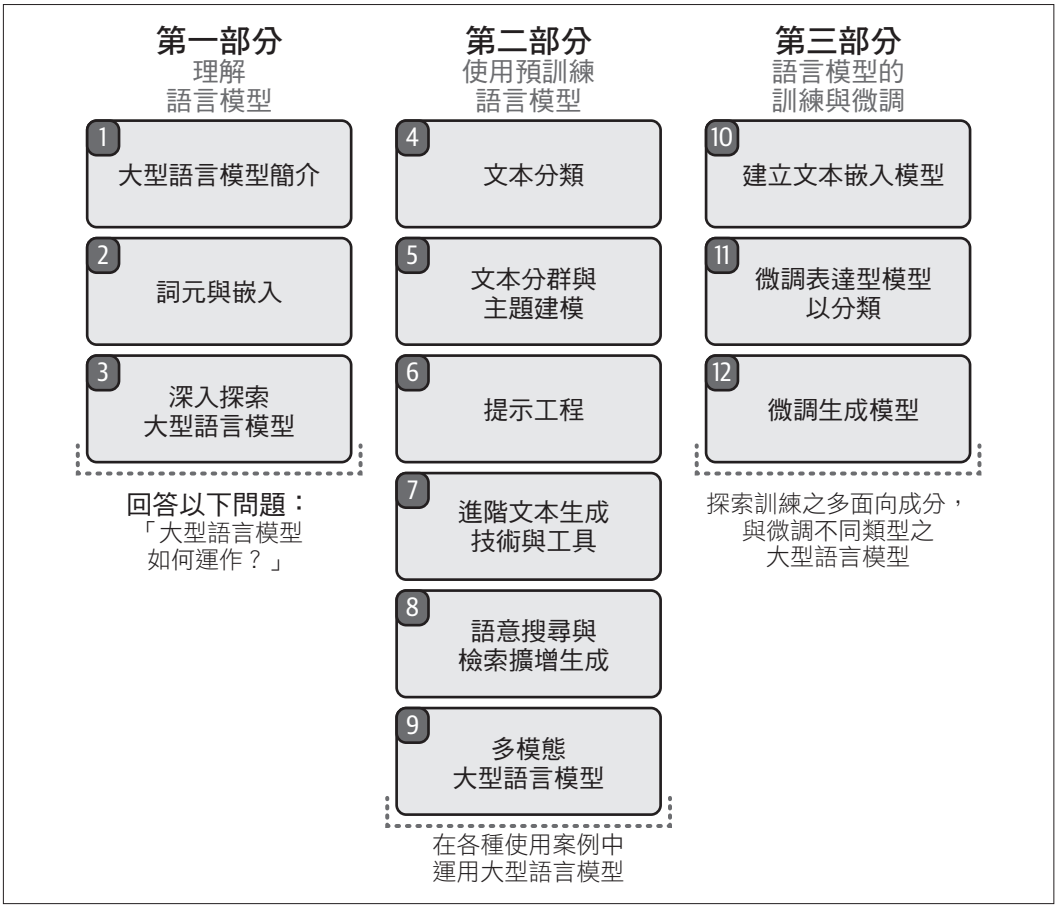

圖 P-1　本書的各部分與章節。

第一部分：理解語言模型

第一部分將探討語言模型的內部運作機制，涵蓋從小型到大型的模型，先從該領域的概述與常見技術開始（見第 1 章），接著深入分析這些模型的兩個核心組件：分詞化（tokenization）與嵌入（embedding）（見第 2 章）。本部分最後將呈現 Jay 的知名作品 Illustrated Transformer（*https://oreil.ly/UI4lN*）經過更新與擴展後的版本，並深入解析這些模型的架構（見第 3 章）。此部分還將介紹本書中使用的許多術語與定義。

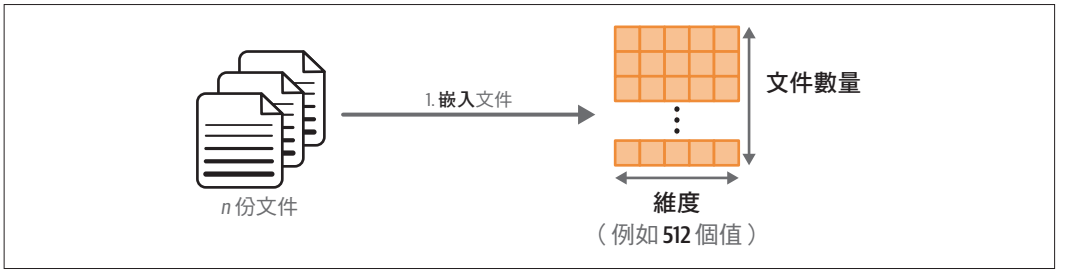

圖 5-3　第 1 步：使用嵌入模型將文件轉換為嵌入。

選擇有針對語意相似性任務進行優化的嵌入模型，對分群來說特別重要，因為我們試圖找到語意相似的文件群組。幸運的是，目前大多數嵌入模型都專注於語意相似性。

如前一章所述，這裡將使用 MTEB leaderboard（*https://oreil.ly/XFrbO*）來選擇嵌入模型。在分群任務中需要一個在效能和推理速度間取得平衡的嵌入模型，這裡不再使用上一章的「sentence-transformers/all-mpnet-base-v2」模型，而是改用「thenlper/gte-small」（*https://oreil.ly/h-Gkg*）模型。這個模型較新，對分群任務表現更佳且推理速度更快。

```
from sentence_transformers import SentenceTransformer

# 為每個摘要生成嵌入
embedding_model = SentenceTransformer("thenlper/gte-small")
embeddings = embedding_model.encode(abstracts, show_progress_bar=True)
```

