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

被動資訊輻射

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

情境資訊顯示

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

主動通知

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

解釋性見解

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

互動式探索

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

主要優勢

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

應用和範例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

協作決策制定 (CDM)

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

範例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

持續學習

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

應用和示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

倫理考量

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

人機協作在緩解 AI 風險中的作用

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

技術進步與未來展望

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

人機協作系統的挑戰與限制

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

智慧錯誤處理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

傳統錯誤處理方法

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

情境錯誤診斷

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

情境錯誤診斷的提示詞工程

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

用於情境式錯誤診斷的檢索增強生成

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

智慧型錯誤報告

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

預測性錯誤預防

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

智能錯誤恢復

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

個人化錯誤通訊

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

自適應錯誤處理工作流程

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

品質控制

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

評估器

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

問題

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

解決方案

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

示例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

注意事項

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

理解黃金標準參考

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

無參考評估的運作方式

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

護欄

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

問題

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

解決方案

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

運作原理

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

範例

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

注意事項

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

防護機制和評估：一體兩面

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

防護機制和無參考評估的互換性

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

實現雙重用途的防護機制和評估

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

詞彙表

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

詞彙表

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

A

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

B

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

C

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

D

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

E

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

F

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

G

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

H

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

I

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

J

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

K

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

L

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

M

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

N

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

O

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

P

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

Q

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

R

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

S

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

T

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

U

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

V

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

W

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

Z

這部分內容並未包含在樣本書中。您可以在 Leanpub 上購買這本書，網址是 <http://leanpub.com/patterns-of-application-development-using-ai-zh-Hant>

Index

- ACID 特性, 92
- AI, 84, 109, 122, 128, 178
 - applications, 118
 - model, 83, 84
 - 對話式, 5, 179
 - 應用程式, 106, 140
 - 模型, 73, 137, 178
 - 決策點, 220
- AI 工作者串接, 93
- Alpaca, 10
- Altman, Sam, 13
- Amazon Web Services, 216
- Anthropic, 17, 29, 57, 109, 117
- API, 104
- APIs, 56
- audit logging, 90
- BERT, 10, 18
- boundary conditions, 218
- Brotli, 217
- Byte Pair Encoding (BPE), 11
- Capybara 函式庫, 222
- Chain of Thought (CoT), 119
- ChatGPT, 22, 40
- classification, 102
- Claude, 6, 32, 61
- Claude 3, 37, 107, 109, 115, 117
- Claude 3 Opus, 58
- Claude v1, 13
- Claude v2, 13
- Cohere (LLM 提供商), 17
- Cohere (大型語言模型供應商) , 18
- concurrent workflows, 217
- context
 - infinitely long inputs, 12
 - window, 11
- Customer Sentiment Analysis, 84
- C (程式語言) , 98
- data
 - persistence, 92
 - preparation, 92
- Databricks 員工, 39
- Datadog, 213
- decision
 - making capabilities, 84
- document clustering, 102
- Dohan, et al., 32
- ELK stack, 93
- errors
 - handling, 90, 218
 - 智能錯誤處理, 122

- event-driven architecture, 91
- F#, 77
- Facebook, 18
- finalize 方法, 136, 137
- FitAI, 179
- Gemma 7B, 8
- GitLab, 77
- Google, 17
 - API, 48, 51
 - Cloud Platform, 216
 - Gemini, 16
 - Gemini 1.5 Pro, 10, 13, 14
 - PaLM (Pathways Language Model), 13
 - PaLM (Pathways 語言模型), 18
 - T5, 10
 - 雲端 AI 平台, 18
- GPT-3, 10, 12
- GPT-4, 5, 10, 13, 16, 23, 32, 37, 48, 88, 98,
 - 101, 108, 113, 172, 173, 215
- Graham, Paul, 14
- GraphQL, 91
- Groq, 19, 101
- gzip, 217
- Hohpe, Gregor, 88
- Honeybadger, 78
- HTTP, 128
- input
 - validation, 217
- intelligent workflow orchestration, 217
- JSON (JavaScript Object Notation), 112,
 - 115, 126
- JSON (JavaScript 物件表示法), 107
- JSON (JavaScript 物件表示法) , 143
- K-means, 102
- language
 - models, 31
- Large Language Model (LLM), 12, 13, 101,
 - 172
- Latent Dirichlet Allocation, 102
- Llama, 10
- Llama 2-70B, 37
- Llama 3 70B, 8
- Llama 3 8B, 8
- Louvre, 31
- Managed Streaming for Apache Kafka, 30
- Markdown, 126
- Memorial Sloan Kettering Cancer Center,
 - 30
- MessagePack, 216
- Meta, 18
- Metropolitan Museum of Art, 31
- Mistral, 19
 - 7B, 8
 - 7B Instruct, 13, 173
- Mixtral
 - 8x22B, 8
 - 8x7B, 43
- Multimodal
 - language models, 16

