目次

[一、建立 nvm 環境 1](#_Toc204981336)

[二、安裝 nodejs 與版本切換 5](#_Toc204981337)

[三、AI 基礎概論與演進 7](#_Toc204981338)

[四、AI 應用概論 10](#_Toc204981339)

[（一）電腦視覺應用 15](#_Toc204981340)

[（二）自然語言應用 15](#_Toc204981341)

[（三）知識挖掘應用 15](#_Toc204981342)

[（四）智慧文件處理 16](#_Toc204981343)

[（五）生成式 AI 應用 16](#_Toc204981344)

課程範例網址:

<https://github.com/telunyang/nodejs_ai_basics>

# 一、建立 nvm 環境

Node Version Manager (nvm) 是一款專門讓你安裝、切換、管理不同版本 Node.js 的工具，對於多專案開發、高相容性需求，或跨版本測試非常實用。它省去繁瑣手動安裝/解除安裝的麻煩，讓切換版本像打指令一樣簡單，適合開發者靈活使用不同版本環境。

Windows 版 nvm：<https://github.com/coreybutler/nvm-windows/releases>

原始官方下載網頁：<https://github.com/nvm-sh/nvm>

以 Windows 環境為例：

一張含有 文字, 螢幕擷取畫面, 軟體, 數字 的圖片

AI 產生的內容可能不正確。

圖：按下「nvm-setup.exe」，下載安裝檔

一張含有 符號, 螢幕擷取畫面, 圖形, 設計 的圖片

AI 產生的內容可能不正確。

圖：下載後的安裝檔圖示，請連續點兩下

（出現「您是否要允許此 App 變更您的裝置」，請按下**是**）

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

AI 產生的內容可能不正確。

圖：選擇「I accept the agreement」，再按下「Next」

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

AI 產生的內容可能不正確。

圖：安裝路徑自由切換，強烈建議路徑不要有 multi-bytes 的文字（例如中文、日文、韓文等非英語系文字），之後請按下「Next」

一張含有 文字, 電子產品, 螢幕擷取畫面, 軟體 的圖片

AI 產生的內容可能不正確。

圖：放置主要 node.js 版本的目錄，會被加到系統路徑，之後請按下「Next」

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

AI 產生的內容可能不正確。

圖：全部取消勾選（可依需求保留），按下「Next」

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

AI 產生的內容可能不正確。

圖：不用填寫（欲關注開發團隊動向除外），直接按下「Next」

一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

AI 產生的內容可能不正確。

圖：確認配置沒問題，按下「Install」，開始安裝

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

AI 產生的內容可能不正確。

圖：取消勾選「Open with Powershell」，按下「Finish」

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

圖：重新開啟程式編輯器如 VS code（已開啟，就先整個關掉再重開），打開 Terminal（終端機），輸入「nvm」，如果出現圖中內容，即安裝成功

# 二、安裝 nodejs 與版本切換

| 常用終端機指令 | 說明 |
| --- | --- |
| nvm | 觀看指令用法 |
| nvm list/ls | 列出所有已安裝的 node.js 版本 |
| nvm install lts/latest | 安裝長期支援/最新的 node.js 版本 |
| nvm use <版本號> | 切換到特定版本的 node.js (要先 nvm install ...) |
| nvm uninstall <版本號> | 刪除特定版本的 node.js |
| node -v | 切換完成後，確認當前環境的 node.js 版本號 |
| npm -v | 同上，確認 npm 版本 (node.js 的套件管理器) |

