## 可用於機器學習系統設計面試的一個框架

許多工程師認為，ML 演算法（例如自動迴歸 Transformer 或 Diffusion 擴散模型）就是 ML 系統的全部。但是，如果要打造、部署一個生成式 AI 系統，所要做的事絕不是只有訓練模型而已。這類的系統通常都很複雜，其中所用到的各種組件，包括用來預處理 / 處理大型資料集的各種資料流程、用來評估輸出品質和安全性的各種評估機制、可針對不同規模送出 AI 生成內容的基礎設施，以及用來確保系統表現長期一致性的監控機制。

在 ML（尤其是生成式 AI）的系統設計面試過程中，你經常都會遇到許多開放式問題。舉例來說，你有可能會被要求設計出一個客服聊天機器人，或是專為創意設計者設計出一個 AI 圖片編輯工具。像這樣的問題，根本就沒有唯一的「正確」答案。面試官真正感興趣的，其實是觀察你如何處理這些複雜的問題、看看你對生成式 AI 各種概念的理解程度、觀察你設計整個系統的過程，以及你在設計上做出不同選擇背後的理由。

如果想順利通過生成式 AI 系統設計面試的考驗，很重要的就是要有一套可依循的結構化框架。雜亂無章的回答，只會擾亂你思考的過程，降低整件事清楚的程度。本書會介紹一個框架，引導你通過生成式 AI 系統設計面試。這個框架包含了以下幾個關鍵的步驟：

1. 把各種需求明確化
2. 用框架把問題轉化成機器學習任務
3. 資料的準備
4. 模型的開發
5. 評估
6. ML 系統的整體設計
7. 部署和監控

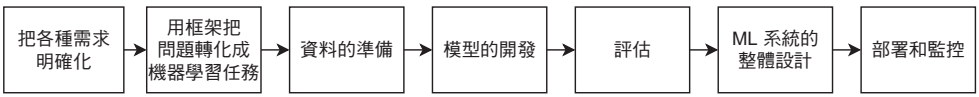


圖 1.6：ML 系統設計的幾個步驟

接下來我們會深入研究每個步驟，並檢視一下生成式 AI 系統設計的一些關鍵考慮因素和討論要點。

## 把各種需求明確化

當你想要開發出一個 ML 系統來解決特定的任務時，一開始通常只有很少量的資訊。同樣的，在面試過程中，機器學習系統設計的問題通常都很模糊，只提供很少的細節。舉例來說，面試官可能會要求你「設計出一個圖片生成系統」。你的第一步，就是要提出一些可以讓需求更明確的問題。但是，你究竟應該問哪些問題呢？

你所提出的問題，應該要有助於理解整個問題的範圍，以及系統所要實現的具體目標。這些問題大致上可以分成兩種類型：

- 功能性需求
- 非功能性需求

### 功能性需求

功能性需求所要說明的，就是這個系統應該做什麼——也就是這個系統的核心能力。舉例來說，「可根據使用者的提示來生成圖片，而且可自定義圖片的風格」，這就是文字轉圖片系統的功能性需求。在設計生成式 AI 系統時，功能性需求極為重要，因為這些需求可以塑造出系統的高階架構。這些需求會引導系統各組件與功能的開發，讓這個系統可以滿足使用者的需求。

### 非功能性需求

非功能性需求比較關注的是系統運作的情況，而不是系統會做什麼事。這類的需求包括系統的延遲、吞吐量等等的效能表現指標，以及系統公平性、安全性與可擴展性之類的各種考量。舉例來說，圖片生成系統的非功能性需求，可能就會去定義圖片生成過程可接受的速度，以及必須滿足的圖片品質標準。雖然這些需求在生成式 AI 系統設計中通常都算是比較進階的主題，而且這些需求或許並不會明顯去改變初始的架構，但很重要的是最好能早一點去識別出這些需求並建立足夠的理解，因為這些需求通常會在設計比較後期的階段（尤其是效能調整和系統改進階段）產生很大的影響。