- models, 15
- Naive Bayes, 102
- narrative building, 15
- natural language
 - Natural Language Processing (NLP), 101
- New Relic, 215
- Ollama, 19
- Olympia, 25, 48, 109, 122, 129, 143
- Olympia 的知識庫, 75
- OpenAI, 3, 17, 29, 57
- OpenRouter, 20, 21, 129, 215
- OPT 模型, 18
- output verification, 218
- Perplexity (提供商), 8
- Process Manager, 91
- Protocol Buffers, 216
- publish-subscribe systems, 91
- PyTorch, 18
- Qwen2 70B, 8
- Rails, 164
- Railway Oriented Programming (ROP), 78
- Raix, 194
 - 函式庫, 81
- RSpec, 218, 219, 222
- Ruby, 77, 78, 95, 140, 222
- Ruby on Rails, 1, 93, 193, 202
- Rudall, Alex, 17
- Rust (Programming Language), 77
- Rust (程式語言), 98
- Scout, 215
- server-sent events (SSE), 128
- SQL 注入, 55
- stream handlers, 128
- stream processing, 128
- Stripe, 110
- Support Vector Machines (SVM), 102
- system directive, 83
- T5, 18
- Together.ai, 19
- Top-k 採樣, 36
- Top-p (核採樣), 35
- topic identification, 102
- Unicode-encodable language, 11
- Universal ID, 217
- Wall, Larry, 2
- Wisper, 78, 90, 129, 137
- Wooley, Chad, 77
- XML, 114
- Yi-34B, 37
- 一致性
 - 和可重現性, 113
- 上下文
 - 增強, 34
- 串流處理, 134, 140
 - 邏輯, 136
- 串流資料, 130

- 主動式, 24
- 交通管理, 24
- 人工介入, 193
- 人工智慧, 58, 171
 - 對話式, 23
 - 模型, 133, 135
 - 複合式系統, 23
- 人工智能, 51, 114
 - 複合系統, 22, 25
- 人工智能！應用程式, 127
- 人機協作 (HITL), 153
- 企業應用程式架構, 28
- 企業整合模式, 88
- 位元組對編碼 (BPE), 9
- 使用者介面
 - 設計, 185
- 使用者介面 (UI)
 - 介面, 167, 181
 - 技術, 178
 - 框架, 181
- 使用者信任, 184
- 使用者心理學, 182
- 使用者測試與回饋, 166
- 使用者生成內容, 93
- 使用者體驗, 164
- 供應鏈
 - 優化, 24
- 保險驗證, 86
- 個人化, 160, 184, 188
 - 個人化微文案, 175
 - 個人化表單, 168
- 個人化商品推薦, 76
- 倫理
 - 影響, 168
- 偏見
 - AI 中的偏見與公平性, 220
- 傀儡師, 151
- 備用策略, 92
- 內容
 - 內容分類, 94
 - 過濾, 20
- 全域直譯器鎖 (GIL), 97
- 公地悲劇, 162
- 函式
 - 呼叫, 104
 - 呼叫歷史記錄, 134
- 函數呼叫失敗, 114
- 函數式程式設計, 76
- 函數！名稱, 132
- 分散式架構, 214
- 分詞, 9
- 分類, 39
- 分類和目標策略, 164
- 創意寫作, 25, 39
- 動態 UI 生成, 160
- 動態任務路由, 188
- 動態工具選擇, 112
- 包容性介面, 168
- 協同過濾, 76
- 參數
 - 參數數量, 21
 - 效果, 109
 - 範圍, 8