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

圖：列出所有安裝的 node.js 版本，目前尚未安裝

一張含有 文字, 字型, 螢幕擷取畫面, 標誌 的圖片

AI 產生的內容可能不正確。

圖：到 node.js 官方網站檢視當前長期支援版本

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

圖：安裝 lts 版本時，確認與官方 lts 版本一致。

完成後，輸入「nvm use <版本號>」

（出現「您是否要允許此 App 變更您的裝置」，請按下**是**）

一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

AI 產生的內容可能不正確。

圖：確認是否切換到當前安裝的版本

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

圖：再次確認安裝列表，可以看到安裝了哪些版本（星號代表 Active version）

一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

AI 產生的內容可能不正確。

圖：反安裝指定的版本（有需要再執行這個指令，上課先別刪...）

一張含有 文字, 字型, 螢幕擷取畫面, 圖形 的圖片

AI 產生的內容可能不正確。

圖：確認目前的 node.js 版本

一張含有 文字, 字型, 螢幕擷取畫面, 圖形 的圖片

AI 產生的內容可能不正確。

圖：確認目前的 npm 版本

# 三、AI 基礎概論與演進

人工智慧（AI，Artificial Intelligence）是電腦系統模擬人類智慧的能力，例如學習、推理、感知、語言理解與決策，AI 涉及的主要領域包括：機器學習（ML，Machine Learning）、自然語言處理（NLP，Natural Language Processing）、電腦視覺（CV，Computer Vision）…等。

AI 技術的層級與演進關係，採用由外而內、由上而下的結構：

* 最外層：Artificial Intelligence
  + 表示 AI 的整體範疇，涵蓋所有智慧化技術與應用。
* 中層：Machine Learning
  + 機器學習是 AI 的主要子領域，透過演算法與數據來讓系統自我學習與改進。
* 內層：Deep Learning
  + 深度學習是機器學習的子集，利用多層神經網路（如 CNN、RNN、Transformer）處理複雜任務。

兩個主要應用分支：

* Natural Language Processing (NLP)：語言處理，像 ChatGPT、翻譯與語音助手。
* Computer Vision (CV)：計算機視覺，用於圖像識別、自駕車與監控等。

一張含有 文字, 螢幕擷取畫面, 字型, 圓形 的圖片

AI 產生的內容可能不正確。

圖：AI 涉及的主要領域

表：演進時間軸

| 時間 | 里程碑與事件 |
| --- | --- |
| 古代–19世紀 | 希臘自動機與哲學家對「機械智慧」的思考，如亞里斯多德邏輯與 Leonardo 的機械人。 |
| 1950年 | Alan Turing 發表《Computing Machinery and Intelligence》，提出「機器能思考嗎？」與圖靈測試。 |
| 1956年 | John McCarthy 等人在達特茅斯研討會正式提出「artificial intelligence」一詞，被視為 AI 學科誕生。 |
| Logic Theorist 程式面世，由 Newell、Simon 與 Shaw 開發，是首個 AI 自動推理程式。 |
| 1958–60年代 | Rosenblatt 提出 perceptron（早期神經網路），McCarthy 推出 Lisp 語言，Samuel 發明能自學的西洋跳棋程式（machine learning 名稱首次出現）。 |
| 1970–80年代 | 專家系統與符號 AI 盛行（如 Dendral、ELIZA、Shakey 機器人）。 |
| 1974 至 1980 年間，進入第一次 AI 冬天，資金與研究興趣急遽下降。1969 年 Minsky & Papert 的《Perceptrons》揭露單層神經網路的根本局限性，動搖人們對連結主義的信心；1973 年英國 Lighthill 報告批評 AI 研究過度誇大，導致英國政府大幅停止資助；美國國防部高等研究計劃署（DARPA，Defense Advanced Research Projects Agency）因技術過度承諾與成果未達預期，從約 1973–74 年開始大幅削減非導向研究之資助。 |
| 第二次 AI 冬天（Second AI Winter）：約 1987–1993。  Lisp 機（LISP machines）市場在 1987 年崩盤，專門硬體被通用工作站取代，專屬硬體公司迅速倒閉；專家系統（expert systems）雖早期風光，但後期維護成本高、缺乏學習能力且失誤率高，被認為「脆弱」並逐漸被棄用；日本第五代電腦計畫（Fifth Generation Computer Project）未達預期，引發信心危機，專案逐漸停滯；DARPA 的 Strategic Computing Initiative 在 1988 年起逐步取消資金與計畫，進一步打擊研究環境。 |
| 1980年代後期 | Judea Pearl 提出貝葉斯網路、統計方法開始融入 AI；LeCun 等人開發 CNN 卷積神經網路。 |
| 1990–2000年代 | 統計學習與機器學習技術蓬勃發展，AI 工具開始在商業、醫療、語音與圖像識別等領域應用。 |
| 2000–2009 年 | 傳統機器學習 (classical ML) 的黃金期。隨著網路普及與大數據興起，支援向量機（SVM）、隨機森林、決策樹、集成方法等算法廣泛應用於多個領域，包括推薦系統、語音識別、金融預測、醫療影像等。 |
| 2003–2009 年 | 神經網路回歸起點。LSTM 在 2003 年崛起，2009 年在手寫辨識比賽中首次得獎，顯示 RNN 在時序資料處理上的潛力；2006 年，Hinton 與團隊提出深度信念網路（DBN），前導訓練技術為後續的深度學習奠基。 |
| 2004–2011 年 | GPU 加速與 CNN 再興起。2004–2006 年，研究者將 CNN 與一般神經網路實作於 GPU 上，加速比 CPU 快約數十倍；IDSIA 團隊於 2011 年開發的 DanNet CNN，在圖形辨識比賽上首次達到超越人類水平的辨識準確度。 |
| 2010–2012 年 | 深度學習引發革命。2010 年左右，工業界開始將深度學習應用於大規模語音辨識，並逐漸優於傳統方法；2012 年 AlexNet 的出現是關鍵轉捩點（AlexNet 的勝利，象徵深度學習正式超越傳統模型，開啟「AI 革命」的新篇章）；Krizhevsky、Sutskever、Hinton 使用 GPU 在 ImageNet 上訓練深層 CNN，錯誤率大幅領先傳統 ML 方法，引領深度學習時代開端；同年，Ng 和 Dean 利用深層模型從未標記 YouTube 影像中學習識別「貓」，展現無監督特徵學習能力。 |
| 2012年之後 | 深度學習崛起，強力加速器（GPU）與大數據驅動下，表現在語音與圖像領域突破性成果 |
| 2017年 | Transformer 架構問世，Attention 機制成為 NLP 的主流技術。 |
| 2020年代 | Generative AI（如 GPT 系列）、大規模語言模型迅速商業應用，帶動新一波 AI 熱潮；也引發倫理與政策關注。 |