讓我們檢查每個文件嵌入包含多少數值：

```
# 檢查嵌入的維度
embeddings.shape
```

```
(44949, 384)
```

每個嵌入有 384 個數值，這些數值共同代表文件的語意表達法，可以將這些嵌入視為用於分群的特徵。

降低嵌入的維度

在對嵌入分群之前，需要考慮其高維度的特性。隨著維度數量的增加，每個維度內可能值的數量也將呈指數增長，在每個維度中找到所有的子空間也就越來越複雜。

因此，高維資料對許多分群技術來說可能具有挑戰性，因為它使識別有意義的分群變得更加困難。可以使用降維（dimensionality reduction）技術來解決這個問題，如圖 5-4 所示，此技術可以減少維度空間的大小，並用更少的維度來表達相同的資料。降維技術旨在透過找到低維表達法來保留高維資料的全域結構。

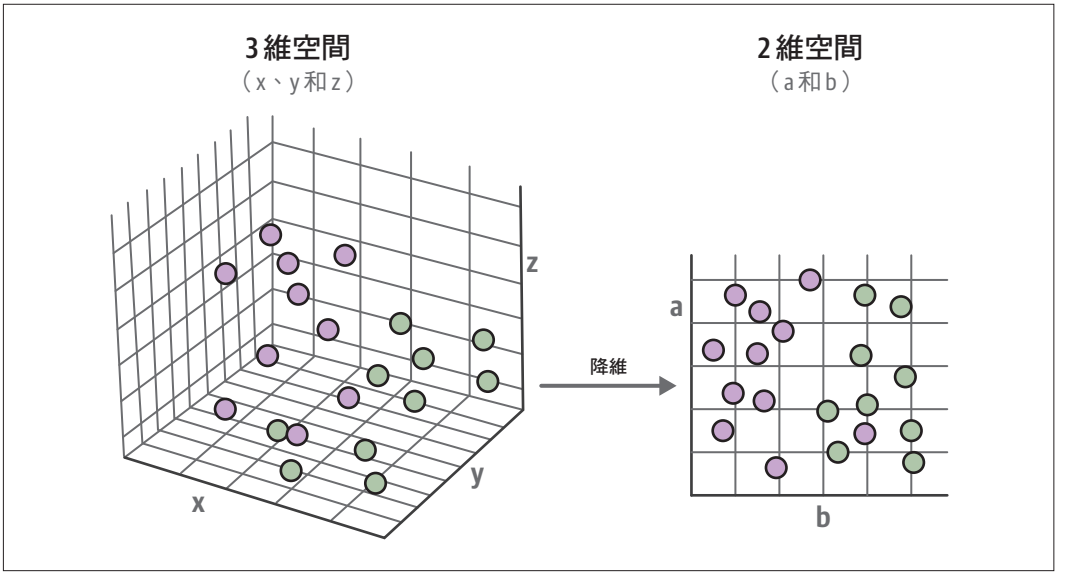

圖 5-4　降維使高維空間中的資料壓縮為低維表達法。

請注意，這是一種壓縮技術，底層演算法並非隨意地刪除維度。為了幫助分群模型建立有意義的分群，流程中的第二步是進行降維處理，如圖 5-5 所示。

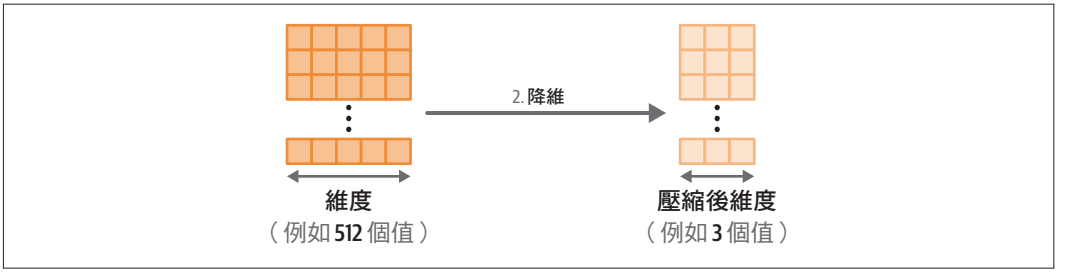

圖 5-5　第 2 步：使用降維技術將嵌入壓縮為較低的維度空間。

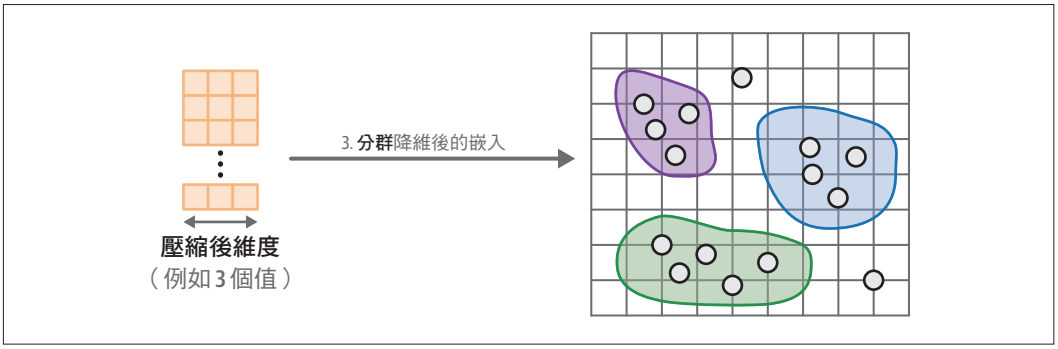

圖 5-6　第 3 步：使用降維後的嵌入進行文件分群。

雖然常見的選擇是基於中心的演算法，例如需要指定想生成分群數量的 k-means，但事先並不知道有多少分群。這時就要選擇基於密度（density-based）的演算法，因為這種演算法可以自由計算分群數量，並且不強制所有資料點都屬於某個分群，如圖 5-7 所示。

