前言

ChatGPT 一問世，我和同事們都陷入了迷惘，並非是驚訝於模型的規模或能力，因為 AI 社群早在過去十幾年來，就已經知道擴大模型規模可以提升效能。2012 年 AlexNet 的作者們在其劃時代的論文（*https://oreil.ly/XG3mv*）中指出：實驗證明只要有更快的 GPU 和更大的數據集，成果就會大有進展。[1,2]

而真正讓我嚇到的是，這種能力的提升解鎖了無數的應用。我原本以為模型品質指標的稍微提升只會帶來應用的合理增長，卻沒想到它引爆了各種新奇的發展。

新的 AI 能力不僅增加了 AI 應用的需求，也降低了開發者的進入門檻。建構 AI 應用變得極其容易，甚至無須編寫程式就能完成應用的開發。這種轉變，將 AI 從一門專業學科轉化成所有人都能運用的強大開發工具。

儘管目前 AI 看似新穎，但其實它的基礎技術存在已久，早在 1950 年代就有語言模型的相關論文了。檢索增強生成（RAG）應用建立在檢索技術之上，在 RAG 一詞出現以前，這種技術早已在搜尋和推薦系統中行之有年了。系統性實驗、嚴謹評估、為更快、更便宜的模型持續最佳化等，都是部署傳統機器學習應用的最佳實例，也會是基於基礎模型開發應用的最佳範例。

因為熟悉又好用，許多 AI 工程技術會讓人們以為 AI 工程了無新意。雖然許多建構 AI 應用的原則相同，但 AI 模型的規模和能力的提升卻帶來新的機會與挑戰，且同時也需要新的解決方案。

1 AlexNet 論文作者之一的 Ilya Sutskever 是 OpenAI 的聯合創辦人，他以 GPT 模型實現此想法。

2 我在 2017 年（*https://x.com/chipro/status/937384141791698944*）的小專案以語言模型評估翻譯品質，結論是還需要「更好的語言模型」。

本書涵蓋調適基礎模型、從原始數據到最終產品解決實際問題的流程，其中包含其他工程領域的驗證技術與基礎模型衍生的技術。

會想寫這本書起因於我想從我參與的專案、讀過的論文和採訪過的人學習，而我也確實學到很多。本書的寫作過程用到包括幾個重要 AI 實驗室（OpenAI、Google、Anthropic 等）的研究者、框架開發者（輝達、Meta、Hugging Face、Anyscale、LangChain、LlamaIndex 等）、不同規模 AI / 數據公司的執行者和負責人、產品經理、社群研究者，以及獨立應用開發者等超過上百次的對話和訪談記錄，請惠予參考「致謝」。

特別是從早期就一起測試我的想法的讀者們，他們引導我產生不同的觀點，進而讓我接觸到新的問題和方法，這部分我也學到不少。我在部落格分享一部分本書內容（*https://buyenchip.com/blog/*），收到的數千條評論帶給我很多新視角或想法的確認。

現在這本書到了您的手裡，我希望我還能從您的獨特經驗和觀點繼續學習。請隨時透過 X（*https://x.com/chipro*）、*LinkedIn*（*https://www.linkedin.com/chiphuyen*）或電子郵件 *hi@huyenchip.com* 與我分享您對本書的任何回饋。

本書內容

本書提供涵蓋用以適配基礎模型的大型語言模型（LLMs）和大型多模態模型（LMMs）框架的各種特定應用。

建構應用的方法很多，本書除了概述各種常見解法，也提出了一系列值得思考的問題，幫助你評估最適合自身需求的解決方案。本書能幫助您解答的問題包括：

- 我應該開發這個 AI 應用嗎？
- 我怎麼評估應用？可以用 AI 評估 AI 產品嗎？
- 幻覺是怎麼產生的？如何檢測並減少幻覺？
- 提示工程的最佳範例是什麼？
- RAG 原理為何？運行 RAG 的策略有哪些？
- 什麼是代理？如何建構和評估代理？
- 怎麼判斷在什麼狀況下，才需要微調模型？
- 多少數據量才夠？如何驗證數據品質夠好？

- 如何讓模型更快、更便宜、更安全？
- 如何創建可持續改善應用的回饋迴路？

本書還將帶您探索龐大的 AI 領域：模型類型、評估基準，以及看似無窮無盡的使用案例和應用模式。

本書大部分內容會以我親自參與過的案例研究進行說明，並輔以大量參考資料的佐證以及不同背景專家的深度檢視。儘管本書花了兩年時間撰寫，但它有我過去十年在語言模型和機器學習系統的實務經驗。

如同我之前在 O'Reilly 出版過《設計機器學習系統》（DMLS）一書，本書不會介紹特定的工具或 API，而是專注於 AI 工程的基礎，由於工具很快就會被淘汰，但基礎知識歷久彌新。[3]

