對本書的讚譽

這絕對是我讀過關於提示工程的書籍中，最棒的資源；*Mike* 和 *James* 在他們的領域中堪稱大師。

——*Dan Shipper*，共同創辦人與 *CEO*，*Every*

這本書是提示工程和生成式 *AI* 基礎知識的可靠入門書。作者以簡單、實用且易懂的方式，涵蓋從初學者到進階水準的廣泛實用技術。如果想要提升 *AI* 系統的準確性和可靠性，這本書理應在你的書架上占有一席之地。

——*Mayo Oshin*，創辦人與 *CEO*，*Siennai Analytics*，*LangChain* 早期貢獻者

Phoenix 和 *Taylor* 的指南就像生成式 *AI* 無邊海洋中的一座燈塔。他們的書是我們 *Phiture AI Labs* 團隊中的基石，幫助我們學會如何運用大型語言模型和擴散模型，來創造符合客戶應用和遊戲精髓的行銷資產。透過提示工程，我們能夠大規模生成符合品牌風格的客製化內容。這本書可不是空有理論，而是一堂說明如何將 *AI* 的潛力轉化為量身定制解決方案的實務大師班。對於想在創意與效率方面，將 *AI* 整合提升到嶄新高度的開發者來說，這本書絕對值得一讀。

——*Moritz Daan*，創辦人 / 合夥人，*Phiture Mobile Growth Consultancy*

《生成式 *AI* 中的提示工程》可能是未來保障你技術職涯的最佳方式。這無疑是任何從事 *AI* 實務應用的人最值得擁有的資源。不論是新手或經驗老到的 *AI* 工程師，本書中豐富且精細的原則，都有助於他們在可預見的未來中保持競爭力。

——*Ellis Crosby*，*CTO* 與共同創辦人，*Incremento*

這本書對於代理機構和專業服務團隊來說是必備指南。透過將 *AI* 來整合服務與客戶交付、運用自動化管理，以及加速解決方案，在各方面都會樹立新的行業標準。您在本書中會找到各種實用資訊和戰術，幫助你充分理解並利用 *AI* 的潛力。

——*Byron Tassoni-Resch*，*CEO* 與共同創辦人，*WeDiscover*

這是一本既有趣又內容豐富的書，將實用技巧與扎實的基礎知識巧妙地結合起來。生成式 *AI* 的世界正以驚人的速度發展，擁有一套無論底層模型如何更替都能交付成果的工具組，可說是價值非凡！

——*Riaan Dreyer*，數位與資料長，冰島銀行

本書作者巧妙地將提示工程的複雜性，轉化為一套適用於文字和圖像生成的實用工具包。這份指南涵蓋了從標準操作到尖端技術，為讀者提供充分運用生成式 *AI* 模型威力的各種實用技巧。

——*Aditya Goel*，生成式 *AI* 顧問

第四章

使用 LangChain 的進階文字生成技巧

簡單的提示工程技術一般來說已能夠解決大多數的任務，但有時你需要使用更強大的工具套件包來解決複雜的生成式 AI 問題，這類問題與任務包括：

上下文長度

將整本書總結成一個易於理解的概要。

結合一系列的 *LLM* 輸入 / 輸出

為一本書建立故事，包括角色、情節和建構世界觀。

執行複雜的推理任務

讓 LLM 作為代理。例如，可以建立一個 LLM 代理來幫助你實現個人的健身目標。

要熟練地處理這些複雜的生成式 AI 挑戰，熟悉 LangChain（開放原始碼框架）是非常有效的作法。這套工具能夠大大簡化並增強你的 LLM 工作流程。

簡介 LangChain

LangChain 是一個多功能框架，有助於建立各種運用 LLM 的應用程式，並提供了 Python（*https://oreil.ly/YPid-*）和 TypeScript（*https://oreil.ly/5Vl0W*）版本的套件包。其核心理念是，最具影響力和獨特的應用程式，不只是透過 API 來對接語言模型，還要能夠做到以下事項：

增強資料意識

該框架的目標是建立語言模型與外部資料來源之間的無縫串接。

增強代理能力

該框架致力於讓語言模型能與環境互動並產生影響。

LangChain 框架請參考圖 4-1，提供一系列的模組抽象化，這些抽象化對於運算 LLM 來說相當關鍵，並且對這些抽象化提供了相當廣泛的實作選項。

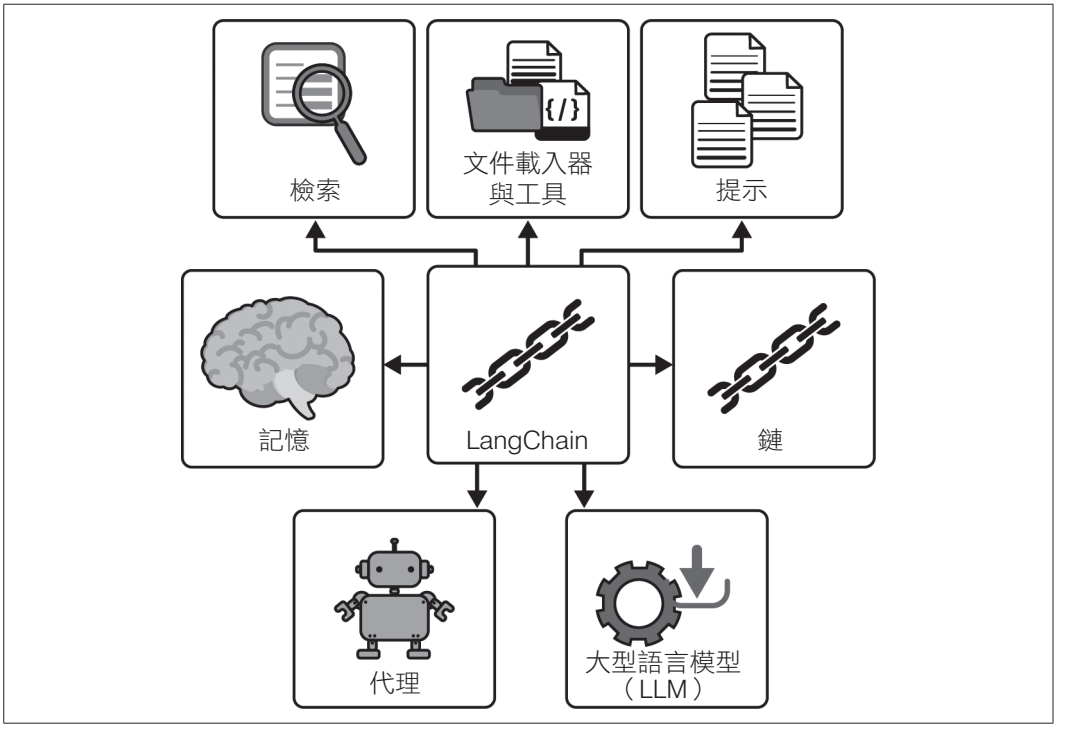


圖 4-1　LangChainLLM 框架的主要模組

每個模組都設計得相當易用，可以高效率地獨立或搭配使用。目前 LangChain 中有六個常見模組：

模型 *I/O*

處理與模型相關的輸入 / 輸出運算

檢索

專注於為 LLM 檢索相關文字

鏈

也稱為 *LangChain 可運行單元*，鏈使我們得以建構一系列 LLM 運算或函式呼叫

代理

允許鏈根據高階指示或指令說明來決定要使用的工具

記憶

在不同鏈的執行之間長久儲存應用程式的狀態

回呼

用於在特定事件（例如每次生成新的標記時）執行額外的程式碼

環境設定

請在終端機中使用以下任一指令來安裝 LangChain：

- `pip install langchain langchain-openai`
- `conda install -c conda-forge langchain langchain-openai`