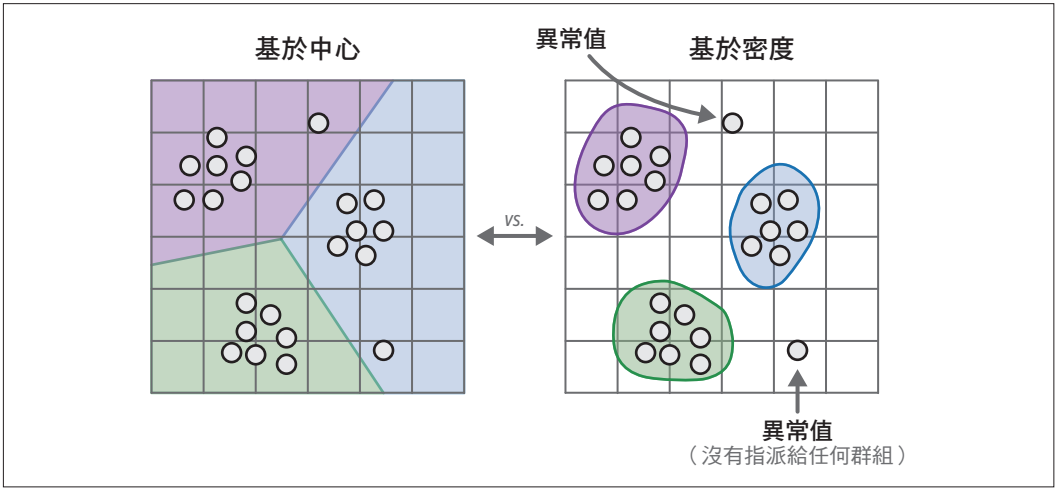

圖 5-7　分群演算法不僅影響分群的生成方式，也影響它們的呈現方式。

解決方法是利用詞袋模型的優勢，即快速生成有意義的表達法，然後使用更強大但較慢的技術，例如嵌入模型，來進一步優化表達法；正如圖 5-19 所示，可以重新排名（rerank）最初的單字分布以改進結果表達法。需要注意的是，重新排名初始結果集這個概念，在神經搜尋（neural search）中非常常見，詳見第 8 章。

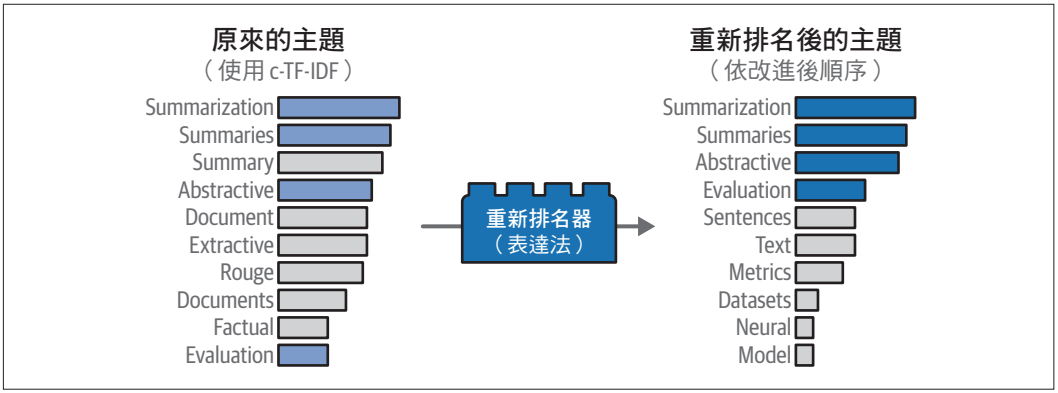

圖 5-19　透過重新排名原始 c-TF-IDF 分布來微調主題表達法。

因此可以設計一個新的樂高積木，如圖 5-20 所示，讓該積木接收初始主題表達法，並生成改進的表達法。

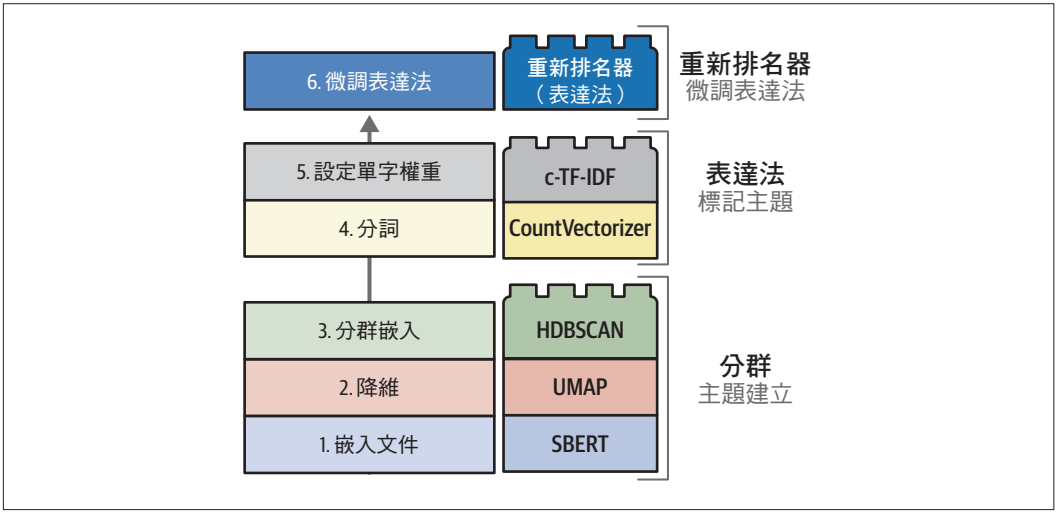

圖 5-20　重新排行表達法的積木位於 c-TF-IDF 表達法之上。

這些方法都可以透過 LangChain 框架（*https://oreil.ly/gmWSX*）輕鬆實現，本章將主要依賴該框架來應用這些進階技術。LangChain 是早期為簡化 LLM 操作而設計的一個框架，它提供許多有用的抽象化功能，值得注意的是，其他新興框架如 DSPy（*https://oreil.ly/DJ-wf*）和 Haystack（*https://oreil.ly/HgE7q*）也具有相似能力。LangChain 的部分抽象化功能如圖 7-1 所示，檢索功能則將在下一章討論。