和《設計機器學習系統》（DMLS）一起讀《AI 工程》（AIE）

AIE 可以和 DMLS 配合閱讀。DMLS 聚焦在包含表格資料標註、特徵工程和模型訓練的傳統機器模型上建構應用，而 AIE 則專注在包含提示工程、上下文構建和參數效率微調的基礎模型上建構應用。兩本書本身都是完整且模組化的，也適合分別閱讀。

因為基礎模型就是 ML 模型，所以某些概念兩本書都會討論。如果某個主題和 AIE 相關但在 DMLS 一書已深入討論過，那我就不會再重複放入本書，但我仍會註明相關的參考資源。

要注意的是，許多 DMLS 的主題並不會出現在 AIE 一書中，反之亦然。本書第一章包含傳統 ML 工程和 AI 工程的差別，實際的系統都會涵蓋傳統 ML 模型和基礎模型，所以還是有必要熟悉它們。

3 2017 年在教授如何使用 TensorFlow 的課程給了我一個痛苦的教訓：工具和教程過時的速度太快了。

要判斷某件事物是否會被淘汰其實很難，通常我會依據三個標準。首先我會判斷是否是受到 AI 運作的基本限制才有的問題，否則有更好的模型問題自然就會消失。而如果是基礎問題，我會分析其面對的挑戰並提出解決個別挑戰的方案。我喜歡從簡單的問題開始解決，然後逐步導入稍微複雜的方案來應對更大的挑戰。

其次，我會大量請教比我聰明的研究人員和工程師，聽聽他們認為最關鍵的問題和解法。

偶爾我也會參考 Lindy 效應（*https://en.wikipedia.org/wiki/Lindy_effect*），此效應認為一項技術的未來壽命與其當前年齡成正比。因此如果某事物已經存在了一段時間，我就假設它還會繼續存在一段更長的時間。

然而在本書中，我有時也會加入能對某些應用開發者提供即時幫助或有趣的解決方法。

本書不是理論

雖然書中提到特定工具並使用虛擬程式碼來說明某些概念，但書並非教程導向、不是要教您如何使用工具。相反地，本書提供選擇工具的框架，包含許多權衡不同解決方案的討論，以及您在評估解法時會碰到的問題。要用某個工具的話，您可以輕易地在網路上找到相關教程，AI 聊天機器人也可幫助我們入門很多熱門工具。

本書不是 ML 理論書籍，它不會解釋什麼是神經網路，也不會教你如何從頭開始建構和訓練模型。本書會解釋許多重要的理論概念，但更多是聚焦於協助您成功建構解決 AI 應用實際問題的實務書籍。

雖然沒有 ML 專業也能建構基於基礎模型的應用，但有 ML 和統計學的基本理解可以幫助我們建構更好的應用，進而減少不必要的麻煩。您無須任何 ML 背景也能看懂本書，但掌握以下概念對您開發 AI 應用時會更有幫助：

- 如取樣、確定性、分佈等機率概念
- 如監督、自監督、對數概似（likelihood）、梯度下降、反向傳播、損失函數和超參數調整等 ML 概念
- 包括前饋、遞迴和 Transformer 等各種神經網路架構
- 如準確性、F1、精確度、召回率、餘弦相似度和交叉熵等指標

如果您不懂這些，那也別擔心，本書會提供簡要說明或參考指南幫您跟上。

Transformer 架構

要理解 Transformer，我們先看看它解決了什麼問題。Transformer 架構是在 seq2seq（序列到序列）（*https://arxiv.org/abs/1409.3215*）架構成功後崛起的。2014 年推出的 seq2seq，大幅改進了當時極為困難的機器翻譯和文本總結任務。2016 年，Google 將 seq2seq 應用於 Google Translate（*https://oreil.ly/fb1aR*），據稱這次更新帶來了「迄今為止機器翻譯品質的最大提升」，並引發對 seq2seq 的廣泛關注，也使其成為處理文本序列任務的首選架構。

seq2seq 包含處理輸入的編碼器和生成輸出的解碼器，因其輸入和輸出都是標記序列而得名。seq2seq 使用 RNN（遞迴神經網路）作為編碼器和解碼器的核心。在最基本的形式中，編碼器依序處理輸入標記，生成表示輸入的最終隱藏狀態；解碼器則根據這個最終隱藏狀態和之前生成的標記，按順序生成輸出標記。圖 2-4 的上半部分展示了 seq2seq 架構的示意圖。