註：<https://zh.wikipedia.org/zh-tw/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD%E5%8F%B2>

# 四、AI 應用概論

以下將以電腦視覺、自然語言處理、知識挖掘、智慧文件與生成式 AI 等應用來進行說明。以下的應用，需要用到 Google AI Studio 的 API key 以及 Huggingface 的 access token。

取得 Google AI Studio 的 API key

連結：<https://aistudio.google.com/>

備註：需要在 Chrome 登入 Google 帳號（課程結束後，沒有其它用途的話，強烈建議一律登出）。

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

AI 產生的內容可能不正確。

圖：進入 Google AI Studio 頁面，按下上方的「Get API key」

一張含有 螢幕擷取畫面, 字型, 文字, 圖形 的圖片

AI 產生的內容可能不正確。

圖：若是先前尚未建立 API key，請點擊一下

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

圖：若先前尚未建立 API key，可先建立 projects，之後再選擇建立好的 project 來建立 API key。按下「Create API key in existing project」

一張含有 字型, 圖形, 標誌, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

圖：產生 API key 的等待畫面

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

圖：API key 建立完成，請按下「Copy」後，貼在程式當中使用

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

AI 產生的內容可能不正確。

圖：若是忘記 API key，可以按下中間的連結，會出現上圖的畫面，讓你按下「Copy」

取得 Hugging Face 的 access token (非必要)

連結：<https://huggingface.co/>

備註：請先註冊帳號（課程結束後，沒有其它用途的話，強烈建議一律登出）。

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

AI 產生的內容可能不正確。

圖：登入後，按下右上角的圖示，會出現選單，按下「Access Tokens」

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

圖：尚未建立 token 的話，按下右上角「Create new token」

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

AI 產生的內容可能不正確。

圖：因為我們只是希望下載、讀取開放權重的模型，所以 Token type 選擇「Read」，Token name 可以任意填寫，例如「test」，之後按下「Create token」