反饋

- 反饋循環, 45

- 可擴展性, 188, 214

- 可用性問題, 183

- 可解釋性, 220

- 合成數據生成, 40

- 吞吐量, 20

命令列

- 命令列介面 (CLI), 19

- 商品推薦, 76

- 問答系統, 5

- 單樣本學習, 46

- 回應圍欄, 151, 174

- 回滾機制, 223

- 國際化, 164

- 圖形模型, 32

- 基於內容的過濾, 76

- 基礎模型, 40

- 墨丘利（羅馬神）, 33

- 外部服務或 API, 106

多代理

- 問題解決器, 23

- 多工作器模式, 100

- 多數投票法, 98

- 多步驟工作流程, 93

- 多重工作器, 143

- 大型語言模型, 141, 143

- 大型語言模型 (LLM), 1, 3, 22, 52, 53, 56,
60, 62, 72, 93, 104, 105, 114, 120,
123, 124, 126, 167, 178, 196

- 領域, 20

- 大型語言模型 (LLM) , 159

- 字典, 112

- 存在懲罰, 36

- 客戶支援, 24

- 客服聊天機器人, 25

- 客製化, 20

實驗

- 框架, 164

- 封閉式和開放式問答, 39

對話

- 循環, 137

- 紀錄, 135, 137

- 少樣本學習, 47

- 少樣本提示, 48

- 工具使用, 104, 127

- 工具呼叫, 131

- 帳戶, 75

- 平板電腦, 185

- 平行執行, 214

- 延遲, 20

- 強制工具選擇, 112

- 微服務架構, 73

- 微調, 64

- 心智理論, 29

- 快取, 214

- 思維鏈, 33

性能

- 權衡, 4

- 悲觀鎖定, 92

情境

- 情境內容生成, 159, 167, 168

- 情境內容生成模式, 162, 163

- 情境欄位建議, 169
- 情境決策制定, 190
- 範圍, 190
- 情感分析, 12, 85, 99
- 情感基調, 124
- 情緒分析, 94, 95, 115, 124
- 應用程式介面, 131
- 應用程式設計和框架, 167
- 應用程式開發, 186
- 批次處理, 215
- 持續整合與部署 (CI/CD), 223
 - 管道, 223
- 持續風險監測, 87
- 指令微調, 7
- 指令調校
 - 指令調校模型, 39
- 指令調校！指令調校模型, 37
- 排序器, 26
- 推理, 4
- 提煉過程, 60
- 提示
 - 工程, 33, 34, 45, 51, 182
 - 提示模板, 45
 - 提示精煉, 34
 - 鏈接, 45
- 提示詞
 - 串接, 56
 - 工程, 30, 43, 52
 - 提示詞提煉, 57, 62, 215
 - 提示詞模板, 174
 - 提示詞物件, 58
 - 改進, 53
 - 設計, 44, 53
- 摘要, 39
- 擬人化, 54
- 擴增實境眼鏡, 185
- 收窄路徑, 29
- 改寫, 40
- 效率, 187
- 效能
 - 優化, 113, 165
 - 問題, 215
- 效能優化, 212
- 教育應用, 24
- 整合 LLM, 159
- 整合測試, 219
- 數位環境, 163
- 數據
 - 分析, 25, 126
- 文字清理, 94
- 斷路器邏輯, 140
- 日誌保留和輪替, 213
- 智慧型手機, 185
- 智慧工作流程編排, 215
- 智能內容審核器, 197
- 智能工作流程編排, 186, 193
- 最小權限原則, 56
- 本地開發環境, 133
- 桌上型電腦, 185
- 業務規則, 186
- 概念和實務挑戰, 168
- 概率模型, 32
- 樂觀鎖定, 92
- 標記式標籤, 55