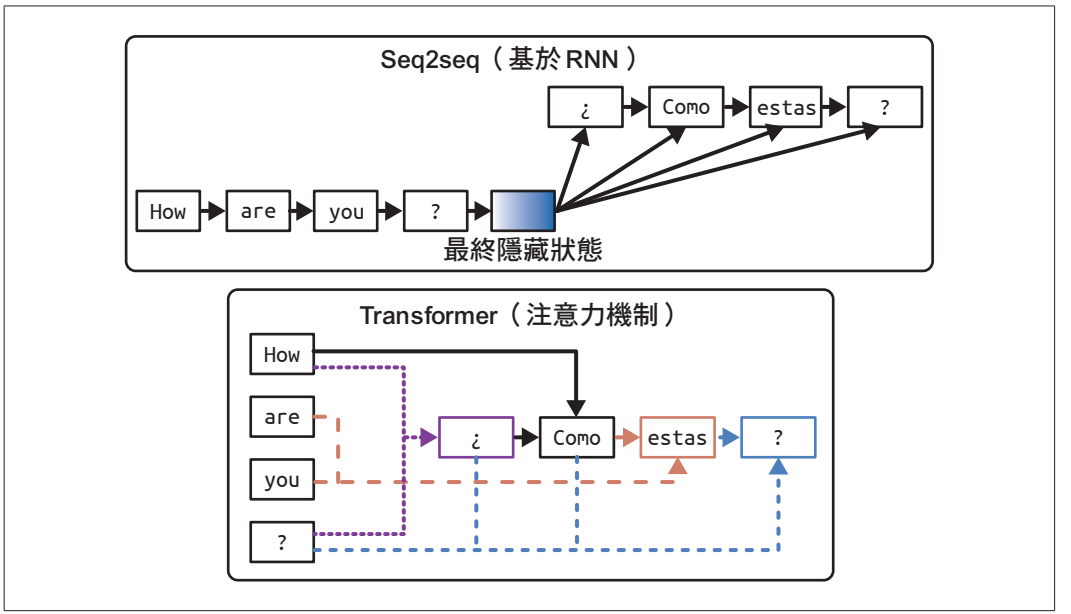

圖 2-4　seq2seq 架構與 Transformer 架構。Transformer 架構中，箭頭表示解碼器在生成每個輸出標記時關注的標記。

Vaswani 等人（2017）解決了 seq2seq 存在的兩個問題，首先是 seq2seq 解碼器僅依賴輸入的最終隱藏狀態生成輸出標記。直觀來說，這就像只根據書本的總結生成答案而限制了輸出品質。其次是 RNN 編碼器和解碼器要求輸入處理和輸出生成依序進行，對於

長序列來說，如果輸入有 200 個標記，seq2seq 必須等每個標記處理完成才能處理下一個，因而導致速度變得非常慢[6]。

Transformer 架構透過注意力機制解決了這兩個問題。注意力機制允許模型在生成每個輸出標記時，根據不同輸入標記的重要性進行取捨，這就像可用參考書中的所有頁面生成答案。圖 2-4 的下半部分展示了簡化的 Transformer 架構示意圖。

雖然注意力機制常與 Transformer 模型聯繫在一起，但它其實比 Transformer 論文早了三年出現，並且可與其他架構結合使用。Google 在 2016 年將注意力機制與 seq2seq 架構應用於其 GNMT（Google 神經機器翻譯）模型，但直到 Transformer 論文展示了注意力機制能在不依賴 RNN 的情況下運作，它才真正被廣泛接受[7]。

Transformer 完全摒棄 RNN 架構，它的輸入標記可以平行處理，大幅加快了輸入處理速度。雖然 Transformer 消除了順序輸入的瓶頸，但基於 Transformer 的自迴歸語言模型在輸出生成上仍存在順序瓶頸。

基於 Transformer 的語言模型推論分為兩個步驟：

預填充

模型平行處理輸入標記。這一步生成首個輸出標記所需的中間狀態，包括所有輸入標記的鍵向量和值向量。

解碼

模型按順序逐一生成輸出標記。

第 9 章會進行後續探討，這種預填充的平行性與解碼的順序性啟發了許多最佳化技術，進而使語言模型推論更有效率、更具成本效益。

6 由於 RNN 的遞迴結構，它特別易受梯度消失和梯度爆炸影響。梯度需透過多步傳播，太小則多次相乘後趨近於零，使模型難以學習，太大則因指數增長而導致訓練不穩定。

7 Bahdanau 等人，「Neural Machine Translation by Jointly Learning to Align and Translate」（暫譯：透過聯合學習對齊和翻譯的神經機器翻譯）（*https://arxiv.org/abs/1409.0473*）。

注意力機制　Transformer 架構的核心是注意力機制，理解這一機制是理解 Transformer 運作的關鍵。注意力機制運用了鍵、值和查詢向量：

- 查詢向量（Q）呈現解碼器在每個解碼步驟的當前狀態。以書本總結為例，查詢向量就像尋找資訊來撰寫總結的人。
- 每個鍵向量（K）呈現之前的標記。如果把每個標記比作書中的一頁，鍵向量就像頁碼。注意解碼步驟中的之前標記包含了輸入標記和之前已生成的標記。
- 每個值向量（V）呈現由模型學習生成的之前標記實際內容。每個值向量就像頁面的具體內容。

注意力機制透過計算查詢向量與鍵向量的點積（*https://en.wikipedia.org/wiki/Dot_product*），判斷模型對每個輸入標記的關注程度，高分表示模型在生成總結時會更多參考該標記的內容（即其值向量）。圖 2-5 展示了帶有鍵、值和查詢向量的注意力機制示意圖。在此圖中，查詢向量從之前的標記中尋找資訊，以生成下一個標記。圖中的查詢向量從之前的 `How, are, you, ?, ¿` 產生下個標記。

