

## 目次

一、建立 nvm 環境.....	1
二、安裝 nodejs 與版本切換 .....	5
三、AI 基礎概論與演進 .....	7
四、AI 應用概論.....	10
(一) 電腦視覺.....	14
(二) 自然語言處理.....	20
(三) 語音處理.....	28
(四) 知識挖掘.....	40
(五) 智慧文件處理.....	52
四、生成式 AI (Optional) .....	57

課程範例網址：

[https://github.com/telunyang/nodejs ai basics](https://github.com/telunyang/nodejs_ai_basics)

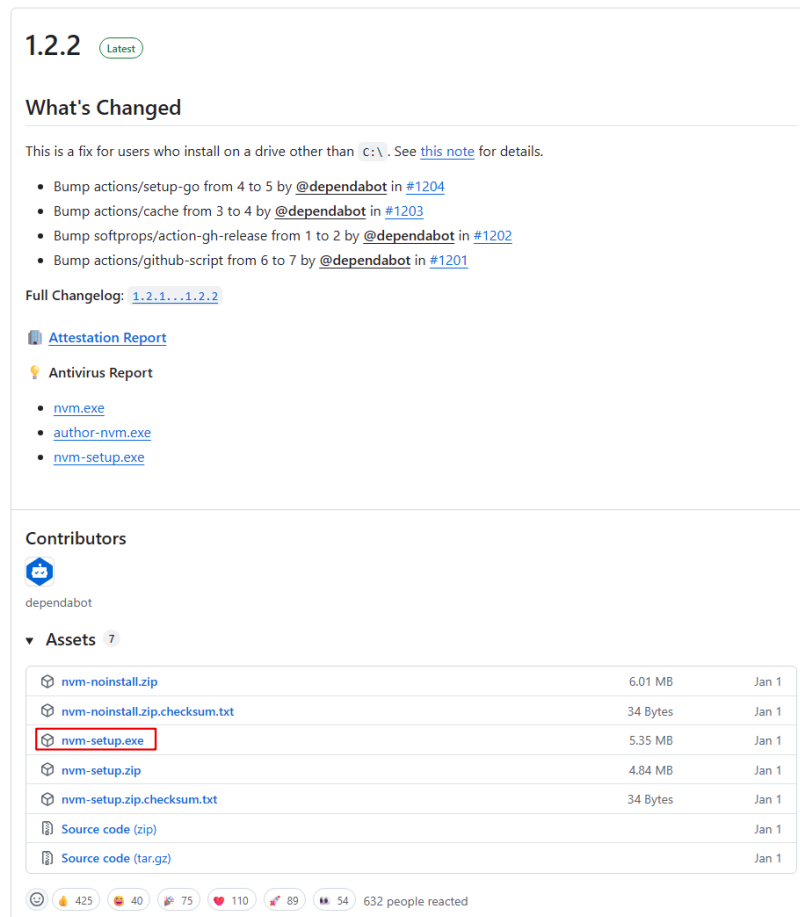
## 一、建立 nvm 環境

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：



1.2.2 Latest

### What's Changed

This is a fix for users who install on a drive other than `C:\`. See [this note](#) for details.

- Bump actions/setup-go from 4 to 5 by [@dependabot](#) in [#1204](#)
- Bump actions/cache from 3 to 4 by [@dependabot](#) in [#1203](#)
- Bump softprops/action-gh-release from 1 to 2 by [@dependabot](#) in [#1202](#)
- Bump actions/github-script from 6 to 7 by [@dependabot](#) in [#1201](#)

Full Changelog: [1.2.1...1.2.2](#)

[Attestation Report](#)

[Antivirus Report](#)

- [nvm.exe](#)
- [author-nvm.exe](#)
- [nvm-setup.exe](#)

### Contributors

[dependabot](#)