下面幾個問題或許可以協助你踏出第一步：

- **商業上的目標**：這個系統主要的目標是什麼呢？它有什麼具體的用途？舉例來說，在設計圖片說明（image captioning）系統時，一定要先知道這個系統是否會被用於電子商務平台，用來生成詳細的產品說明，還是要針對社群媒體上的照片，給出簡短的圖片說明建議。
- **系統的功能**：系統的功能可能會影響機器學習的設計方式；這個系統應該支援哪些功能呢？舉例來說，在設計圖片生成（image generation）系統時，很重要的就是要知道使用者能否針對所生成的圖片，提供各種回饋意見或是進行評價，因為這些互動的資訊，也可以回頭用來強化我們的模型。同樣的，在設計一個 LLM 大語言模型時，很重要的就是先瞭解一下，我們的系統究竟應該支援哪些語言。
- **資料**：有哪些資料來源呢？資料集有多大？資料都是有標記過的嗎？這幾個問題都非常重要，因為資料的品質和數量，很可能會影響系統設計的方式。
- **限制**：有哪些可運用的運算資源呢？這個系統會在雲端運行，還是專為設備端運行而設計？
- **系統規模**：預計會有多少使用者來使用這個系統呢？需要生成多少張圖片？需求的成長率，預計將會如何呢？釐清這幾個問題是很重要的，因為只給一小群使用者使用的生成圖片系統，所需要考慮的系統可擴展性，與那種針對好幾百萬使用者提供服務的系統肯定是不同的。
- **效能**：內容生成的速度應該要有多快呢？是否需要即時生成？內容的品質與生成的速度，哪一個比較重要呢？

這份列表並不算很全面，不過它還是可以作為一個很好的起點。至於其他的主題，例如個人隱私、倫理道德考量、資料安全等等，其實也都是很重要的。

這個步驟結束之後，你和面試官針對系統的範圍和需求，應該已經達成一致的看法了。一開始就能讓各種細節更加明確化，通常是一個很好的做法，因為這樣你才能確保自己可以滿足面試官的期待。

## 用框架把問題轉化成機器學習任務

如果面試官要求你設計出一個功能，能夠針對使用者的許多 Email 自動做出總結，這樣你就有了一個需要解決的問題了。但是，你並不能直接要求 AI 幫你總結 Email。實際上你必須先把這個問題化為一個框架，然後才能用 AI 技術來解決問題。先用框架把問題轉化成機器學習任務，可說是設計 ML 系統其中一個非常關鍵的步驟，因為這樣的做法會直接影響整個設計的其他部分。

在解決問題時，首先要判斷的就是——真的需要用到 ML 嗎？不過，既然我們要設計的是生成式 AI 系統，通常你可以假設過程中一定會用到 ML，因為 ML 正是開發這類系統最主要的一種工具。

如果要用框架把問題轉化成 ML 機器學習任務，下面兩個步驟還蠻有用的：

- 指定系統的輸入與輸出
- 選擇合適的 ML 做法

### 指定系統的輸入與輸出

為了把問題轉化一個框架，首先就是要定義系統的輸入與輸出。這部分牽涉到的就是要識別出輸入資料的模態（modality；例如文字、圖片、聲音、影片等等），還有預期將會得到什麼樣的輸出。舉例來說，聊天機器人系統的輸入就是使用者的查詢文字，輸出則是系統的回應。

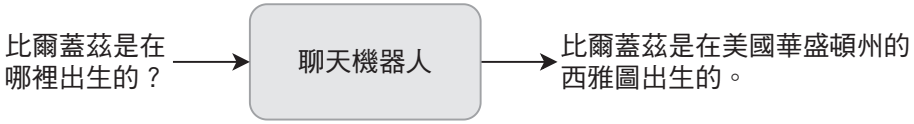


圖 1.7：聊天機器人的輸入與輸出

# 2

# Gmail 智慧撰寫

Gmail 的智慧撰寫（Smart Compose）功能 [1] 可以在使用者撰寫 Email 時，針對接下來的幾個單詞給出一些建議。本章打算探索這個功能，並檢視一下大多數生成式系統都會用到的 Transformer 架構。