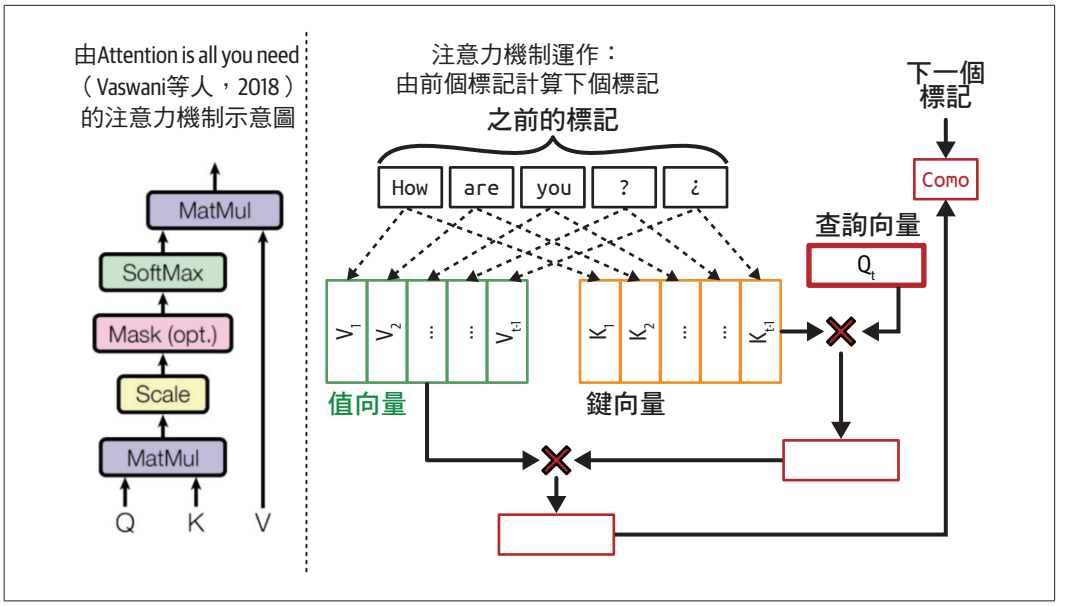

圖 2-5　注意力機制運作示意圖，左邊是著名 Transformer 論文「注意力即所需的一切」（Vaswani 等人，2017）。

MLP 模塊

MLP 模塊由線性層組成，中間以「非線性啟動函數」隔開。每個線性層是一個執行線性轉換的權重矩陣，而啟動函數則幫助線性層學習非線性模式。線性層也常被稱為前饋層。

常用的非線性啟動函數包括分別應用於 GPT-2 和 GPT-3 的 ReLU（Agarap，2018）（*https://arxiv.org/abs/1803.08375*） 和 GELU（Hendrycks 和 Gimpel，2016）（*https://arxiv.org/abs/1606.08415*）。這些啟動函數的運作很簡單[9]，例如 ReLU 只是將負值轉為 0，數學表達為：

$$\mathrm{ReLU}(x) = \max(0, x)$$

在 Transformer 模型中，Transformer 區塊的數量通常被稱為模型的層數。基於 Transformer 的語言模型通常在所有 Transformer 區塊前後會各配置一個額外的模塊：

Transformer 區塊前的嵌入模塊

這個模塊包含分別將標記及其位置轉換為嵌入向量的嵌入矩陣和位置嵌入矩陣。簡單來說，位置索引的數量決定了模型的最大上下文長度。例如若模型支援 2,048 個位置，其最大上下文長度就是 2,048，但技術上還可以透過某些方法，在不增加位置索引數量的情況下擴展上下文長度。

Transformer 區塊後的輸出層

這個模塊將模型的輸出向量映射為標記的機率分佈，用於取樣模型輸出（詳見「取樣」小節）。這個模塊通常包含一個稱為「反嵌入層」的矩陣。由於它是生成輸出前的最後一層，也有人將其稱為模型「頭」（head）。

圖 2-6 展示了 Transformer 模型架構示意圖。模型的規模由其組成模塊的維度決定，重要的值包含：

- 模型的維度，決定了 Transformer 區塊中鍵、查詢、值和輸出投射矩陣的規模。
- Transformer 區塊的數量。
- 前饋層的維度。
- 詞彙規模。

9 為什麼簡單啟動函數適用於像 LLM 這樣複雜的模型呢？曾有一段時間，研究人員競相提出複雜的啟動函數，但結果顯示花裡胡哨的函數效果並不好。模型只需非線性函數以打破前饋層的線性而已，運算更快、更簡單的函數較好，複雜函數反而耗費過多的訓練算力和記憶體。