▼ Assets 7

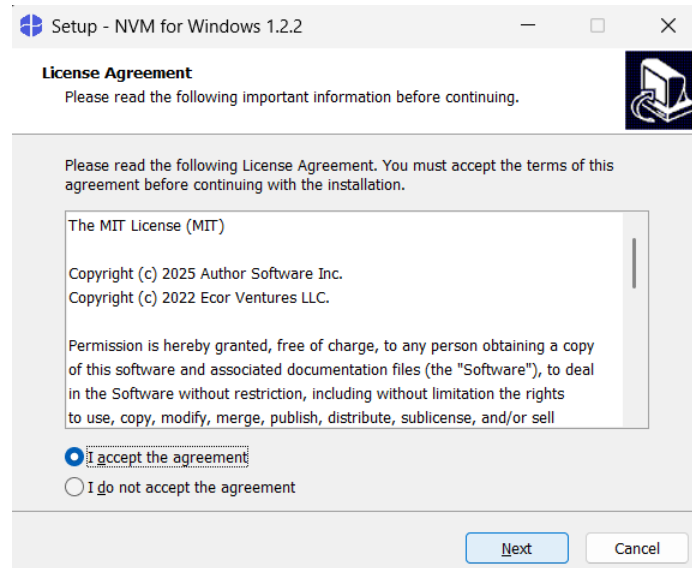
<a href="#">nvm-noinstall.zip</a>	6.01 MB	Jan 1
<a href="#">nvm-noinstall.zip.checksum.txt</a>	34 Bytes	Jan 1
<a href="#">nvm-setup.exe</a>	5.35 MB	Jan 1
<a href="#">nvm-setup.zip</a>	4.84 MB	Jan 1
<a href="#">nvm-setup.zip.checksum.txt</a>	34 Bytes	Jan 1
<a href="#">Source code (zip)</a>		Jan 1
<a href="#">Source code (tar.gz)</a>		Jan 1

👍 425 🍌 40 🗨️ 75 ❤️ 110 🚩 89 🏷️ 54 632 people reacted

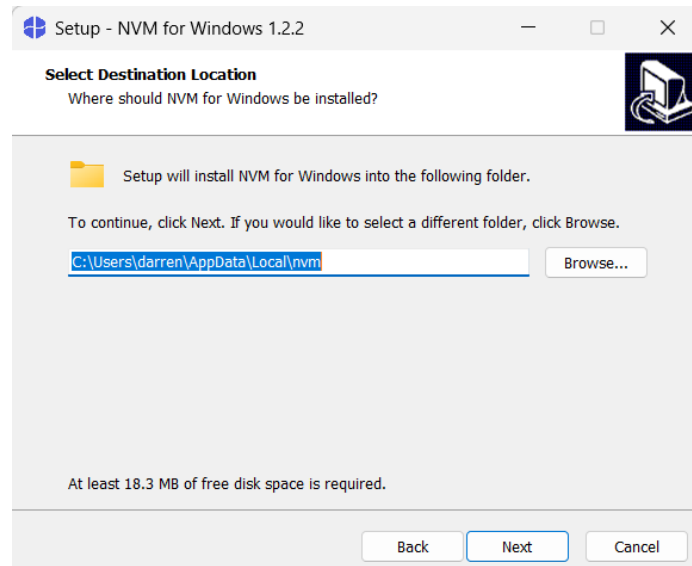
圖：按下「nvm-setup.exe」，下載安裝檔



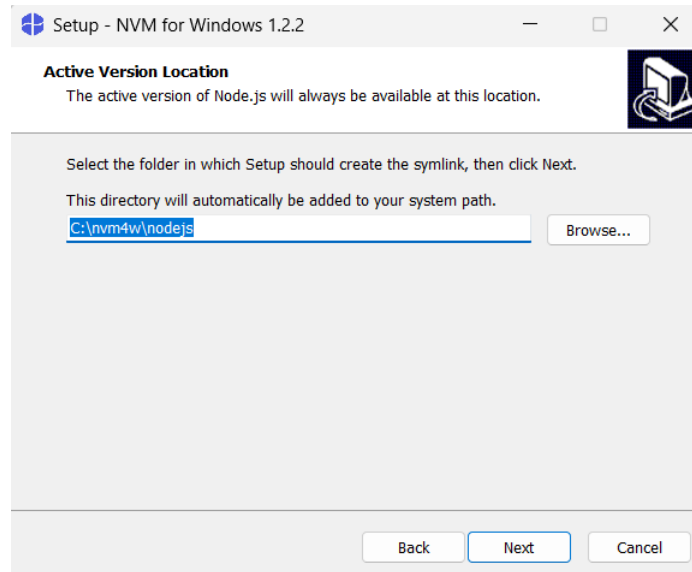
圖：下載後的安裝檔圖示，請連續點兩下  
(出現「您是否要允許此 App 變更您的裝置」，請按下是)