- 模式匹配, 130
- 模組化, 73
- 檢索增強生成 (RAG), 23, 28, 34, 63, 106
- 檢索式模型, 5
- 歷史模式, 190
- 水星 (行星), 33
- 汞 (元素), 33
- 決策
 - 樹, 187
 - 用例, 113
 - 點, 210
- 流程管理器, 88
 - 企業整合, 193
- 溫度參數, 40
- 漸進式揭露, 176
- 潛在空間, 30, 31
- 激勵策略, 181
- 無狀態, 135
- 無監督學習, 3
- 無障礙設計, 183, 184
- 現代應用程式, 188
- 瓶頸, 190
- 生態系統, 126
- 生成式使用者介面 (GenUI), 167, 174, 175, 178, 181, 184
- 生成式預訓練轉換器 (GPT), 6, 53
- 生產力, 161
- 異常處理, 191, 193
- 病史收集, 85
- 症狀評估和分層, 85
- 監控
 - 和警報, 191
 - 與日誌記錄, 93
 - 監控和日誌記錄, 212
 - 監控指標, 213
 - 知識庫, 5
 - 知識管理, 24
 - 硬體, 21
 - 確定性行為, 44
 - 神經網路, 3, 5
 - 票據分配, 205
 - 稽核和合規, 212
 - 端對端測試, 222, 223
 - 精細的日誌記錄, 213
 - 系統指令, 109
 - 終結方法, 134
 - 結果解釋器, 122
 - 結構化數據, 114
 - 結構化日誌記錄, 213
 - 結構化輸入輸出, 174
 - 網路連接, 191
 - 緊急響應規劃, 24
 - 線上零售商, 174
 - 線性代數, 32
 - 線性迴歸, 32
 - 縮小路徑, 28
 - 翻譯, 12, 165
 - 聊天機器人應用程式, 100
 - 臨床決策支援, 87
 - 自動擴展, 216
 - 自動續接, 138
 - 自我修復數據, 141

- 自我修復資料, 209
- 自然語言
 - 自然語言處理, 85
- 自迴歸建模, 32
- 自適應 UI, 177
- 自適應工作流程
 - 自適應工作流程組合, 190
- 處理時間, 93
- 虛擬助理, 25
- 複雜任務, 125
- 視覺介面, 178
- 角色扮演式互動, 5
- 觸發訊息, 88
- 訓練數據, 31
- 詐欺偵測
 - 系統, 81
- 詞元, 4, 9
- 語法規則, 3
- 語法錯誤, 112
- 語言
 - 模型, 51, 57
 - 相關任務, 3
 - 語言偵測, 94
- 語音控制界面, 25
- 資料
 - 完整性, 205
 - 流程, 92
 - 處理任務, 106
 - 處理管道, 205
 - 資料同步, 92
 - 資料檢索, 92
 - 資料驗證, 222
 - 隱私, 20, 183
- 資料庫, 104
- 支援物件, 88
- 鎖定策略, 92
- 資訊
 - 擷取, 39
 - 檢索, 5, 106
- 超參數, 34
- 跨模態生成, 16
- 軟體架構, 2
- 輸入
 - 提示, 43
- 輸入參數, 109
- 轉換器架構, 4
- 迭代優化, 60, 123
- 追蹤關鍵指標, 210
- 邊緣案例, 44
- 醫學發現, 85
- 重複懲罰, 38
- 重試機制, 92
- 量化, 21
- 錯誤
 - 恢復, 222
 - 比率, 93
 - 處理, 92, 122
- 開源模型託管服務供應商, 174
- 開發框架, 127
- 關鍵模式, 188
- 陣列, 112
- 除錯, 190
- 和測試, 113

除錯和故障排除, 212

集成系統, 98, 100

 工作器集成, 99

雜湊, 130

零樣本學習, 45

電子商務, 162, 187

電子商務應用, 75

電腦科學, 55, 57

靈活性和創造力, 165

非同步處理, 214

預備環境, 223

預測, 4

風險分層, 86

風險因子, 79, 81

首字符響應時間 (TTFT), 21

高性能完成, 19