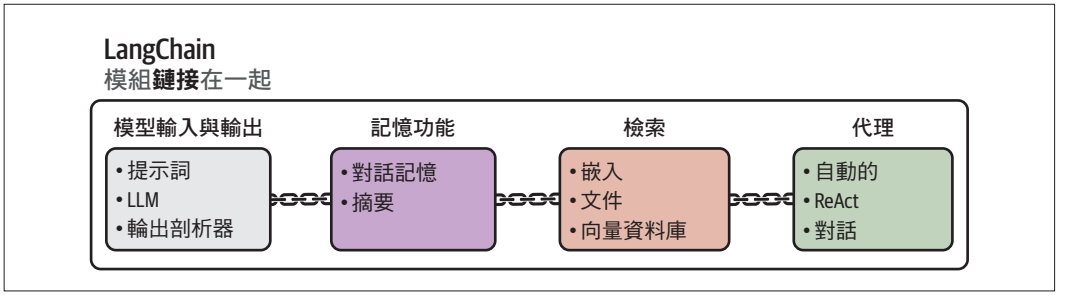

圖 7-1　LangChain 是一個完整的 LLM 框架，包含模組化的組件，可以將它們鏈接在一起以實現複雜的 LLM 系統。

每一項技術本身都有其明顯優勢，但它們的真正價值並不在於單獨使用，將這些技術結合在一起才能展現 LLM 系統的非凡效能；這些技術的融合是 LLM 真正大放異彩之處。

模型輸入與輸出：使用 LangChain 載入量化模型

在利用 LangChain 的功能來擴展 LLM 的能力之前，首先需要載入 LLM。如同前幾章，這裡將使用 Phi-3，但這次的不同之處在於，會使用 GGUF 模型變體。GGUF 模型是其原始模型的壓縮版本，透過一種稱為量化（quantization）的方法達成，能減少表達大型語言模型參數所需的位元數量。

位元（bit）是一系列 0 和 1，透過二進位形式來編碼值。更多位元可以表達更廣泛的數值範圍，但也需要更多記憶體來儲存這些數值，如圖 7-2 所示。

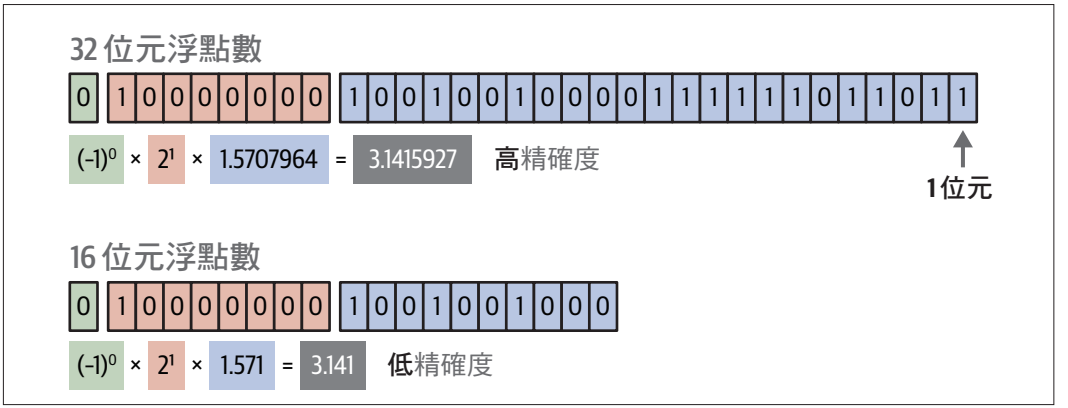

圖 7-2　試圖用 32 位元浮點數和 16 位元浮點數來表達 π。注意當位元數減半時，精確性會有所降低。

量化旨在減少表達 LLM 參數所需的位元數，同時又盡可能保留原始資訊。這會帶來一些精確度（precision）的損失，但通常可以透過提升模型執行的速度、降低 VRAM 的需求來補償，而且模型的準確度通常與原始版本相差無幾。

可用以下類比說明量化：當有人問您現在幾點時，答案可能是「14 時 16 分」，這是一個雖然正確但不算精確的答案，可以說「14 時 16 分 12 秒」，這會更準確，但是提到秒數通常沒有這個必要性，所以常常會簡化為完整的分鐘數。量化的過程很類似，它降低數值的精確度，就像移除秒數，但不會移除關鍵資訊，例如保留小時和分鐘。

第 12 章將深入探討量化的運作原理，也可以參考 Maarten Grootendorst 的《A Visual Guide to Quantization》（*https://oreil.ly/9Xt8U*）來獲取更直觀的理解。現在只需要知道，這裡將使用 Phi-3 的 8 位元變體，相比原始的 16 位元變體，記憶體需求幾乎減少了一半。

最好選擇至少 4 位元量化模型，因為這類模型能在壓縮與準確性之間取得良好平衡。雖然可以使用 3 位元甚至 2 位元的量化模型，但效能下降會很明顯，因此，精確度更高的小型模型通常是比較好的選擇。

首先需要下載模型（*https://oreil.ly/uXIeB*）。請注意，連結中包含多個不同位元版本的檔案，這裡選擇的 FP16 是 16 位元變體：

```
!wget https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/main/Phi-3-
mini-4k-instruct-fp16.gguf
```

最基本的鏈接形式是一個單鏈接。雖然鏈接的形式和複雜性可以百百種，但通常它會將 LLM 與額外工具、提示詞或功能連接起來。這種將組件連接到 LLM 的概念如圖 7-3 所示。