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

AI 產生的內容可能不正確。

圖：此時會產生一個 token，建議額外記錄或暫存，不然它無法重新檢視 token

一張含有 螢幕擷取畫面, 多媒體軟體 的圖片

AI 產生的內容可能不正確。

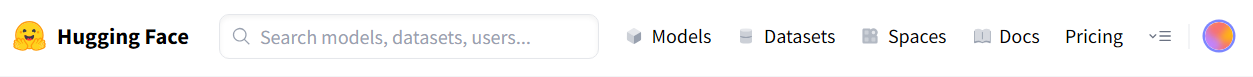
圖：忘了暫存 token，可以按下右邊的功能鈕，再按下「Invalidate and refresh」連結，它會重新提供新的 token 給你

安裝套件

npm 網站連結：<https://www.npmjs.com/>

**npm i @huggingface/transformers @google/genai canvas --save**

註：@huggingface/transformers 套件**目前只支援 onnx 模型格式**。



圖：進入 Hugging Face 網站，按下中間的連結「Models」

一張含有 文字, 螢幕擷取畫面, 軟體, 網頁 的圖片

AI 產生的內容可能不正確。

圖：頁籤列表中，檢視「Libraries」

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

AI 產生的內容可能不正確。

圖：要同時勾選「ONNX」和「Transformers.js」，下載的模型資料夾才會包括 .onnx 格式的模型權重。

模型權重

在深度學習裡，模型權重（weights）代表神經網路在訓練過程中所得到的參數，這些參數可以調整模型學習的行為，並在學習的過程中取得有用的特徵（feature）以及對輸入資料進行理解與投射（project）。簡單來說，深度學習的模型由許多神經元（neuron）之間的連接所組成，而權重就是連接這些神經元之間的參數（值）。

表：模型權重格式簡介

|  |  |
| --- | --- |
| 模型權重格式 | 說明 |
| .bin | PyTorch 的序列化儲存格式，通常由 PyTorch/Huggingface 框架讀取居多，常見的名稱為 pytorch\_model.bin。 |
| .onnx | Open Neural Network eXchange，一種通用的模型交換格式，可以跨框架（例如 PyTorch、TensorFlow、Caffe 等等）來讀取與使用，常見的名稱為 model.onnx。 |
| .safetensors | 一種較安全的模型權重儲存格式，一般模型在讀入程式的過程中，序列化資料可能會被竄改，而此格式會直接被模型讀取，驗證權重資料的完整性，避免被惡意修改的可能，常見的名稱為 model.safetensors。 |

註：<https://547labrecord.blogspot.com/2025/02/blog-post.html>

Pipeline

pipeline 方便我們在使用 Hugging Face 所下載的模型時，用最簡潔的方式來完成任務（tasks）。

參考連結：<https://huggingface.co/docs/transformers.js/api/pipelines>

## 電腦視覺

電腦視覺（Computer Vision）是 AI 的一個領域，讓電腦能「看懂」圖片或影片的內容。它的目標是模擬人類視覺，從影像中辨識物體、分析場景、追蹤動作，並做出對應的判斷。常見應用包括人臉辨識、自動駕駛、醫療影像診斷與工業檢測。