如果想要安裝本書會用到的所有套件，請用本書 GitHub 中的 *requirements.txt*（*https://oreil.ly/WKOma*）檔。

建議在虛擬環境中安裝這些套件：

建立一個虛擬環境

```
python -m venv venv
```

啟動虛擬環境

```
source venv/bin/activate
```

安裝相依套件

```
pip install -r requirements.txt
```

LangChain 需要整合一或多個模型提供者。例如，如果要使用 OpenAI 的模型 API，就要用 `pip install openai` 來安裝其 Python 套件。

如第一章所述，最佳做法是在終端機中設定名為 `OPENAI_API_KEY` 的環境變數，或使用 `python-dotenv`（*https://oreil.ly/wvuO7*）來從 *.env* 檔中載入。然而就原型設計來說可以跳過這一步，做法是在 LangChain 中載入聊天模型時直接送入你的 API 金鑰：

```
from langchain_openai.chat_models import ChatOpenAI
chat = ChatOpenAI(api_key="api_key")
```

由於安全原因，不建議在腳本中寫明 API 金鑰。反之應改用環境變數或設定檔來管理你的金鑰。

隨著 LLM 領域的不斷發展，你可能會面臨不同模型 API 之間差異所帶來的挑戰。介面缺乏標準化會在提示工程中增加更多複雜性，並阻礙你在專案中無縫整合不同模型的過程。

這就是 LangChain 登場的時候啦。作為一個全面性的框架，LangChain 讓你能夠輕鬆應對不同模型的各種介面。

LangChain 的功能確保你不必在每次切換模型時都得重新設計提示或程式碼。這種超越平台的方式促進了對 Anthropic（*https://www.anthropic.com*）、Vertex AI（*https://cloud.google.com/vertex-ai*）、OpenAI（*https://openai.com*）和 BedrockChat（*https://oreil.ly/bedrock*）等諸多模型的快速實驗。這不僅加快了模型評估過程，還藉由簡化複雜的模型整合流程，而省下寶貴的時間和資源。

後續段落會用到 OpenAI 套件與其在 LangChain 中的 API。

聊天模型

像 GPT-4 這樣的聊天模型已成為與 OpenAI API 互動的主要方式。相較於提供最基礎的「輸入文字、輸出文字」回應，它們提出的是以聊天訊息作為輸入和輸出元素的互動方法。

使用聊天模型生成 LLM 回應，代表將一或多筆訊息輸入到聊天模型中。就 LangChain 來說，目前可接受的訊息類型有 `AIMessage`、`HumanMessage` 和 `SystemMessage`。聊天模型的輸出則永遠是 `AIMessage`。

SystemMessage

代表應作為 AI 系統的指示資訊，這些訊息會以某種方式來引導 AI 的行為或動作。

HumanMessage

代表來自與 AI 系統互動的人類資訊。這可能是一個問題、一個命令或人類使用者需要 AI 處理和回應的任何其他輸入。

AIMessage

代表來自 AI 系統本身的資訊。這通常是 AI 對 `HumanMessage` 的回應，或針對 `SystemMessage` 指示的結果。

務必要使用 `SystemMessage` 來提供明確的方向。OpenAI 已改進 GPT-4 和即將推出的 LLM 模型，使它們會更加注意這類訊息中所給定的指引。

讓我們用 LangChain 建立一個笑話生成器吧。

Input:

```
from langchain_openai.chat_models import ChatOpenAI
from langchain.schema import AIMessage, HumanMessage, SystemMessage

chat = ChatOpenAI(temperature=0.5)
messages = [SystemMessage(content='''Act as a senior software engineer
at a startup company.'''),
（messages = [SystemMessage(content='''請扮演一家新創公司的資深
軟體工程師。'''),）
HumanMessage(content='''Please can you provide a funny joke
about software engineers?''')]
（HumanMessage(content='''你可以講一個關於軟體工程師的有趣笑話
嗎？''')]）
response = chat.invoke(input=messages)
print(response.content)
```

Output:

```
Sure, here's a lighthearted joke for you:
Why did the software engineer go broke?
Because he lost his domain in a bet and couldn't afford to renew it.
（當然可以，這裡有一個輕鬆的小笑話：
為什麼那個軟體工程師破產了？
因為他在某次賭博中輸掉了他的網域，而且還沒錢續約。）譯註
```

譯註　domain 除了指財產之外，在網路 / 軟體領域也代表網域。

在上述程式碼中，首先匯入了 ChatOpenAI、AIMessage、HumanMessage 和 SystemMessage。然後，建立一個 temperature 參數為 0.5（代表隨機性）的 ChatOpenAI 類別實例。

建立模型後，會將用於定義 LLM 角色的 SystemMessage 物件，以及用於請求一個與軟體工程師相關笑話的 HumanMessage 物件，一併填入名為 messages 的清單中。

使用 .invoke(input=messages) 呼叫聊天模型，並將一個由訊息所組成的清單提供給 LLM，接著用 response.content 來取得 LLM 的回應。

還有一個可使用 chat(messages=messages) 來直接呼叫 chat 物件的傳統方法：

```
response = chat(messages=messages)
```

串流聊天模型

你可能注意到在使用 ChatGPT 時，文字是逐字回傳的。這種獨特的回應生成模式稱為串流處理，它在提升基於聊天的應用程式效能方面扮演著重要的角色：

```
for chunk in chat.stream(messages):
    print(chunk.content, end="", flush=True)
```

在呼叫 chat.stream(messages) 時，它會一次產生一段訊息。這代表聊天訊息的每一段都是單獨回傳的。每段到達之後就會立刻顯示在終端機中並更新。這樣一來，串流處理就能把 LLM 回應的延遲降到最低。

從終端使用者的角度來看，串流處理的好處相當多。首先，它大幅減少了使用者的等待時間。逐字元生成文字之後，使用者就可以開始解讀訊息。不需要等到完整訊息建構完成後才看到。這反過來又大幅提高了使用者互動性並讓延遲降到最低。