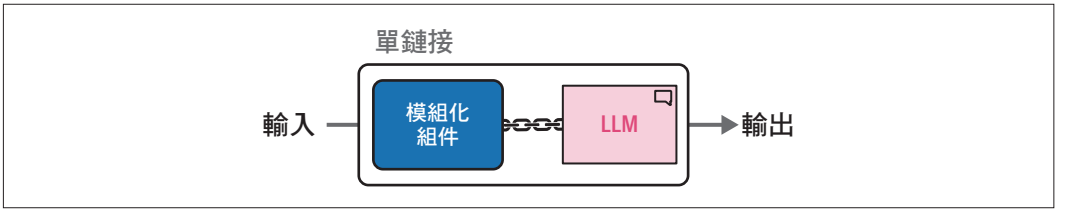

圖 7-3　單鏈接將一些模組化組件，如提示詞模板或外部記憶體連接到大型語言模型（LLM）。

實務上，鏈接的複雜性可能會迅速增加，可以根據需要擴展提示詞模板，甚至可以將多個獨立的鏈接結合在一起來建立複雜的系統。為了徹底了解鏈接的運作方式，這裡將探討 Phi-3 提示詞模板添加到 LLM 的方法。

單鏈接範例：提示詞模板

從建立第一個鏈接開始，即 Phi-3 所需的提示詞模板，前一章曾探索使用 `transformers.pipeline` 來自動地應用聊天模板的方法。然而，並非所有框架都自動支援此功能，有些可能需要明確地定義提示詞模板。在 LangChain 中使用鏈接來建立和使用預設的提示詞模板，既能簡化使用 LLM 的流程，又能作為學習提示詞模板操作的實戰經驗。

如圖 7-4 所示，將提示詞模板與 LLM 鏈接以生成所需的輸出。這樣，只需定義使用者和系統的提示詞，而不必每次都複製貼上完整的模板。

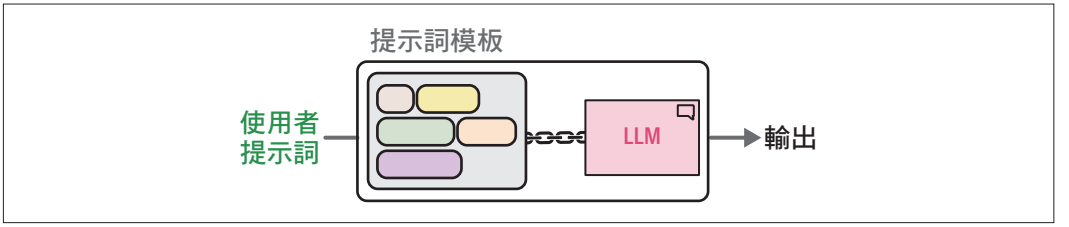

圖 7-4　透過將提示詞模板與大型語言模型鏈接起來，只需定義輸入提示詞，模板就會自動建構。

Phi-3 的模板由 4 個主要組件組成：

- `<s>`：指出提示詞的開始。
- `<|user|>`：指出使用者提示詞的開始。
- `<|assistant|>`：指出模型輸出的開始。
- `<|end|>`：指出提示或模型輸出的結束。

這些組件如圖 7-5 所示，並輔以一個範例。

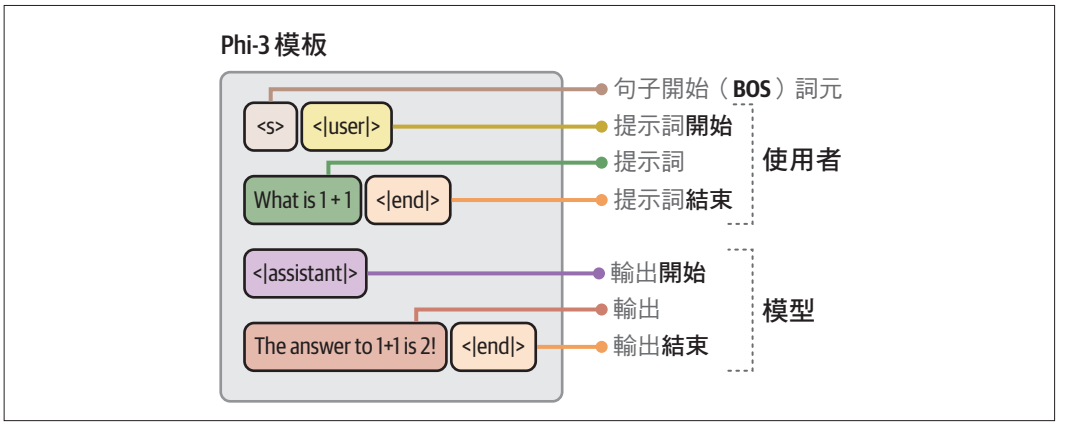

圖 7-5　Phi-3 期待的提示詞模板。

要生成一個簡單的鏈接，首先需要建立一個符合 Phi-3 模型所需格式的提示詞模板。使用此模板，模型會接受一個 `system_prompt`，通常用於描述我們期望的 LLM 表現，接著再用 `input_prompt` 向 LLM 提出具體問題。

```
from langchain import PromptTemplate

# 建立帶有 "input_prompt" 變數的提示詞模板
template = """<s><|user|>
{input_prompt}<|end|>
<|assistant|>"""
prompt = PromptTemplate(
    template=template,
    input_variables=["input_prompt"]
)
```

接下來，可以將這個提示詞模板與 LLM 結合起來，形成第一個鏈接：

```
basic_chain = prompt | llm
```

多模態大型語言模型

說到大型語言模型時，多模態（multimodality）可能不會是第一個浮現的概念，畢竟，它們是語言模型！但很快就能看到，如果模型能夠處理文字以外的資料類型，將變得更加有用。例如，若語言模型能夠快速查看一張圖片並回答有關它的問題，絕對能派上用場。能夠處理文本和影像的模型可稱為多模態，因為這兩者都是一種模態（modality），如圖 9-1 所示。