|  |
| --- |
| 1-1.js |
| /\*\*   \* 圖片特徵擷取與相似度計算   \*   \* 參考模型: https://huggingface.co/nomic-ai/nomic-embed-vision-v1.5   \*/  // 匯入模組  import { pipeline } from '@huggingface/transformers';  // 建立圖片特徵擷取管道  const pipe = await pipeline('image-feature-extraction', 'nomic-ai/nomic-embed-vision-v1.5', {      dtype: 'auto', // dtype: auto, fp32, fp16, q8, int8, uint8, q4, bnb4, q4f16  });  // 定義圖片路徑  const imagePath = [      './images/1-1\_0.jpg', // Image 0      './images/1-1\_1.jpg', // Image 1      './images/1-1\_2.jpg', // Image 2      './images/1-1\_3.jpg', // Image 3  ];  // 計算 Cosine Similarity  function cos\_sim(main\_vector, other\_vector) {      /\*\*       \* 計算兩個向量的餘弦相似度       \* @param {Array} main\_vector - 主向量       \* @param {Array} other\_vector - 其他向量       \* @returns {number} - 餘弦相似度       \*/      // 公式:      // cos(θ) = A·B / (||A|| \* ||B||)      // 其中 A·B 是兩個向量的點積，||A|| 和 ||B|| 是兩個向量的模長。      // 這裡的點積是指兩個向量的內積，      // 進階討論:      // 有沒有什麼額外的方法，讓兩個向量直接計算點積就能得到餘弦相似度，藉此簡化這個公式？      // 檢查向量長度是否一致      if (main\_vector.length !== other\_vector.length) {          throw new Error('向量維度不一致，無法計算餘弦相似度');      }      // 計算點積和內積      let dot\_product = 0;      let inner\_product\_A = 0;      let inner\_product\_B = 0;      // 向量計算      for (let i = 0; i < main\_vector.length; i++) {          dot\_product += main\_vector[i] \* other\_vector[i];          inner\_product\_A += main\_vector[i] \* main\_vector[i];          inner\_product\_B += other\_vector[i] \* other\_vector[i];      }      // 計算餘弦相似度      return dot\_product / (Math.sqrt(inner\_product\_A) \* Math.sqrt(inner\_product\_B));  }  // 指定圖片的特徵 (zero-based index)  let index = 0; // 這裡可以修改為 0, 1, 2, 3 等，代表要用來比較的圖片索引  const main\_vector = await pipe(imagePath[index]);  // 檢視變數內容  // console.log(main\_vector);  // 計算其他圖片與第 1 張的相似度  for (let i = 0; i < imagePath.length; i++) {      // 跳過用來比較的圖片索引      if (i === index) {          continue;      }      // 獲取第 i 張圖片的特徵 (不包含第 1 張)      const other\_vector = await pipe(imagePath[i]);      // 計算第 1 張與其它張圖片的餘弦相似度      const similarity = cos\_sim(main\_vector.ort\_tensor.cpuData, other\_vector.ort\_tensor.cpuData);      // 輸出相似度結果      console.log(`Image ${index} 與 Image ${i} 相似度: ${similarity.toFixed(4)}`);  } |

|  |
| --- |
| 1-2.js |
| /\*\*   \* 物件偵測   \*   \* 參考模型: https://huggingface.co/Xenova/detr-resnet-50   \*/  // 匯入模組  import { pipeline } from '@huggingface/transformers';  import { writeFileSync } from 'fs';  import { registerFont, loadImage, createCanvas } from 'canvas';  // 註冊字型  // 注意：如果要顯示中文，請確保字型支援中文  registerFont('./fonts/NotoSansTC-Regular.ttf', { family: 'Noto Sans TC' });  // 建立物件偵測管道  const pipe = await pipeline('object-detection', 'Xenova/detr-resnet-50', { dtype: 'auto' });  // 定義圖片路徑  const imagePath = "./images/1-2\_0.jpg"; // 單一圖片路徑  // 執行物件偵測  const results = await pipe(imagePath);  // 檢視變數內容  // console.log(results);  // 顯示結果  console.log("物件偵測結果:");  results.forEach((result, index) => {      console.log(`物件 ${index + 1}:`);      console.log(`  類別: ${result.label}`);      console.log(`  信心度: ${result.score}`);      console.log(`  位置: [${result.box.xmin}, ${result.box.ymin}, ${result.box.xmax}, ${result.box.ymax}]`);  });  // 將偵測結果繪製到圖片上  const drawResultsOnImage = async (imagePath, results) => {      // 載入圖片並建立畫布      const image = await loadImage(imagePath);      const canvas = createCanvas(image.width, image.height);      const ctx = canvas.getContext('2d');      // 繪製原始圖片      ctx.drawImage(image, 0, 0);      // 設定字型      ctx.font = '20px "Arial"'; // 如果你有註冊中文字型，可以改成 '20px "Noto Sans TC"'      ctx.textBaseline = 'top'; // 設定文字基線為頂部，這樣文字會從邊框上方開始繪製      // 繪製每個偵測結果      results.forEach(result => {          const { xmin, ymin, xmax, ymax } = result.box;          // 繪製紅色邊框          ctx.strokeStyle = 'red';          ctx.lineWidth = 2;          ctx.strokeRect(xmin, ymin, xmax - xmin, ymax - ymin);          // 標註文字（白底紅字提升可讀性）          const label = `${result.label} (${(result.score \* 100).toFixed(4)}%)`;          const textWidth = ctx.measureText(label).width;          // 背景白框          ctx.fillStyle = 'white';          ctx.fillRect(xmin, ymin - 22, textWidth + 6, 22);          // 文字          ctx.fillStyle = 'red';          ctx.fillText(label, xmin + 3, ymin - 20);      });      // 儲存圖片      const outputPath = './images/1-2\_0\_detected.jpg';      const buffer = canvas.toBuffer('image/jpeg'); // 希望格式是 png 就改成 'image/png'      writeFileSync(outputPath, buffer);      console.log(`結果圖片已儲存至: ${outputPath}`);  };  // 繪製結果到圖片上  await drawResultsOnImage(imagePath, results); |