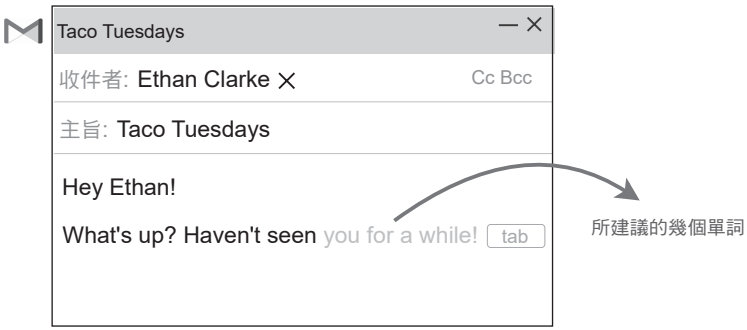


圖 2.1：Gmail 的智慧撰寫功能

## 把各種需求明確化

下面就是應試者與面試官之間的一段典型互動：

**應試者**：不同使用者可能會有不同的寫作風格。這個系統需不需要做出個人化的建議？

**面試官**：為了簡單起見，我們先不用製作出個人化的功能。

**應試者**：唯有當系統對自己的預測很有信心時，才應該去建議接下來的幾個單詞嗎？

**面試官**：是的。

**應試者**：Email 資料集必須足夠大，才能訓練好模型。我們的資料量大概有多少呢？

**面試官**：假設我們的資料集包含了大約十億封 Email。

**應試者**：在提出建議時，可能會運用到不同的資料。舉例來說，使用者過去的 Email，或是當前這封 Email 的主旨，都可以拿來加以運用。為了簡單起見，我可以只運用當下這封 Email 的內文來作為前後文嗎？

**面試官**：很好的想法。不過實際上我們會用到的，並不只有使用者在當下這封 Email 裡輸入的內容。我們可以先從 Email 的內文開始著手。如果有時間的話，也可以稍微擴展一下前後文的範圍，把其他相關的資訊包含進來。

**應試者**：這個系統應該支援哪些語言？

**面試官**：一開始只考慮英語就行了。

**應試者**：我們需要確保這個系統不會有偏見嗎？

**面試官**：對這個系統來說，這是個很重要的需求。系統在提供建議時，不應該做出帶有偏見的假設。

**應試者**：Gmail 有多少活躍的使用者？這個功能需不需要考慮計算的成本？

**面試官**：Gmail 大約有 18 億使用者，單一使用者一天最多可能會發送 500 封 Email。我們確實會在意計算的成本，不過這裡可以先把焦點聚焦於系統的開發。未來系統進行迭代更新時，我們可以再進一步優化效率。

**應試者**：這個系統應該做出即時的建議嗎？

**面試官**：是的。預期的延遲應該要做到難以察覺的程度；大約 100 毫秒內就可以接受了。

## 用框架把問題轉化成機器學習任務

我們會在本節用一個框架，把智慧撰寫功能轉化成機器學習任務。首先我們要瞭解系統的輸入與輸出，這樣才能挑選出合適的 ML 做法，讓它能透過學習來完成任務。

### 指定系統的輸入與輸出

模型的輸入就是使用者所輸入的一系列單詞。輸出則是這一系列單詞後續的文字。這個模型所生成的東西，就是使用者接下來可能會輸入的一些單詞。


圖 2.2：智慧撰寫系統的輸入與輸出

### 選擇合適的 ML 做法

這個智慧撰寫系統會生成文字內容，因此我們把它歸類為文字生成任務。有許多的 ML 架構都是設計成用來處理序列型資料，而文字生成所需要的正是這類的架構。其中比較流行的兩種架構，就是 RNN（循環神經網路）[2] 和 Transformer [3]。

Transformer 比 RNN 多出了幾個優勢，下面就是它的兩個主要優點：

- **平行化**：在 RNN 的架構下，一個步驟的計算所要花費的時間，會被累計到下一個步驟，從而形成一整串與時間相關的操作。但 Transformer 的自注意力機制，可以讓所有的輸入 Token 同時進行處理。
- **更擅長處理長序列**：Transformer 採用了自注意力機制，因此可以把注意力集中在一長串序列裡的任何一部分，而不用管距離有多遠。相較之下，RNN 因為它本身的循序結構，加上梯度消失的問題，因此很難處理長距離的依賴關係。

# 總結