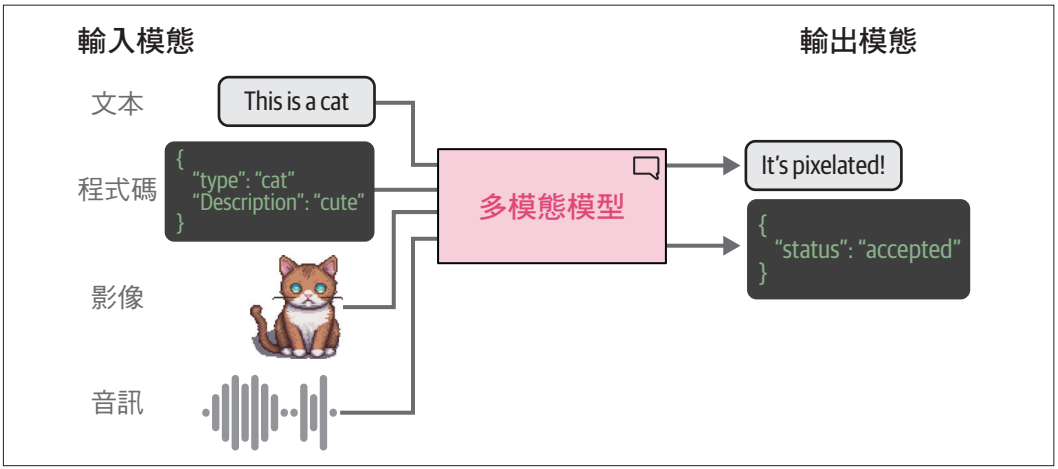

圖 9-1　能夠處理不同類型或模態資料的模型，如影像、音訊、視訊或感測器稱為多模態。模型有可能接受某一模態作為輸入，但並不一定能夠在該模態中生成內容。

前面已經見識到來自 LLM 的各種新興行為，從泛化能力和推理能力到算術和語言學。而隨著模型變得越來越大、越來越聰明，它們的技能也在不斷增長[1]。

接收和推理多模態輸入的能力，可能進一步增加並幫助開發之前無法實現的能力。在實際情況下，語言並非孤立存在，例如，肢體語言、面部表情、語調等都是增強口語表達的溝通方式。

同樣的道理也適用於 LLM；如果能讓它們推理多模態資訊，可能會增強它們的能力，並且可以部署它們來解決新的問題。

本章將探討多種具有多模態能力的 LLM 及其在實際應用中的意涵，就從探討使用改良版的 Transformer 技術，將影像轉換為數值表達法開始，接著將展示使用這個 Transformer 擴展 LLM 以包含視覺任務的方式。

用於視覺的 Transformer

本書的各章中已經看到基於 Transformer 的模型，成功的應用在各種語言建模任務中，從分類和分群到搜尋和生成式建模。因此，研究人員尋求將 Transformer 的一些成功經驗推廣到電腦視覺領域也就不足為奇。

他們提出的方法稱為視覺 Transformer（Vision Transformer，ViT），和以往預設的卷積神經網路（convolutional neural network，CNN）相比，它在影像識別任務中取得更為明顯的成功[2]。與原始 Transformer 一樣，可使用 ViT 將非結構化資料如影像，轉換為可以用於各種任務如分類的表達法，如圖 9-2 所示。

ViT 依賴於 Transformer 架構中的一個重要組件，即編碼器。正如第 1 章所言，編碼器負責將文本輸入轉換為數值表達法，然後再傳遞給解碼器。然而，在編碼器執行其任務之前，文本輸入需要先進行分詞處理，如圖 9-3 所示。

1 Jason Wei et al. “Emergent abilities of large language models.” *arXiv preprint arXiv:2206.07682* (2022).

2 Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale.” *arXiv preprint arXiv:2010.11929* (2020).