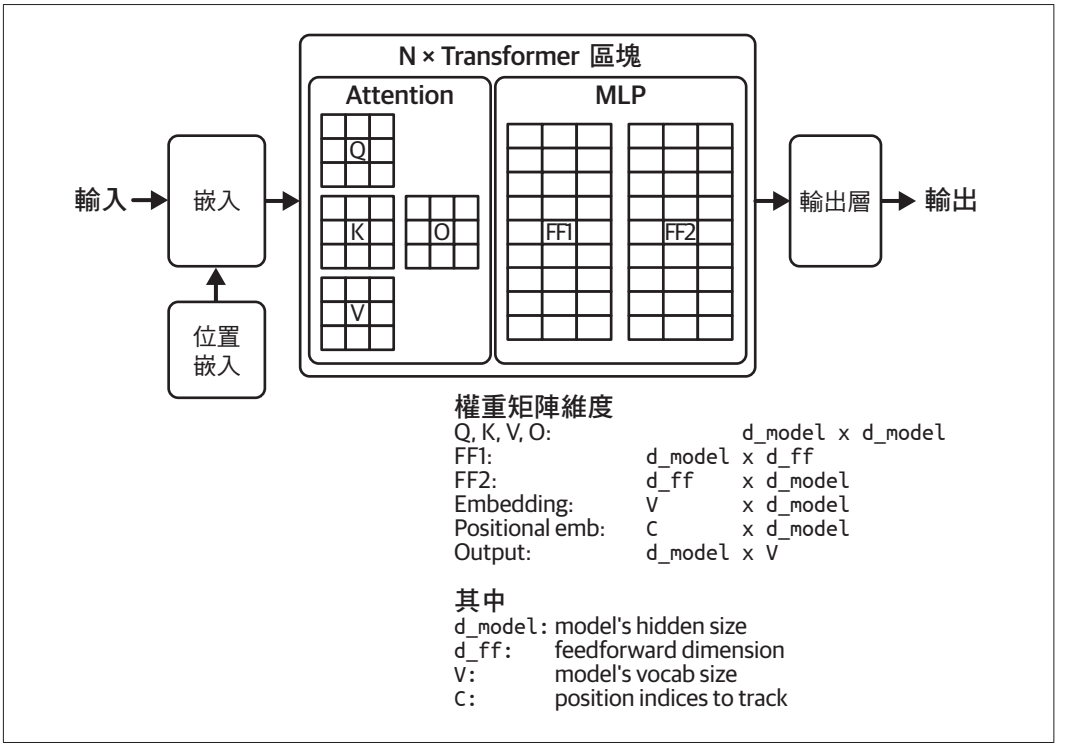

圖 2-6　Transformer 模型權重組成。

更大的維度值會導致模型規模更大。表 2-4 列出了不同的 Llama 2（Touvron 等人，2023）（*https://arxiv.org/abs/2307.09288*）和 Llama 3（Dubey 等人，2024）（*https://arxiv.org/abs/2407.21783*）的模型維度值。需要注意的是，雖然增加上下文長度會影響模型的記憶體利用率，但不會改變總參數數量。

表 2-4　不同 Llama 模型的維度值。

模型	Transformer 區塊數	模型維度	前饋維度	詞彙規模	上下文長度
Llama 2-7B	32	4,096	11,008	32K	4K
Llama 2-13B	40	5,120	13,824	32K	4K
Llama 2-70B	80	8,192	22,016	32K	4K
Llama 3-7B	32	4,096	14,336	128K	128K
Llama 3-70B	80	8,192	28,672	128K	128K
Llama 3-405B	126	16,384	53,248	128K	128K

其他模型架構

雖然 Transformer 模型目前占居主流地位，但它並非唯一的架構。自從 AlexNet（*https://oreil.ly/1spG5*）在 2012 年掀起深度學習熱潮以來，就有各種架構輪番登場。seq2seq 流行了四年（2014-2018），GAN（生成對抗網路）（*https://arxiv.org/abs/1406.2661*）則吸引了更長時間的關注（2014-2019）。相比之下，Transformer 的持續性更強，自 2017 年以來一直屹立不倒[10]。那麼，還要多久才會出現更好的架構呢？

開發一個超越 Transformer 的全新架構並不容易[11]。自 2017 年以來，Transformer 已經過大幅度的最佳化，因此要取代它的新架構必須在規模和硬體都有很好的表現才有機會[12]。

不過還是有一點希望啦！雖然目前 Transformer 模型處於主流地位，但已有一些替代架構開始受到關注了。

RWKV 是一個熱門的模型（Peng 等人，2023）（*https://github.com/BlinkDL/RWKV-LM*），其基於 RNN 且支援平行訓練。由於其 RNN 特性，理論上它不像 Transformer 模型那樣會受上下文長度限制，但實際上，無限上下文長度並不保證其在長上下文任務上的表現優於 Transformer。

長序列建模仍是 LLM 開發的核心挑戰之一。SSM（狀態空間模型，Gu 等人，2021a）（*https://arxiv.org/abs/2110.13985*）是一種在長距離記憶中展現巨大潛力的架構，自 2021 年推出以來，已有許多提升效率、長序列處理能力和模型擴展性的技術。以下是幾個展示新架構演進的技術：