然而，這種技術也帶來了一些挑戰，其中之一是在串流過程中解析結果。理解並適當回應正在生成中的訊息可能非常複雜，尤其是當內容龐雜且包含大量細節的時候。

建立多個 LLM 生成結果

在某些情況下，你可能會發現從 LLM 生成多個回應非常好用，尤其是在產生動態內容（如社群媒體發文）的時候。與其提供單一筆訊息清單，不如提供由多個訊息清單所組成的清單。

```
Input:
```

```
    # 2x lists of messages, which is the same as [messages, messages]
    synchronous_llm_result = chat.batch([messages]*2)
    print(synchronous_llm_result)
```

```
Output:
    [AIMessage(content='''Sure, here's a lighthearted joke for you:\n\nWhy did
    the software engineer go broke?\n\nBecause he kept forgetting to Ctrl+ Z
    his expenses!'''),
    AIMessage(content='''Sure, here\'s a lighthearted joke for you:\n\nWhy do
    software engineers prefer dark mode?\n\nBecause it\'s easier on their
    "byte" vision!''')]
```

使用 `.batch()` 而不是 `.invoke()` 的好處在於 OpenAI 的 API 請求數量可以平行化處理。

對於 LangChain 中的任何可運行單元，你可以在 `batch` 函式中加入一個 `RunnableConfig` 參數，其中包含許多可配置的參數，包括 `max_concurrency`：

```
from langchain_core.runnables.config import RunnableConfig

# Create a RunnableConfig with the desired concurrency limit:
config = RunnableConfig(max_concurrency=5)

# Call the .batch() method with the inputs and config:
results = chat.batch([messages, messages], config=config)
```

在電腦科學領域中，*非同步*（*async*）*函式*是那些獨立於其他行程來執行的函式，好允許同時執行多個 API 請求而不需互相等待。在 LangChain 中，這些非同步函式能讓你同時發出多個 API 請求，而不用一個接一個。這對於更複雜的工作流程來說非常好用，可以減少使用者的整體延遲。

LangChain 中的大多數非同步函式都會以字母 `a` 作為前綴，例如 `.ainvoke()` 與 `.abatch()`。如果你想使用非同步 API 來提高任務品質效率的話，就應該好好運用這些函式。

LangChain 提示樣板

到目前為止，你是在 `ChatOpenAI` 物件中直接寫入字串。隨著 LLM 應用程式規模不斷擴大，靈活運用提示樣板就變得越來越重要了。

提示樣板對於生成可再現的 AI 語言模型提示非常好用，**樣板**指的是可以接受參數來建構語言模型文字提示的文字字串。

如果不使用提示樣板的話，你可能要改用 Python 的 `f-string` 格式化：

```
language = "Python"
prompt = f"What is the best way to learn coding in {language}?"
print(prompt) # What is the best way to learn coding in Python?
```

但為何不直接使用 `f-string` 來建立提示樣板呢？之所以建議使用 LangChain 提示樣板是因為可以輕鬆做到：

- 驗證你的提示輸入
- 藉由組合方式來結合多筆提示
- 自定義選擇器來將 k- 個樣本範例插入提示中
- 儲存與載入 *.yml* 和 *.json* 檔案中的提示
- 建立可執行其他程式碼或指令的自定義提示樣板

LangChain 表達式語言（LCEL）

「|」管線運算子是 LangChain 表達式語言（LCEL）的關鍵元件之一，它允許你在資料處理管線中將不同的元件或可運行單元鏈接起來。

在 LCEL 中，| 運算子類似於 Unix 管線運算子。它會把某個元件的輸出作為輸入來傳遞給鏈中的下一個元件。這讓你能夠輕鬆連接和組合各種不同的元件，建立起複雜的運算鏈：

```
chain = prompt | model
```

| 運算子可將提示和模型元件鏈接在一起。提示元件的輸出會作為輸入傳遞給模型元件。這種鏈接機制讓你得以從基本元件建構出更複雜的鏈，並實現不同處理階段之間的資料無縫流動。

順帶一提，鏈的順序很重要，因此就算以下的鏈在技術上是可行的：

```
bad_order_chain = model | prompt
```

但這麼做會在呼叫 `invoke` 函式後發生錯誤，因為 `model` 的回傳值與提示的預期輸入不相容。

讓我們使用提示樣板來建立一個商業名稱生成器，可以回傳五到七個相關的商業名稱：

```
from langchain_openai.chat_models import ChatOpenAI
from langchain_core.prompts import (SystemMessagePromptTemplate,
ChatPromptTemplate)

template = """
You are a creative consultant brainstorming names for businesses.
（你是一位創意顧問，正在為企業構思品牌名稱。）

You must follow the following principles:
（你必須遵守以下原則：）
{principles}

Please generate a numerical list of five catchy names for a start-up in the
{industry} industry that deals with {context}?
（請根據以下條件，為一間處於 {industry} 行業、專注於 {context} 的新創
公司，產出 5 個吸引人的名稱，並以編號清單呈現。）

Here is an example of the format:
1. Name1
2. Name2
3. Name3
4. Name4
5. Name5
"""
（以下是格式範例：
1. 名稱 1
2. 名稱 2
3. 名稱 3
4. 名稱 4
5. 名稱 5）

model = ChatOpenAI()
system_prompt = SystemMessagePromptTemplate.from_template(template)
chat_prompt = ChatPromptTemplate.from_messages([system_prompt])

chain = chat_prompt | model

result = chain.invoke({
    "industry": "medical",
    "context":'''creating AI solutions by automatically summarizing patient
    records''',
    "principles":'''1. Each name should be short and easy to
    remember. 2. Each name should be easy to pronounce.
    3. Each name should be unique and not already taken by another company.'''
})
    （"industry": "醫療 ",
    "context":'''建立 AI 解決方案，用於自動摘要病患紀錄。''',
```

```
        "principles":'''1. 每個名稱都應該簡短且好記。2. 每個名稱都應該好發音。
        3. 每個名稱都應該具有獨特性，且沒有其他公司使用。''')

    print(result.content)
```

Output:

```
    1. SummarAI
    2. MediSummar
    3. AutoDocs
    4. RecordAI
    5. SmartSummarize
```

上述程式碼中，首先匯入了 `ChatOpenAI`、`SystemMessagePromptTemplate` 和 `ChatPromptTemplate`。然後，在 `template` 中使用特定引導方式定義一個提示樣板，用於指示 LLM 來生成商業名稱。`ChatOpenAI()` 負責初始化聊天，而 `SystemMessagePromptTemplate.from_template(template)` 和 `ChatPromptTemplate.from_messages([system_prompt])` 則負責建立提示樣板。

可藉由將 `chat_prompt` 和模型鏈接起來建立一個後續可供呼叫的 LCEL `chain`。這會用 `invoke` 函式中的字典值來換掉提示中的 `{industries}`、`{context}` 與 `{principles}` 等占位符。

最後，存取 `result` 變數中的 `.content` 屬性將 LLM 的回應作為字串提取出來。

給予方向並指定格式

所謂精心設計的指示可能包括「你是一位創意顧問，正在為企業名稱腦力激盪」，和「請生成一份編號清單，包含五到七個極具吸引力的新創公司名稱」。這樣的線索能夠引導 LLM 確實執行你所要求的任務。