|  |
| --- |
| 1-3.js |
| /\*\*   \* 圖片分類   \*   \* 參考模型: https://huggingface.co/Xenova/vit-base-patch16-224   \*/  // 匯入模組  import { pipeline } from '@huggingface/transformers';  // 建立圖片分類管道  const pipe = await pipeline('image-classification', 'Xenova/vit-base-patch16-224', { dtype: 'auto' });  // 定義圖片路徑  const imagePath = './images/1-3\_0.jpg'; // 單一圖片路徑  // 執行圖片分類  const result = await pipe(imagePath);  // 檢視變數內容  // console.log(result);  // 顯示分類結果  result.forEach((item) => {      console.log(`Label: ${item.label}, Score: ${item.score.toFixed(4)}`);  }); |

## （二）自然語言處理

自然語言處理（Natural Language Processing, NLP）是人工智慧的一個分支，目的是讓電腦能理解、分析、生成人類語言。它結合語言學與機器學習，用於文字分類、情感分析、翻譯、問答系統與聊天機器人等應用，幫助電腦「讀懂」和「說出」自然語言。

|  |
| --- |
| 2-1.js |
| /\*\*   \* 文字特徵擷取與相似度計算   \*   \* 參考模型: https://huggingface.co/Xenova/bge-m3   \*/  // 匯入模組  import { pipeline, AutoTokenizer } from '@huggingface/transformers';  // 定義模型ID  const model\_id = 'Xenova/bge-m3';  // 初始化斷詞器  const tokenizer = await AutoTokenizer.from\_pretrained(model\_id);  // 建立特徵擷取管道  const pipe = await pipeline('feature-extraction', model\_id, { dtype: 'auto' });  // 定義要處理的文字  const text1 = '感謝你的幫忙，這對我來說非常重要。';  const text2 = '謝謝你的協助，這對我來說意義重大。';  // 執行特徵擷取  const features1 = await pipe(text1);  const features2 = await pipe(text2);  // 檢視變數內容  console.log(features1);  console.log(features2);  // 檢視斷詞的列表  console.log("第 1 句的斷詞:");  console.log(tokenizer.tokenize(text1, { add\_special\_tokens: true }));  console.log("第 2 句的斷詞:");  console.log(tokenizer.tokenize(text2, { add\_special\_tokens: true }));  // 平均池化函式  function mean\_pooling(vector, dims) {      /\*\*       \* 將向量進行平均池化       \* @param {Float32Array} vector - 輸入向量       \* @param {Array} dims - 向量維度       \* @returns {Float32Array} - 平均池化後的向量       \*/      // 例如 vector: Float32Array(13312) [0.08195222169160843,    0.7124125361442566,  -0.9786397218704224, ... 13212 more items]      // 例如 dims: [ 1, 13, 1024 ]      const [batch, tokens, dim] = dims;      // 將向量展開、攤平為一維陣列。目前 pooled 的內容是空的 Float32Array      const pooled = new Float32Array(dim);      // 檢查向量維度 (把原本攤平的 word embedding，重新組成 tokens[i] x dim 的矩陣)      for (let i = 0; i < tokens; i++) {          for (let j = 0; j < dim; j++) {              pooled[j] = pooled[j] + vector[i \* dim + j]; // 可以寫成 pooled[j] += vector[i \* dim + j];          }      }      /\*\*       \* 這裡的 vector[i \* dim + j] 是指第 i 個 token 在第 j 個特徵維度的值。       \* 這樣的計算方式可以將每個 token 在每個特徵維度的值累加到 pooled 中。       \* 最終 pooled[j] 將包含所有 tokens 在第 j 個特徵維度的總和。       \* 這樣的累加方式可以確保每個特徵維度的值都被正確計算。       \* 例如 pooled[0] = vector[0 \* dim + 0] + vector[1 \* dim + 0] + ... + vector[(tokens - 1) \* dim + 0]       \* 例如 pooled[1] = vector[0 \* dim + 1] + vector[1 \* dim + 1] + ... + vector[(tokens - 1) \* dim + 1]       \* 這樣 pooled[0] 就是所有 tokens 在第 0 個特徵維度的總和。       \* 而 pooled[1] 就是所有 tokens 在第 1 個特徵維度的總和。       \* 以此類推，直到計算出 pooled[dim - 1]。       \*/      // 計算平均值，將 pooled 中的每個特徵維度除以 tokens 的數量，這樣可以得到每個特徵維度的平均值。      for (let j = 0; j < dim; j++) {          pooled[j] = pooled[j] / tokens; // 也可以寫成: pooled[j] /= tokens;      }      // 最終 pooled 將包含每個特徵維度的平均值，這樣就完成了平均池化的操作。      // 例如 pooled: Float32Array(1024) [0.08195222169160843, 0.7124125361442566, -0.9786397218704224, ... 1021 more items]      // 這樣 pooled 就是每個特徵維度的平均值，      return pooled;  }  // 計算 Cosine Similarity  function cos\_sim(main\_vector, other\_vector) {      /\*\*       \* 計算兩個向量的餘弦相似度       \* @param {Array} main\_vector - 主向量       \* @param {Array} other\_vector - 其他向量       \* @returns {number} - 餘弦相似度       \*/      // 檢查向量長度是否一致      if (main\_vector.length !== other\_vector.length) {          throw new Error('向量維度不一致，無法計算餘弦相似度');      }      // 計算點積和內積      let dot\_product = 0;      let inner\_product\_A = 0;      let inner\_product\_B = 0;      // 向量計算      for (let i = 0; i < main\_vector.length; i++) {          dot\_product += main\_vector[i] \* other\_vector[i];          inner\_product\_A += main\_vector[i] \* main\_vector[i];          inner\_product\_B += other\_vector[i] \* other\_vector[i];      }      // 計算餘弦相似度      return dot\_product / (Math.sqrt(inner\_product\_A) \* Math.sqrt(inner\_product\_B));  }  // 進行 mean pooling  const pooled1 = mean\_pooling(features1.ort\_tensor.cpuData, features1.ort\_tensor.dims);  const pooled2 = mean\_pooling(features2.ort\_tensor.cpuData, features2.ort\_tensor.dims);  // 計算相似度  const similarity = cos\_sim(pooled1, pooled2);  // 輸出結果  console.log(`第 1 句和第 2 句的餘弦相似度為: ${similarity}`); |

|  |
| --- |
| 2-2.js |
| /\*\*   \* 文字分類   \*   \* 參考模型: https://huggingface.co/Xenova/bert-base-multilingual-uncased-sentiment   \*/  // 匯入模組  import { pipeline } from '@huggingface/transformers';  // 定義模型 ID  const model\_id = 'Xenova/bert-base-multilingual-uncased-sentiment';  // 初始化分類管道  const pipe = await pipeline('sentiment-analysis', model\_id, { dtype: 'auto' });  // 定義要分類的文字  const arr\_text = [      'That is a great movie!',      'I am not happy with the service.',      'The weather is terrible today.',      'I love the new design of the product.',      '今天的天氣真好！',      '這部電影讓我感到很失望。'  ];  // 執行分類  const results = await pipe(arr\_text);  // 檢視變數內容  // console.log(results);  // 輸出結果  for (let i = 0; i < arr\_text.length; i++) {      console.log(`文字: ${arr\_text[i]}`);      console.log(`分類: ${results[i].label}`);      console.log(`分數: ${results[i].score.toFixed(4)}`);      console.log('-------------------------');  } |