- S4，出現在《*Efficiently Modeling Long Sequences with Structured State Spaces*》（Gu 等人，2021b）（*https://arxiv.org/abs/2111.00396*），提升了 SSM 的效率。
- H3，出現在《*Hungry Hungry Hippos: Towards Language Modeling with State Space Models*》（Fu 等人，2022）（*https://arxiv.org/abs/2212.14052*），引入機制讓模型回顧早期標記並比較序列中的標記，類似 Transformer 的注意力機制，但更高效。

10 有趣的事實是：OpenAI 聯合創始人 Ilya Sutskever 是 seq2seq 論文第一作者和 AlexNet 論文第二作者。

11 Ilya Sutskever 提出一個有趣論點，解釋了為何開發超越現有神經網路架構的新架構會如此困難的原因。他認為，神經網路擅長模擬多種電腦程式，梯度下降實際上是搜索神經網路能模擬的所有程式、以找到最佳化的演算法。新架構若要超越現有架構，必須模擬現有架構所無法實現的程式。詳情可觀看 Sutskever 在 Berkeley Siemens 研究所（2023）的演講（*https://oreil.ly/j4wwW*）。

12 Transformer 最初由 Google 設計用來在其 TPUs 上快速運行，後來也在 GPU 上最佳化。

- Mamba，出現在《*Mamba: Linear-Time Sequence Modeling with Selective State Spaces*》（Gu 和 Dao，2023）（*https://oreil.ly/n7wYO*），將 SSM 擴展到 30 億參數。在語言建模中，Mamba-3B 超越了同等規模的 Transformer，甚至與兩倍規模的 Transformer 匹敵。作者指出，相較於 Transformer 的平方增長，Mamba 的推論運算隨序列長度線性增長，並在真實數據上顯示出處理百萬長度序列的改進。
- Jamba，出現在《*Jamba: A Hybrid Transformer–Mamba Language Model*》（Lieber 等人，2024）（*https://arxiv.org/abs/2403.19887*），結合 Transformer 和 Mamba 進一步擴展 SSM。這是一個設計用於單個 80 GB GPU、總計 520 億參數的混合專家模型，其中 120 億為活躍參數（*https://oreil.ly/uyiBH*）。Jamba 在標準語言模型基準和長上下文評估中表現出色，上下文長度達 256 K 個標記，且比普通 Transformer 佔用更少的記憶體。

圖 2-7 展示了 Transformer、Mamba 和 Jamba 區塊。

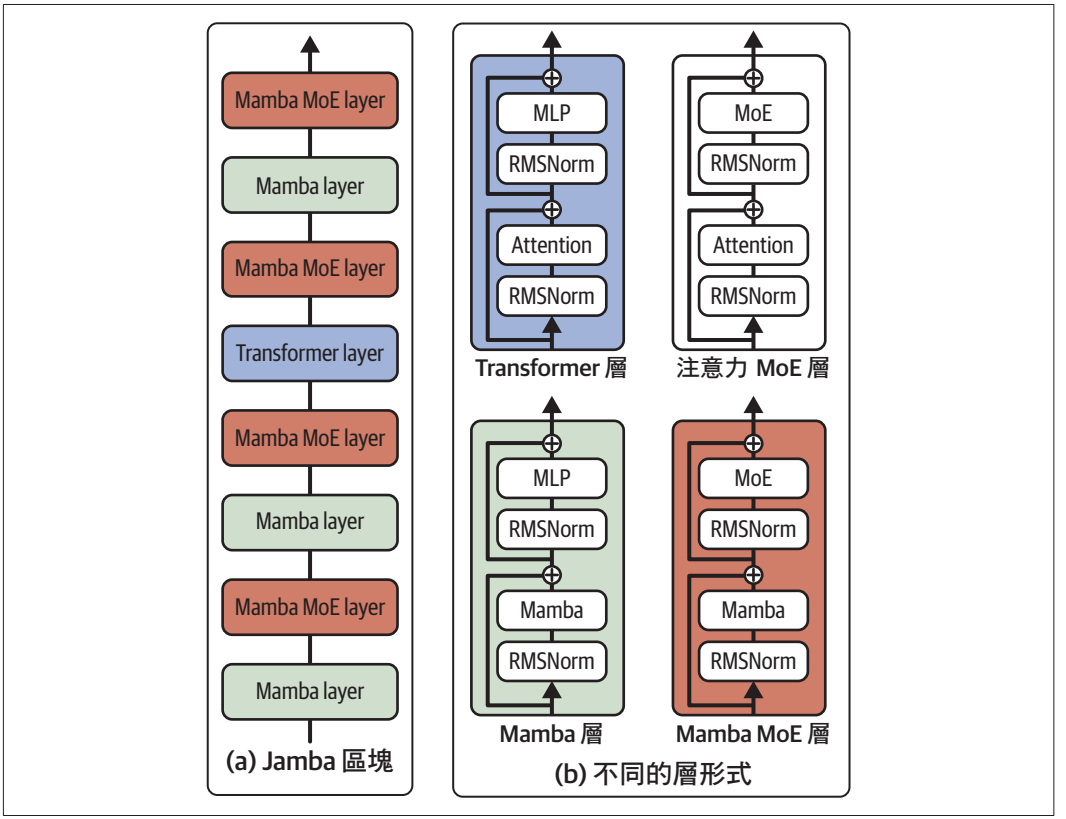

圖 2-7　Transformer、Mamba 和 Jamba 的各層示意圖。圖片改編自《Jamba: 混合 Transformer–Mamba 語言模型》（Lieber 等人，2024）。