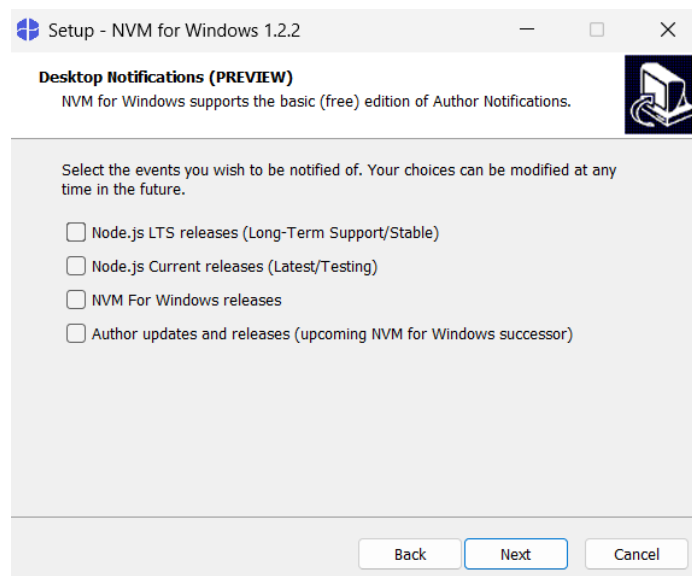
圖：選擇「I accept the agreement」，再按下「Next」



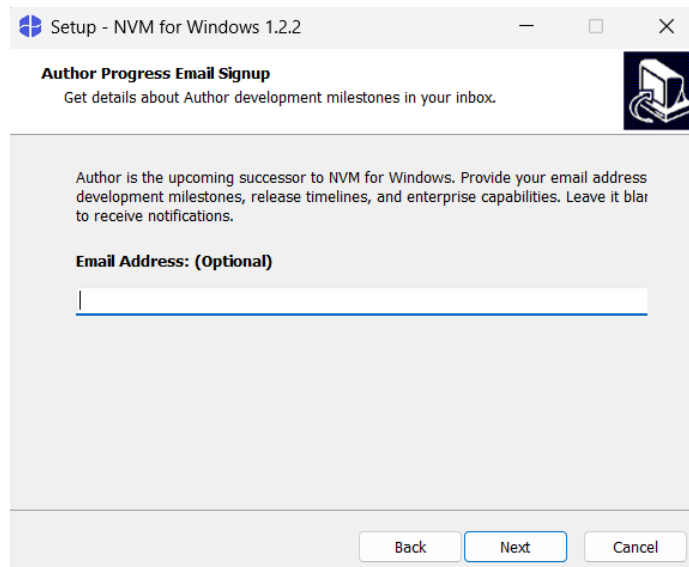
圖：安裝路徑自由切換，強烈建議路徑不要有 multi-bytes 的文字（例如中文、日文、韓文等非英語系文字），之後請按下「Next」



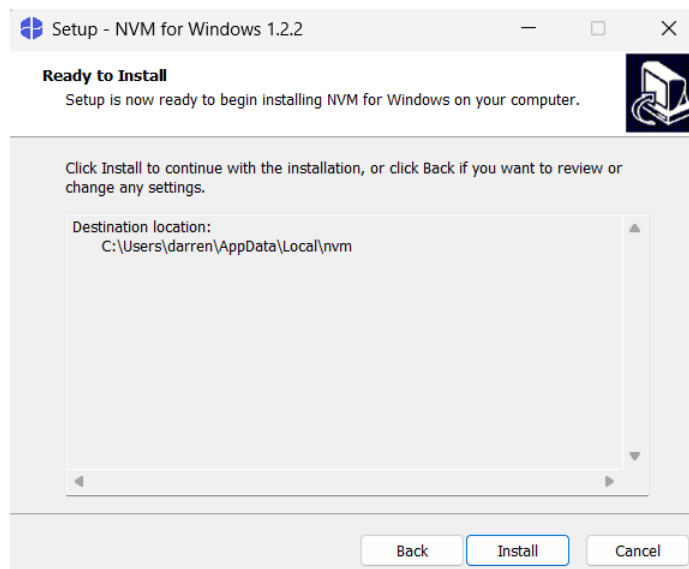
圖：放置主要 node.js 版本的目錄，會被加到系統路徑，之後請按下「Next」