使用提示樣板與聊天模型

LangChain 提供 `PromptTemplate` 這種更為傳統的樣板，它需要 `input_variables` 和 `template` 兩個引數。

Input:

```
    from langchain_core.prompts import PromptTemplate
    from langchain.prompts.chat import SystemMessagePromptTemplate
    from langchain_openai.chat_models import ChatOpenAI
    prompt=PromptTemplate(
```

```
    template='''You are a helpful assistant that translates {input_language} to
    {output_language}.''',
    （template = '''你是一個樂於助人的助理，負責將 {input_language} 翻譯成 {output_
    language}。'''）
    input_variables=["input_language", "output_language"],
)
system_message_prompt = SystemMessagePromptTemplate(prompt=prompt)
chat = ChatOpenAI()
chat.invoke(system_message_prompt.format_messages(
input_language="English",output_language="French"))
```

```
Output:

    AIMessage(content="Vous êtes un assistant utile qui traduit l'anglais en
    français.", additional_kwargs={}, example=False)
```

輸出解析器

在第三章中，你使用正則表達式（regex）來從包含編號清單的文字中提取結構化資料，但 LangChain 可使用**輸出解析器**來自動完成這項運算。

輸出解析器是 LangChain 所提供的高階抽象化，用於從可由 LLM 的字串回應中解析結構化資料。目前可用的輸出解析器包括：

清單解析器

回傳一個由逗號分隔項目所組成的清單。

日期時間解析器

將 LLM 輸出解析為日期時間格式。

枚舉解析器

將字串解析為枚舉值。

自動修正解析器

包裝另一個輸出解析器，當該輸出解析器失敗時，則呼叫另一個 LLM 來修正任何錯誤。

Pydantic（*JSON*）解析器

將 LLM 回應解析為符合 Pydantic 規範的 JSON 輸出。

重試解析器

提供重試功能來解析前一個輸出解析器的失敗結果。

結構化輸出解析器

可用於希望回傳多個欄位的時候。

XML 解析器

將 LLM 回應解析為基於 XML 的格式。

後續你會發現，LangChain 輸出解析器有兩個非常重要的函式：

`.get_format_instructions()`

本函式為你的提示提供必要指示，藉此輸出可解析的結構化格式。

`.parse(llm_output: str)`

本函式負責將 LLM 回應解析為某個預先定義好的格式。

一般情況下，你會發現使用 Pydantic（JSON）解析器搭配 `ChatOpenAI()` 能提供最大的彈性。

Pydantic（JSON）解析器運用了 Pydantic（*https://oreil.ly/QIMih*）這套 Python 函式庫的長處。Pydantic 則是一套使用 Python 型態註解來驗證輸入資料的資料驗證函式庫。這代表 Pydantic 允許你為資料建立規範，並根據這些規範來自動驗證和解析輸入資料。

Input:

```
from langchain_core.prompts.chat import (
    ChatPromptTemplate,
    SystemMessagePromptTemplate,
)
from langchain_openai.chat_models import ChatOpenAI
from langchain.output_parsers import PydanticOutputParser
from pydantic.v1 import BaseModel, Field
from typing import List

temperature = 0.0

class BusinessName(BaseModel):
    name: str = Field(description="The name of the business")
    rating_score: float = Field(description='''The rating score of the
    business. 0 is the worst, 10 is the best.''')
```

第七章

淺談圖像生成擴散模型

本章將介紹幾款最受歡迎的 AI 圖像生成擴散模型。你將了解各款頂尖模型的優點與限制，以便日後可以根據手邊的工作內容明確地選擇出適合的模型。

於 2015 年推出的**擴散模型**為生成模型的一種，針對文字轉圖像的表現驚人。2022 年發表的 DALL-E 2（*https://oreil.ly/dalle2*）則代表擴散模型生成圖像品質的重大突破，而後開放原始碼的 Stable Diffusion（*https://oreil.ly/gjNJ_*），及社群大熱門 Midjourney（*https://oreil.ly/j51L0*）也迅速跟進，形成一個競爭激烈的領域。隨著 DALL-E3（*https://oreil.ly/dalle3*）的整合進 ChatGPT，文字與圖像生成之間的界線將越來越模糊。然而，進階使用者仍需要直接存取底層圖像的生成模型，以取得最理想結果。

擴散模型是在圖像上經由多個步驟添加隨機雜訊（*https://oreil.ly/OrAHA*），然後再預測如何藉由**降噪**（去除雜訊）來反轉擴散過程訓練而成。此方法來自物理學，是用來模擬粒子透過介質**擴散**（散開來）的過程。圖像描述決定了預測結果，所以如果生成的圖像不符，便會調整模型的神經網路權重，讓它可以預測出更符合描述的圖像。經過訓練後的模型可以將隨機雜訊轉換成符合文字提示描述的圖像。

圖 7-1 為 Binxu Wang 於「擴散生成模型的數學基礎」中示範的降噪過程（*https://oreil.ly/57szp*）。

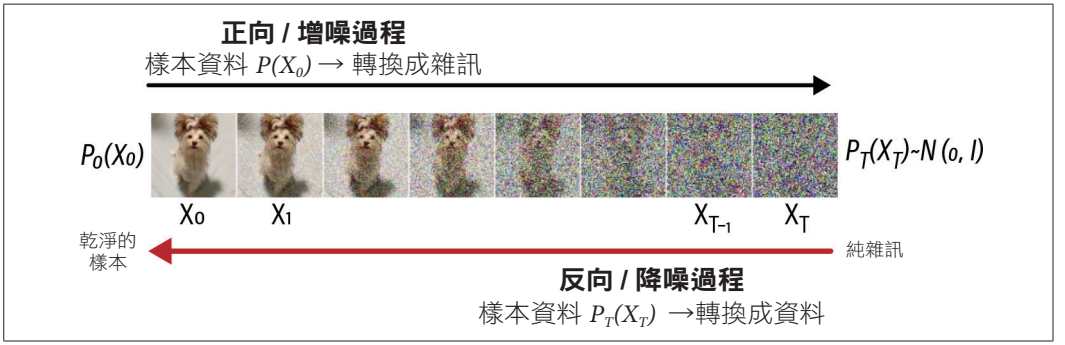


圖 7-1　擴散示意圖

這些模型是用從網路蒐集來的數十億張圖像（與對應的標註文字）之大型資料集訓練而成，因此可以模仿最受歡迎的風格或藝術家。這也同時一直是許多爭議的源頭，因為版權所有者尋求法律申訴（*https://oreil.ly/a4Fyp*），而模型建立者卻主張合理使用。

擴散模型絕非只是複製受版權保護圖像的「複雜拼貼工具」，它只有幾 GB 大，因此不可能包含所有訓練資料的副本。當學者試圖重現 Stable Diffusion 的 35 萬筆訓練資料時，只成功提取了 109 張（Carlini et al. 2023, *https://oreil.ly/SGn9B*）。

模型所做的更類似於人類藝術家在網路上瀏覽每張圖像，並學習定義各個主題和風格的模式。這些模式會編碼成參照潛在空間位置，即由模型生成所有圖像組合之地圖的向量表示，也就是一串數字。使用者輸入的文字提示首先會編碼成向量，接著擴散模型生成符合這些向量的圖像，然後將生成圖像解碼成像素給使用者。

圖 7-2 說明了編碼與解碼過程，資料來源：引用自 Ian Stenbit〈A Walk Through Latent Space with Stable Diffusion〉論文（*https://oreil.ly/qOpis*）.