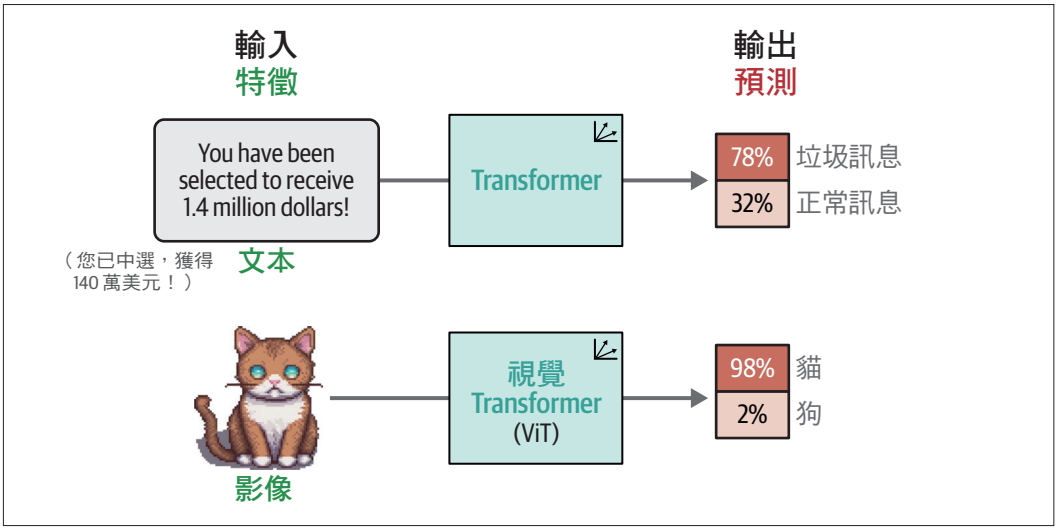

圖 9-2　原始 Transformer 和視覺 Transformer 都將非結構化資料轉換為數值表達法，並最終用於分類等任務。

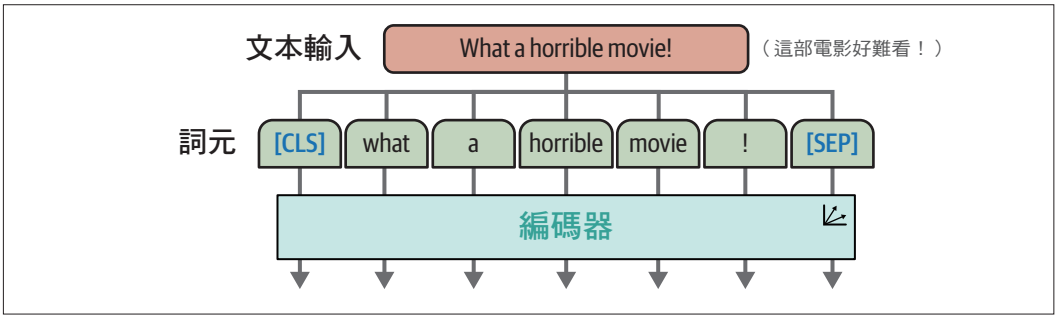

圖 9-3　文字經過分詞器處理後，傳遞到一個或多個編碼器以處理。

由於影像並非由單字組成，無法使用這種分詞處理過程來處理視覺資料；因此，ViT 的作者提出一種方法，將影像分詞為「單字」，好讓它們使用原始的編碼器結構。

想像一下，您擁有一張貓的影像，由若干個像素（pixel）組成，假設是 512×512 個像素。每個單獨的像素並不會傳遞太多的資訊，但將像素的區塊組合在一起時，就會慢慢開始看到更多資訊。

ViT 使用的原理類似於此。它並不是將文本拆分為詞元，而是將原始影像拆分成影像區塊；換句話說，它會將影像在水平方向和垂直方向上分割成若干個區塊，如圖 9-4 所示。

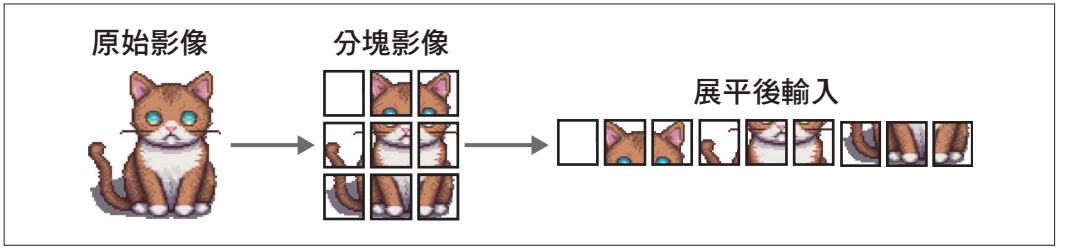

圖 9-4　影像輸入的「分詞」過程，將影像轉換為子影像的區塊。

就像將文本轉換為文本詞元一樣，這裡也將影像轉換為影像區塊，所以可以將這些展平後的影像區塊視為文本中的詞元。然而，與詞元不同的是，不能直接為每個區塊指派一個 ID，因為這些區塊在其他影像中很少出現，這與文本的字彙集不同。

相反的，這些區塊會線性地嵌入以建立數值表達法，即嵌入向量，隨後，就可以將這些嵌入向量作為 Transformer 模型的輸入。這樣，影像區塊就能得到與文本詞元一樣的處理方式，整個過程如圖 9-5 所示。

為了說明，將這些範例中的影像分成 3×3 的區塊，但原始實作使用 16×16 的區塊；畢竟，這篇論文的名稱是《An Image is Worth 16x16 Words》。