雖然部分微調能降低記憶體佔用，但其參數效率不高。研究顯示，部分微調需要較多的可訓練參數才能接近全微調的性能。Houlsby 等人（2019）（*https://arxiv.org/abs/1902.00751*）在 BERT large（Devlin 等人，2018）（*https://arxiv.org/abs/1810.04805*）上的研究表明，要在 GLUE 基準（Wang 等人，2018）（*https://arxiv.org/abs/1804.07461*）上達到與全微調相當的性能，需更新約 25% 的參數。圖 7-7 展示了各種可訓練參數數量的部分微調性能曲線。

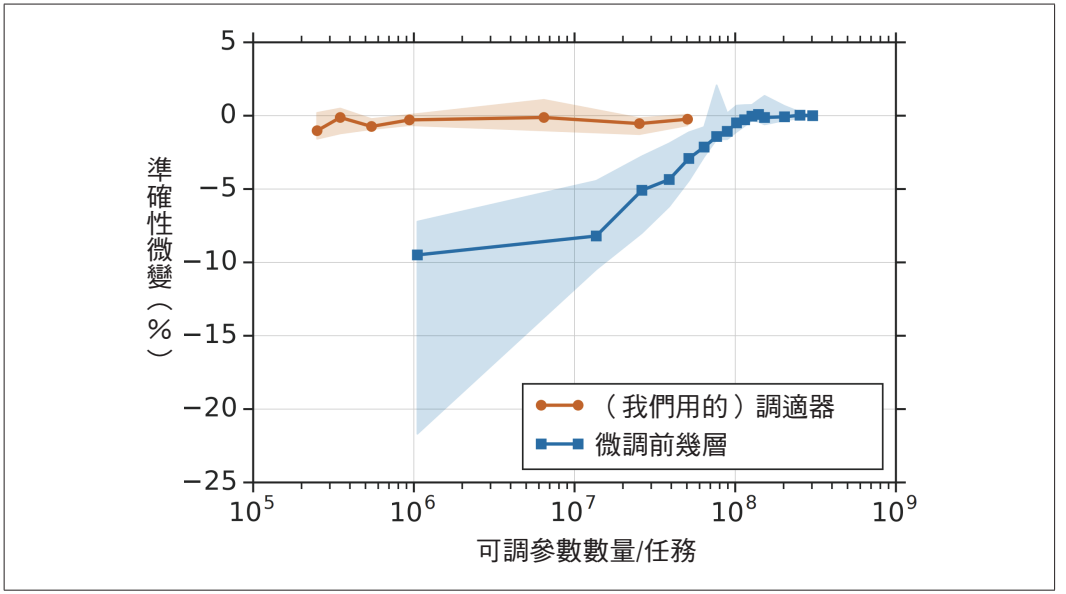

圖 7-7　藍線顯示部分微調需要多少可訓練參數才能達到與全微調相當的性能。圖片來自 Houlsby 等人（2019）。

這引發了一個問題：如何在大量減少可訓練參數的同時，實現接近全微調的性能？這一追求催生了 PEFT。雖然沒有明確的門檻定義何為參數高效，但通常若某技術能以數量級更少的可訓練參數達到接近全微調的性能，就可視為參數高效。

PEFT 的概念由 Houlsby 等人（2019）首次提出。他們證明在模型的適當位置插入額外的參數，就可以用極少的可訓練參數實現強大的微調性能。他們在 BERT 模型的每個 Transformer 區塊中插入兩個調適模塊，如圖 7-8 所示。