這些向量也稱為**嵌入**，是每張圖像在模型地圖中的位置或地點，因此類似的圖像在潛在空間中的位置會比較近。潛在空間是連續的，可以在兩點之間移動（插值），並且仍然在過程中取得有效圖像。例如，從一張狗的圖像插值到一碗水果的圖像，中間過渡的圖像會有連貫性，以呈現出兩種概念的轉變過程。

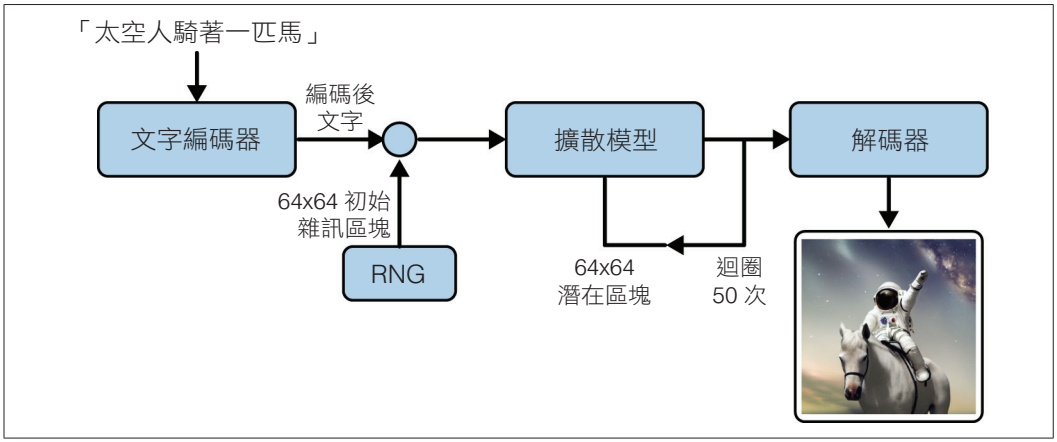


圖 7-2　編碼與解碼過程

圖 7-3 的圖表同樣來自 Ian Stenbit，呈現四張圖之間的過渡步驟（*https://oreil.ly/cjm8A*）：一隻狗（左上），一碗水果（右上），艾菲爾鐵塔（左下）以及摩天大樓（右下）。

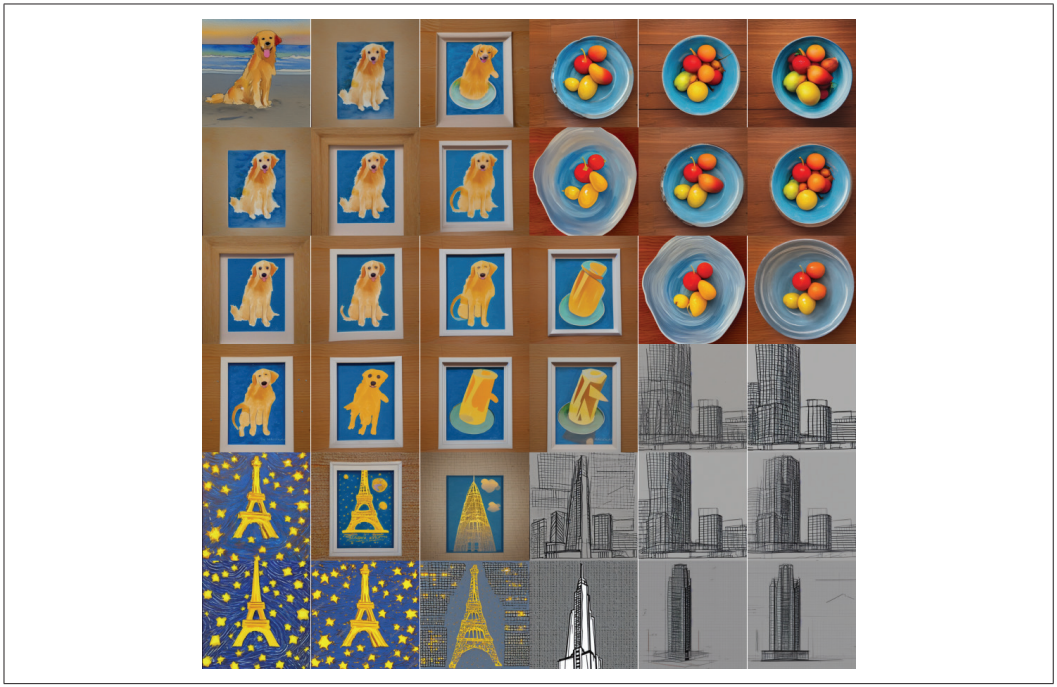


圖 7-3　漫步於潛在空間

在擴散模型的領域中，可將提示工程視為潛在空間中的導航，從所有可用圖像中找出符合你想像的那一張。有許多幫得上忙的正確文字組合，以塑造出所需圖像的技巧和最佳方法；還有一群活躍的 AI 藝術家與學者，致力於建立出一套輔助工具。每個模型和方法都有自己的眉角和特性，取決於它的架構、訓練方法以及用來訓練的資料。打造出最受歡迎的文字轉圖像擴散模型的三大主要組織，在商業模式和功能方面採取的都是截然不同的做法，也因此相較於 OpenAI 主導的 LLM 領域，擴散模型的選擇更加多元豐富。

OpenAI 的 DALL-E

2021 年 1 月，OpenAI 發表了文字轉圖像模型 DALL-E，名稱源自於超現實主義藝術家薩爾瓦多・達利和皮克斯動畫片中的機器人瓦力（WALL-E），以七個月前 OpenAI 發表的一款出色文字模型 GPT-3 改良版為基礎。DALL-E 是生成式 AI 的一項突破，展現大部分人認為電腦無法擁有的藝術能力。圖 7-4 為初代 DALL-E 的能力示範（*https://oreil.ly/dalle1*）。

圖 7-4　DALL-E 的能力示範

DALL-E 並非開放原始碼也沒有開放給公眾使用，但它吸引了許多學者和業餘玩家試圖複製該研究。這些模型中最受歡迎的是 2021 年 7 月發表的 DALL-E Mini（一年後應 OpenAI 要求改名為 Craiyon），儘管它在社群平台上有一群狂熱的支持者，品質還是比正版的 DALL-E 遜色許多。2022 年 4 月，OpenAI 發表了一篇論文並推出了 DALL-E 2（*https://oreil.ly/EqdtP*），因品質大幅提升而吸引了 100 萬人排隊等候。圖 7-5 為當時論文中激起了大眾無限想像空間，現已成為經典的騎馬太空人圖像。