這種方法的有趣之處在於，將這些嵌入向量傳遞給編碼器時，會像文本詞元一樣處理它們；這樣一來，文本和影像的訓練方式就沒有區別了。

由於這些相似性，常以 ViT 用來將各類語言模型轉變為多模態模型，其中最直接的使用方式之一，是在嵌入模型的訓練過程中使用它。

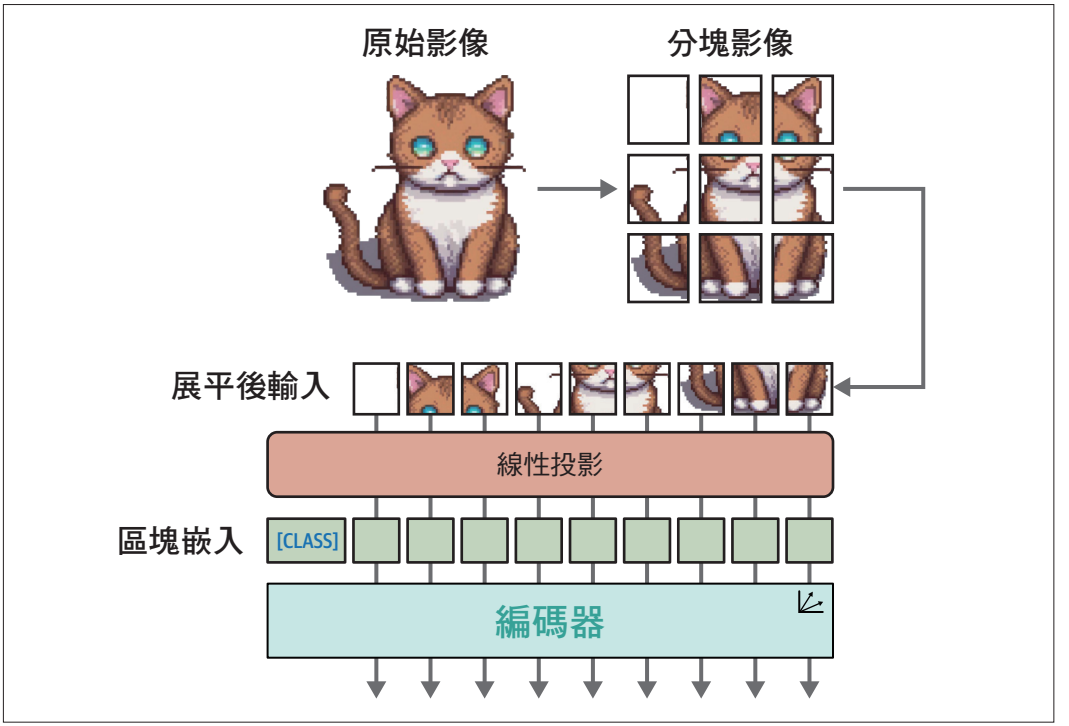

圖 9-5　ViT 背後的主要演算法。在將影像分割並進行線性投影後，會將這些區塊的嵌入向量傳遞給編碼器，並像文本詞元一樣處理。

多模態嵌入模型

在前幾章使用嵌入模型來捕捉文本表達法，例如論文和文件的語意內容後，就可以使用這些嵌入或數值表達法來尋找相似文件、應用分類任務，甚至主題建模。

正如之前多次看到的，嵌入往往是大型語言模型應用的重要驅動力，這是一種用於捕捉大規模資訊，並在資訊汪洋中尋找細針的有效率方法。

話雖如此，目前為止看到的嵌入模型主要都是針對文本，專注於為文本表達法生成嵌入。雖然也有專門嵌入影像的嵌入模型，但本章會將重心放在那些能夠同時捕捉文本和視覺表達的嵌入模型，如圖 9-6 的展示。

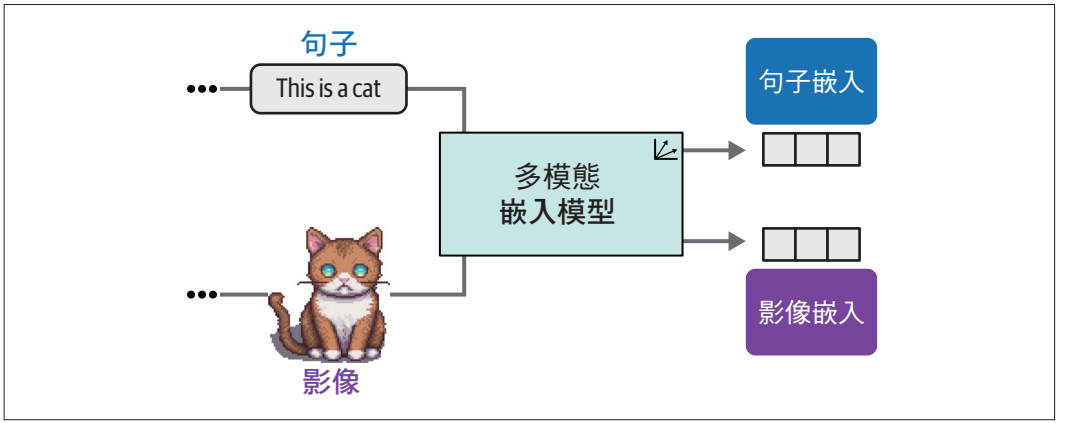

圖 9-6　多模態嵌入模型能夠在同一向量空間中為多種模態建立嵌入。

這樣的優勢在於它可以比較多模態表達法，因為生成的嵌入位於相同的向量空間（圖 9-7），例如，這樣的多模態嵌入模型可以根據輸入的文本尋找影像。如果搜尋和「小狗影像」類似的影像會找到什麼呢？反過來也行。哪些文件與這個問題最有相關性？

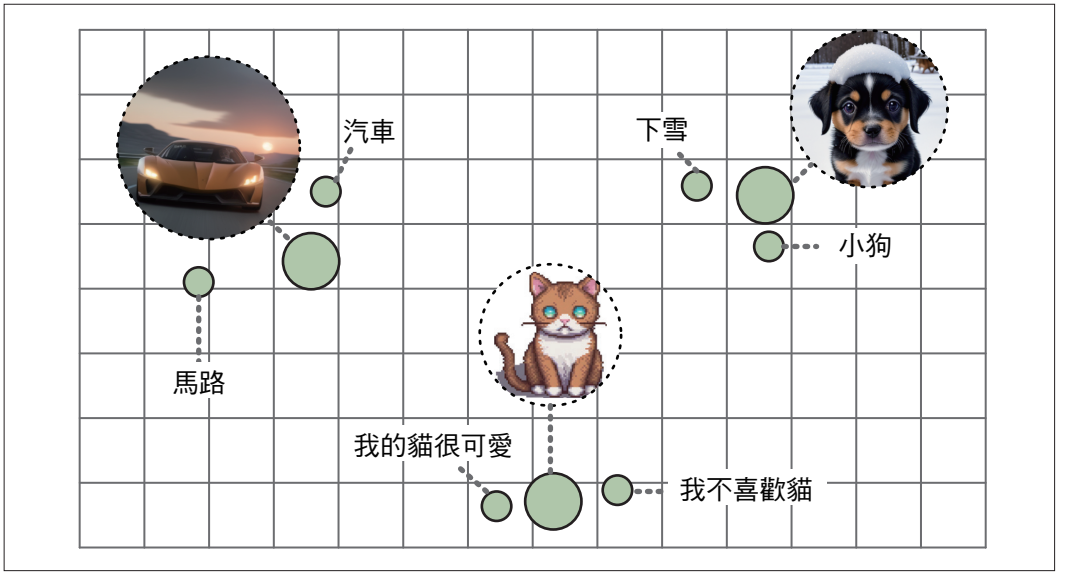

圖 9-7　儘管來自不同模態，具有相似語意的嵌入也會在向量空間中彼此接近。