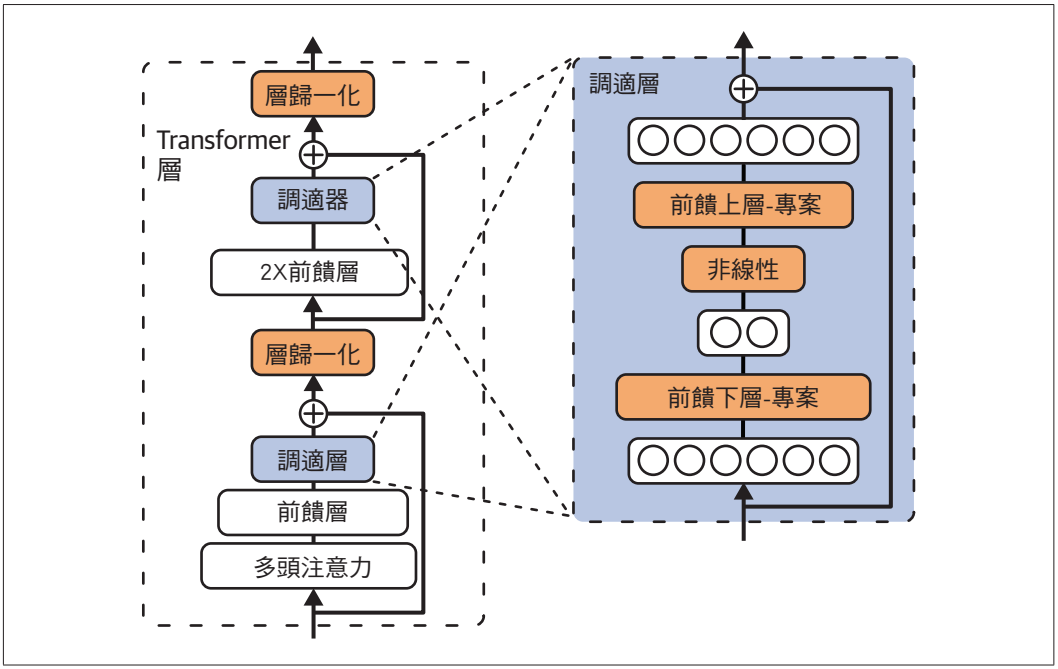

圖 7-8　在 BERT 模型的每個 Transformer 區塊中插入兩個調適模塊，僅更新這些調適模塊，Houlsby 等人（2019）即以少量可訓練參數實現了強大的微調性能。

在微調期間，原始模型參數保持不變，僅更新調適模塊。可訓練參數數量即為調適模塊的參數數量。在 GLUE 基準上，他們僅使用 3% 的可訓練參數數量，就拉近到全微調性能的 0.4% 以內。圖 7-7 中的橙線展示了全微調與不同調適規模微調的性能差距。

然而，這種方法的缺點是增加了微調模型的推論延遲。調適模塊引入額外的層，增加了順向傳播的運算步驟，因而減慢推論速度。

PEFT 使得微調得以在更經濟的硬體上進行，造福更多開發者。此外，PEFT 方法不僅參數高效，通常也可達到樣本高效。全微調可能需要數萬到數百萬個案例才能顯著提升品質，而一些 PEFT 方法僅需數千個案例就能達到優異性能。

基於 PEFT 的顯著吸引力，其技術發展迅速。接下來我們將概述這些技術，並深入探討最常見的 PEFT 技術：LoRA。

來自 PyTorch 的推論最佳化案例研究

圖 9-14 展示了 PyTorch 團隊透過以下最佳化步驟，為 Llama-7B 提升產量的成果（PyTorch，2023）：

1. 使用 torch.compile 將模型編譯為更高效的內核。
2. 將模型權重量化為 INT8。
3. 進一步將模型權重量化為 INT4。
4. 加入推測解碼。

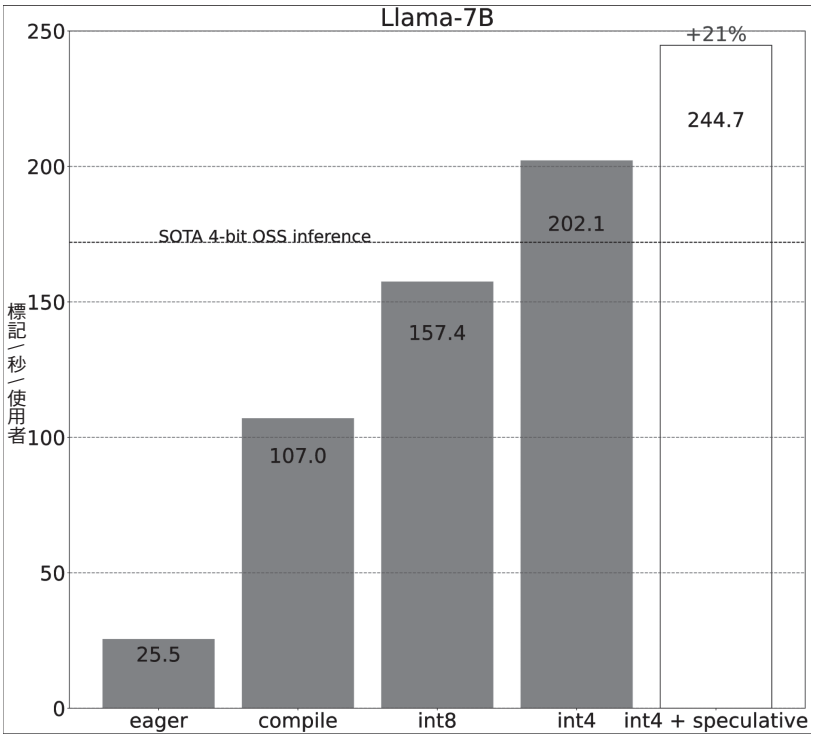

圖 9-14　PyTorch 中不同最佳化技術的產量改進。圖片來源：PyTorch（2023）。

該實驗在配備 80 GB 記憶體的 A100 GPU 上運行。目前尚不清楚這些最佳化步驟如何影響模型的輸出品質。