圖 7-5　DALL-E 2 的圖像品質

出於 AI 倫理與安全之考量，直到 2022 年 9 月都只有候補名單上的人可以使用 DALL-E，且最初禁止生成含有人物以及一長串敏感單詞的圖像。研究學者發現，DALL-E 2 在一些諸如醫生等圖像提示中，加入黑人（black）或女性（female）等單字（*https://oreil.ly/ot4vw*），試圖以粗暴方式解決源自於資料集的偏見；因為網路上所能找到的醫生圖像，絕大多數都是白人男性。

OpenAI 團隊於 2022 年 8 月，在使用者介面中新增圖像修復與擴展功能，這又是一項重大躍進，吸引媒體與社群的關注。這些功能讓使用者可以僅生成圖像的特定部分，或透過在現有圖像周邊生成內容來**拉遠**。然而，使用者能控制的模型參數有限，也無法在自己的資料上微調。模型會在某些圖像上生成亂碼，且在人物的寫實描繪方面表現欠佳，容易生成變形或畸形的手腳、眼睛，如圖 7-6。

圖 7-6　變形的手和眼睛

2022 年 5 月的一篇論文介紹 Google 的 Imagen，並展示其令人驚豔的成果（Ho et al., 2022, *https://oreil.ly/sFaeW*），但考量到 AI 倫理與安全，該模型並未開放給一般大眾。Midjourney（2022 年 7 月）等競爭對手迅速行動，並成功吸收在社群媒體上看到 DALL-E 驚人範例，卻仍在候補名單上苦等的大批人群。2022 年 8 月，Stable Diffusion 發布開放原始碼，打破幾個月前仍看似無人能敵的 OpenAI 領先地位。雖然更進階的 DALL-E 3 模型（*https://oreil.ly/dalle3*）以 ChatGPT 的功能之一推出，有助於 OpenAI 搶回失去的地位，而 Google 也以 Gemini 1.5（*https://oreil.ly/XzQrU*）加入戰場；因此，戰況仍相當激烈，充滿許多變數。

Midjourney

2022 年 7 月，也就是 DALL-E 2 發表的三個月後，Midjourney 公開測試它的第三代模型。這實在是一個推出圖像生成模型的絕佳時機，因為早期使用者展現的 DALL-E 2 能力看起來跟變魔術一樣，但使用權限最初便受到限制。迫切的早期使用者紛紛湧向 Midjourney，而它著名的奇幻美學使它在遊戲與數位藝術圈中獲得了一群死忠鐵粉，這也表現在當時於數位藝術比賽中拔得頭籌、現已成為名畫的圖片上（*https://oreil.ly/ dqshh*），圖 7-7。

圖 7-7　Théâtre d'Opéra Spatial（太空歌劇院）

Midjourney 是第一個具備可行商業模式與商用授權的圖像生成模型之一，這讓它不再局限於實驗用途。它的訂閱模式受到許多早已習慣支付如 Adobe Photoshop 等其他軟體月租費的藝術家青睞；它也允許創作期間不須依生成圖片數量收費，尤其適合需要大量嘗試，才能找到高品質結果的創作初期階段。如果你是 Midjourney 的付費會員，還可以保有任何生成圖像的權利，不像 DALL-E，著作權一律歸 OpenAI 所有。

Midjourney 的獨特之處在於其強烈的社群導向。要使用這個工具，必須先登入 Discord 伺服器（*https://oreil.ly/JKZzD*）（圖 7-8）並在公開頻道或私訊中提交文字提示。由於所有圖像生成皆會自動分享在公開頻道中，且只有最貴的方案才提供私密模式（*https://oreil.ly/OV46r*），因此透過 Midjourney 生成的絕大多數圖像都可供他人參考。這讓使用者彼此能夠快速複製和迭代，讓新手能輕鬆從其他人身上學習。早在 2022 年 7 月，Midjourney 的 Discord 社群人數就已來到 100 萬人（圖 7-8），一年後，成員數量超過了 1300 萬人。

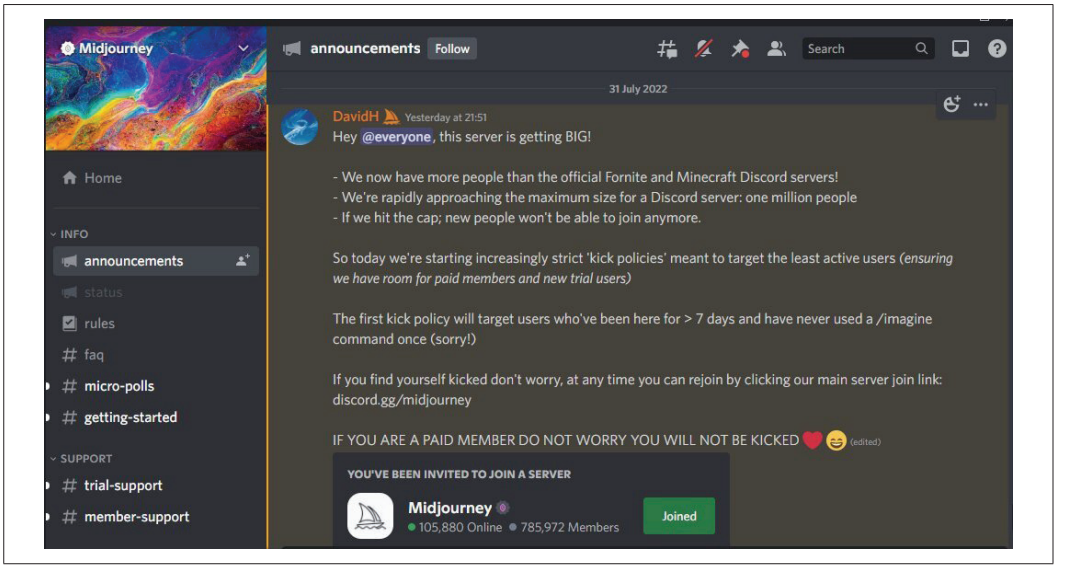


圖 7-8　Midjourney 的 Discord 伺服器，2022 年 7 月

找到心儀的圖片後，可以點選將其**升級放大**（提高解析度）。許多人猜測這個步驟是用於強化學習的訓練資料，類似於人類意見回饋強化學習（RLHF, *https://oreil.ly/3ISZk*），也是 ChatGPT 成功的關鍵。此外，團隊也會定期請使用者為新模型生成的圖片評分以改進性能。Midjourney 於 2022 年 11 月發表第四代模型，隨後於 2023 年 5 月釋出第五代，緊接著在同年 12 月推出第六代。品質明顯提升，如圖 7-6 中的劣質手眼問題大部分皆已消失，且模型所能掌握的風格也更加豐富多元，如圖 7-9。

```
Input:

    a group of best friends women eating salads and laughing
    while high fiving in a coffee shop, cinematic lighting
    （一群交情很好的女性朋友在咖啡廳裡一邊吃沙拉、一邊大笑
    並互相擊掌，場景充滿電影感的燈光效果。）
```