|  |
| --- |
| 2-3.js |
| /\*\*   \* 情感分析   \*   \* 參考模型: https://huggingface.co/Xenova/distilroberta-finetuned-financial-news-sentiment-analysis   \*/  // 匯入模組  import { pipeline } from '@huggingface/transformers';  // 定義模型 ID  const model\_id = 'Xenova/distilroberta-finetuned-financial-news-sentiment-analysis';  // 初始化分類管道  const pipe = await pipeline('sentiment-analysis', model\_id, { dtype: 'auto' });  // 定義要分類的文字  const arr\_text = [      'The company reported a profit increase in the last quarter.',      'Stock prices fell sharply after the announcement.',      'Investors are optimistic about the new product launch.',      'The economic outlook remains uncertain amid rising inflation.',      'I have not decided whether to invest in this stock yet.',      'The market reacted negatively to the CEO\'s resignation.',      'My family and I are planning a vacation next month.',  ];  // 執行分類  const results = await pipe(arr\_text);  // 檢視變數內容  // console.log(results);  // 輸出結果  for (let i = 0; i < arr\_text.length; i++) {      console.log(`文字: ${arr\_text[i]}`);      console.log(`分類: ${results[i].label}`);      console.log(`分數: ${results[i].score.toFixed(4)}`);      console.log('-------------------------');  } |

|  |
| --- |
| 2-4.js |
| /\*\*   \* 文字生成   \*   \* 參考模型: https://huggingface.co/onnx-community/Llama-3.2-1B-Instruct   \*/  // 匯入模組  import { pipeline } from '@huggingface/transformers';  // 定義模型 ID  const model\_id = 'onnx-community/Llama-3.2-1B-Instruct';  // 初始化生成管道  const pipe = await pipeline('text-generation', model\_id, { dtype: 'auto' });  // 定義要生成的訊息  const messages = [      {"role": "system", "content": "You are a helpful assistant."},      {"role": "user", "content": "What is the capital of France?"}  ];  // 執行生成  const results = await pipe(messages, {      max\_new\_tokens: 100,      do\_sample: true,      temperature: 0.7,      top\_k: 50,      top\_p: 0.95  });  // 檢視變數內容  // console.log(results);  // console.log(results[0].generated\_text);  // 輸出結果  console.log('生成的文字:');  const arr\_text = results[0].generated\_text;  // 取得最後一個生成的文字  console.log(arr\_text[arr\_text.length - 1].content); |

|  |
| --- |
| 2-5.js |
| /\*\*   \* 翻譯   \*   \* 參考模型: https://huggingface.co/Xenova/nllb-200-distilled-600M   \* 語系代碼: https://github.com/facebookresearch/flores/blob/main/flores200/README.md#languages-in-flores-200   \*/  // 匯入模組  import { pipeline } from '@huggingface/transformers';  // 定義模型 ID  const model\_id = 'Xenova/nllb-200-distilled-600M';  // 初始化翻譯管道  const pipe = await pipeline('translation', model\_id, { dtype: 'auto' });  // 定義要翻譯的訊息  const results = await pipe('Hello, how is it going today?', {      src\_lang: 'eng\_Latn', // 原始語言設定為英文      tgt\_lang: 'zho\_Hant', // 目標語言設定為 Chinese (Traditional)  });  // 檢視變數內容  console.log(results);  // 輸出翻譯結果  console.log(results[0].translation\_text); |

## （三）語音處理

## （四）知識挖掘

知識挖掘（Knowledge Mining）是從大量資料中萃取有價值資訊與知識的過程，結合自然語言處理、機器學習與搜尋技術，將非結構化資料（如文件、圖片、報告）轉化為可查詢、可理解的知識。常用於智慧客服、企業知識管理、研究分析等場景。

## （五）智慧文件

## （六）生成式 AI

生成式 AI（Generative AI，GenAI）是指能夠根據輸入資料，自動產生文字、圖片、音樂、程式碼等新內容的人工智慧技術。它透過訓練大量資料學習模式，進而創造出具原創性且類似人類創作的內容。常見應用包括聊天機器人、圖像生成、文章撰寫與影音合成等。