輸出如圖 7-9。

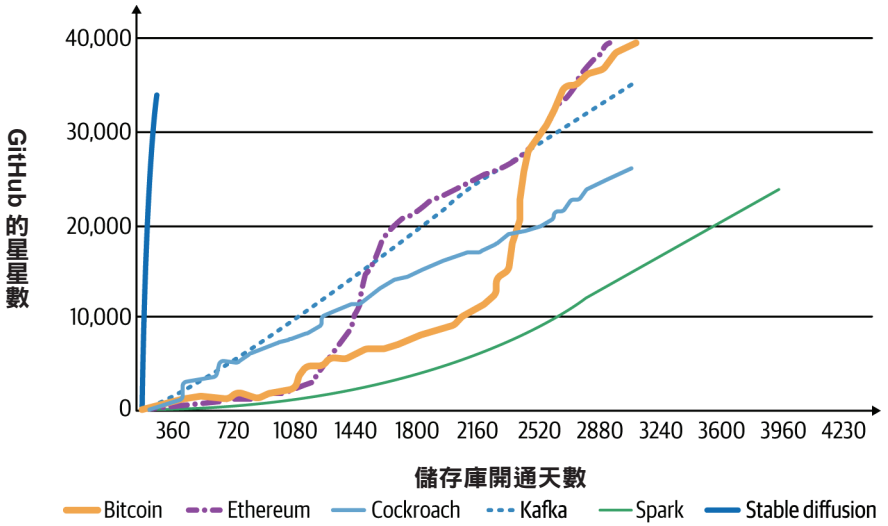
圖 7-9　一群歡笑著享用沙拉的女性

最令人驚奇的是，截至 2023 年 3 月，Midjourney 的團隊一直都不大，只有 11 名員工（*https://oreil.ly/YrmA_*）。Midjourney 的創辦人 David Holz 之前為 Leap Motion 硬體新創公司的成員，他在一次訪談中，證實公司在 2022 年 8 月便已開始獲利（*https://oreil.ly/ jeFYV*）。更神奇的是，儘管不像 OpenAI 那樣享有數十億美元的資金，團隊仍然在 DALL-E 的基礎上打造出卓越性能，包括從圖像移除某些概念的負面提示、增加其他概念普及率的加權詞，以及從上傳圖像負面推導出提示的描述特徵。不過，模型不提供 API，目前只能透過 Discord 使用，這可能會是它成為主流模型的阻礙。

Stable Diffusion

當 DALL-E 2 的候補名單不斷變長的時候，慕尼黑大學的 CompVis 以及應用研究公司 Runway ML 的研究團隊，收到 Stability AI 提供用以訓練 Stable Diffusion 的算力資源。該模型在 2022 年 8 月以開放原始碼的形式發表時震驚了生成式 AI 界，因為其結果媲美

DALL-E 2 和 Midjourney，但又可以在自己的電腦上免費使用（前提是你的電腦具備 8GB VRAM 的中等性能 GPU）。Stable Diffusion 是 GitHub 史上星星累積速度最快的軟體之一（*https://oreil.ly/pwPGX*），發表後的 90 天內便獲得了 33,600 顆（圖 7-10）。



主要開放原始碼的基礎技術在 GitHub 上的星星數成長曲線。Stable Diffusion 在最初的 90 天便累積了 33,600 顆，這是其他專案需要花上數年或數十年才能達到的數量。

圖 7-10　GitHub 開發者採用 Stable Diffusion 的情況

將模型以開放原始碼釋出的決定引發爭議，並激起對 AI 倫理與安全性的擔憂。確實，許多初期的使用案例都是一些 AI 色情內容，這從在 Civitai（*https://civitai.com*）等平台上分享的各種工作場合不宜（NSFW）模型中就可見一斑。然而，讓愛好者和玩家能夠修改跟擴增模型，以及根據自己的資料微調這點，也讓模型的功能得以迅速進化與改善。將模型的所有參數公開給使用者這項決定，例如無分類器的引導（Classifier Free Guidance），提示的遵守程度、降噪（推論過程中要對基礎圖像添加多少雜訊之後再讓模型消除）、以及種子（Seed，開始降噪的隨機雜訊）等，都催生出更多創意與創新兼備的藝術。

開放原始碼的便利與可靠也吸引了不少小公司以 Station Diffusion 為基礎來開發產品，像是 Pieter Level 的 PhotoAI（*https://photoai.com*）和 InteriorAI（*http://interiorai.com*），這兩項產品的月收入就超過了 10 萬美金；以及 Danny Postma 的 Headshot Pro（*https://www.headshotpro.com*）。除了跟 DALL-E 一樣有修復與擴展功能之外，開源貢獻還緊跟 Midjourney 的功能，例如負面提示、加權詞，以及從圖像反推出提示的能力。除此之外，

進階功能如配合圖片中姿勢或構圖的 ControlNet，和點擊圖像中的物體或區域，來生成修補圖像用遮罩的 Segment Anything，也很快就在 2023 年 4 月，發表為 Stable Diffusion 的擴充工具，常見的做法是從 AUTOMATIC1111 的網頁介面進入（*https://oreil.ly/0inw3*），見圖 7-11。

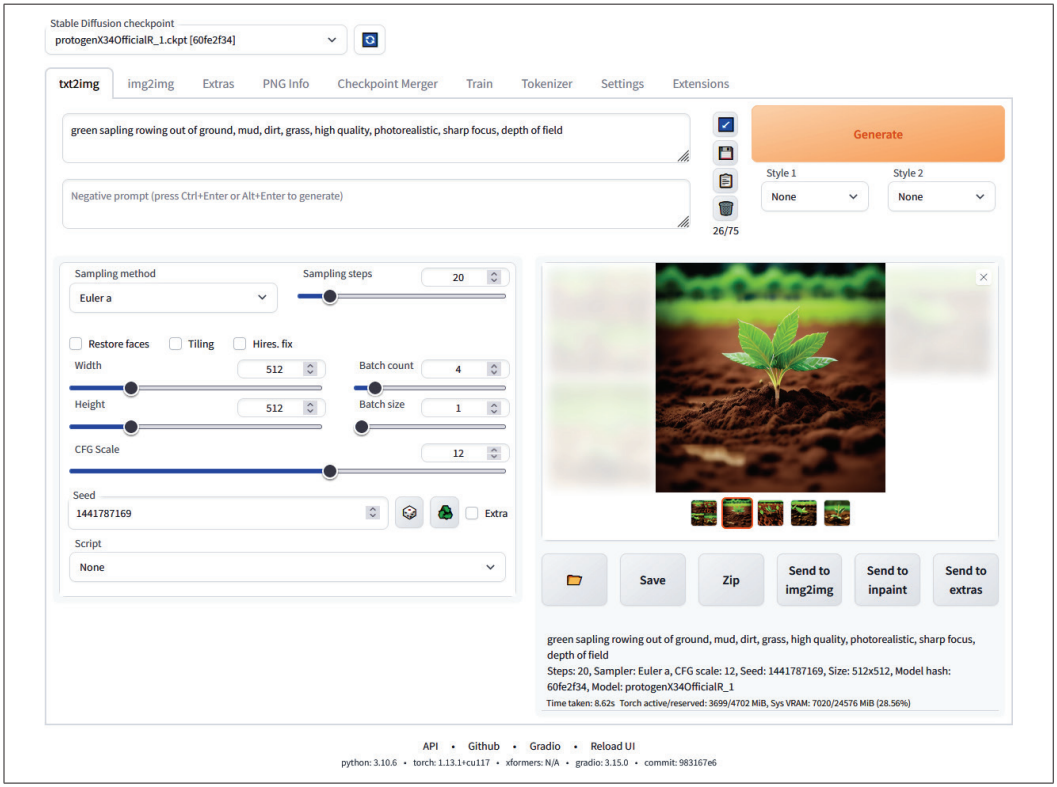


圖 7-11　Stable Diffusion 在 AUTOMATIC1111 網站上的介面

Stable Diffusion 的 1.5 版發表於 2022 年 10 月，至今仍在使用中。因此它會是本書第十章關於進階圖像生成的 ControlNet 相關範例基礎。Stable Diffusion 的權重發布於 Hugging Face 上，讓這個世代的 AI 工程師，都認識這個開放原始碼的 AI 模型平台。第二代的 Stable Diffusion 在一個月後於 2022 年 11 月推出，用原本的 LAION-5B 資料集中，更具美感的子集，一個研究用的大型圖文配對資料集所訓練（*https://oreil.ly/K5vX2*），並過濾掉 NSFW（工作場合不宜）圖片。但 Stable Diffusion 的高階使用者抱怨這類審查機制以及模型效能降低，推測是因為訓練集中的 NSFW 圖片，對生成逼真的人體構造仍有其必要性（*https://oreil.ly/2mgh5*）。

Stability AI 募得超過 100 萬美金（*https://oreil.ly/BT-k5*），並持續開發新模型，包括 DeepFloyd（*https://oreil.ly/UCQ3I*），這是一款更擅長在圖像中生成真實文字的模型，也是其他模型的通病；以及目前最受歡迎的 Stable Diffusion XL 1.0（*https://oreil.ly/gcT4t*，簡稱 SDXL）。SDXL 克服了社群對 2.0 版的審查疑慮，不僅因為限制較少，更因為這款更強大模型所展現出的絕佳成果，它擁有 66 億個參數，而 1.5 版只有 9.8 億個。

Google Gemini

長久以來，Google 便誓言要以他們的 Imagen 成為此領域的競爭者（*https://oreil.ly/K8oWv*，而非公開的 *https://oreil.ly/341QB*），而確實，一群前 Google 員工建立了一款相當有潛力的新款圖像模型，Ideogram（*https://ideogram.ai*），發表於 2023 年 8 月。Google 於 2023 年 12 月終於以 Gemini 進軍圖像生成領域，但很快就因其推廣多樣性的拙劣做法飽受抨擊（*https://oreil.ly/u-Glg*）。Google 內部的政治環境是否會持續成為他們充分利用豐富資源的阻力，這件事仍有待觀察。

文字轉影像

隨著 Stable Diffusion 社群致力於擴展模型的能力（*https://oreil.ly/l7KHB*）以逐幀生成一致的圖像，多數對圖像領域的關注很有可能會轉向**文字轉影像**、**圖片轉影像**甚至**影像轉影像**，包括 AnimateDiff 等潛力十足的開放原始碼專案（*https://oreil.ly/CsJgT*）。此外，作為 Stable Diffusion 的共同創辦人之一，RunwayML 已是引領文字轉影像的先驅，透過他們的 Gen-2 模型開始獲得一些可用的結果（*https://oreil.ly/vS0mA*）。Stable Video Diffusion（*https://oreil.ly/UuApM*）發表於 2023 年 11 月，能夠將文字轉換成短影片，或將既有圖片改成動畫，而 Stable Diffusion Turbo（*https://oreil.ly/uMAkh*）幾乎能夠即時生成圖像。2024 年 2 月發表的 Sora（*https://oreil.ly/sora*）則顯示 OpenAI 在此領域並非坐以待斃。雖然本書不會明確地涵蓋文字轉影像的提示技術，但你所學到關於圖像生成的提示技術都可以直接應用在影像上。

模型比較

隨著對 AI 圖像生成的需求增加和競爭加劇，新的參賽者不斷浮上檯面，而主流大廠也會變得更多元。在本書的工作流程中，你應該已經發現我們會基於不同理由而使用不同模型。DALL-E 3 擅長構圖，且整合在 ChatGPT 中這點也很方便；Midjourney 無論是奇幻還是寫實風，美感依舊最為出色；Stable Diffusion 身為開放原始碼的特點，讓它更加靈活並

具擴充性，也是大多數 AI 企業產品開發的基礎。每個模型都逐漸演變出獨特的風格和能力，這一點可以透過比較各個模型用相同提示的生成結果看出來，如圖 7-12。

```
Input:

    a corgi on top of the Brandenburg Gate
    （在布蘭登堡門上的柯基犬）
```

輸出如圖 7-12。

圖 7-12　在布蘭登堡門上的柯基犬

總結

本章向你介紹了用於 AI 圖像生成的擴散模型。這些模型，例如 DALL-E、Stable Diffusion 和 Midjourney 都是透過隨機雜訊和降噪技術來根據指定文字描述生成圖像。這些模型是根據大型資料集訓練而成，並能夠複製出多種藝術風格。然而，關於版權的紛爭依然存在。你也認識了提示工程原則在潛在空間中導航到所需圖像之指引角色，因此也能應用在圖像生成上。

本章也探討了 OpenAI、Stability AI 和 Midjourney 等公司組織在開發文字轉圖像模型時所採用的不同做法。OpenAI 的 DALL-E 因為它的美術能力而深受喜愛，但使用權限有限且模型複製品的品質較差。另一方面，Midjourney 利用了市場對 DALL-E 替代品的需求，以第 3 代與第 4 代擄獲一群死忠的粉絲。它採取訂閱制的付費方式，並著重於社群發展。

而 Stable Diffusion 因其媲美 DALL-E 和 Midjourney 的生成結果而受到關注，但又擁有開放原始碼以及在個人電腦上便可免費使用的優勢。透過本章，你也認識到 AI 圖像生成的歷史以及 OpenAI、Midjourney 和 Stable Diffusion 等組織各自取得的進展。