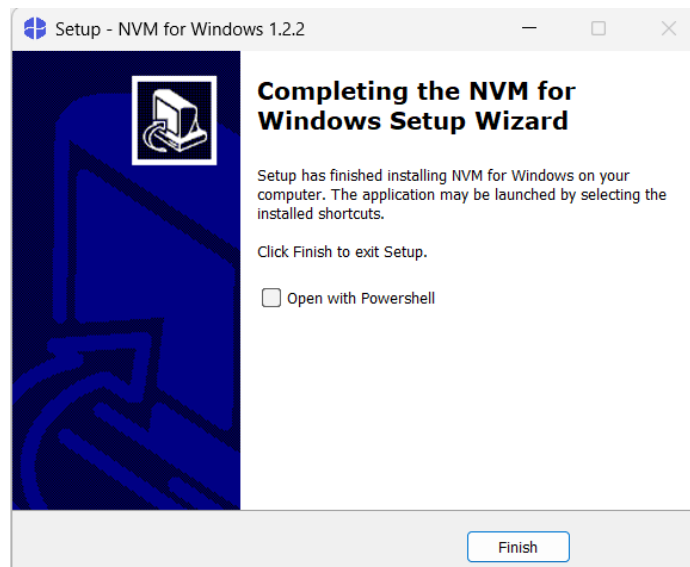
圖：全部取消勾選（可依需求保留），按下「Next」



圖：不用填寫（欲關注開發團隊動向除外），直接按下「Next」



圖：確認配置沒問題，按下「Install」，開始安裝



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

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [版本 10.0.26100.4652]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。

D:\teach\nodejs_ai_basics>nvm

Running version 1.2.2.

Usage:

nvm arch                : Show if node is running in 32 or 64 bit mode.
nvm current              : Display active version.
nvm debug                : Check the NVM4W process for known problems (troubleshooter).
nvm install <version> [arch] : The version can be a specific version, "latest" for the latest current version, or "lts" for the
                             most recent LTS version. Optionally specify whether to install the 32 or 64 bit version (defaults
                             to system arch). Set [arch] to "all" to install 32 AND 64 bit versions.
                             Add --insecure to the end of this command to bypass SSL validation of the remote download server.
nvm list [available]      : List the node.js installations. Type "available" at the end to see what can be installed. Aliased as ls.
nvm on                    : Enable node.js version management.
nvm off                   : Disable node.js version management.
nvm proxy [url]           : Set a proxy to use for downloads. Leave [url] blank to see the current proxy.
                             Set [url] to "none" to remove the proxy.
nvm node_mirror [url]     : Set the node mirror. Defaults to https://nodejs.org/dist/. Leave [url] blank to use default url.
nvm npm_mirror [url]      : Set the npm mirror. Defaults to https://github.com/npm/cli/archive/. Leave [url] blank to default url.
nvm uninstall <version>   : The version must be a specific version.
nvm upgrade                : Update nvm to the latest version. Manual rollback available for 7 days after upgrade.
nvm use [version] [arch]  : Switch to use the specified version. Optionally use "latest", "lts", or "newest".
                             "newest" is the latest installed version. Optionally specify 32/64bit architecture.
                             nvm use <arch> will continue using the selected version, but switch to 32/64 bit mode.
nvm reinstall <version>   : A shortcut method to clean and reinstall a specific version.
nvm root [path]           : Set the directory where nvm should store different versions of node.js.
                             If <path> is not set, the current root will be displayed.
nvm subscribe [--]<topic> : Subscribe to desktop notifications.
                             Valid topics: lts, current, nvm4w, author
nvm unsubscribe [--]<topic> : Unsubscribe from desktop notifications.
                             Valid topics: lts, current, nvm4w, author
nvm [--]version           : Displays the current running version of nvm for Windows. Aliased as v.

D:\teach\nodejs_ai_basics>

```

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

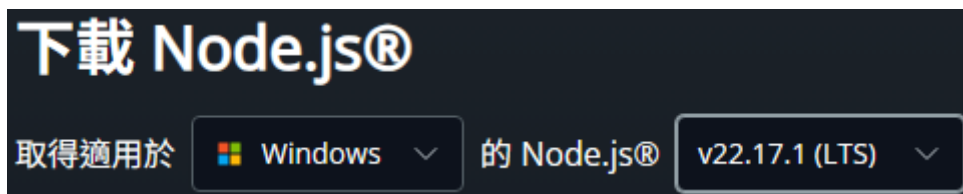
## 二、安裝 nodejs 與版本切換

常用終端機指令	說明
nvm	觀看指令用法

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

```
D:\teach\nodejs_ai_basics>nvm list
No installations recognized.
```

圖：列出所有安裝的 node.js 版本，目前尚未安裝



圖：到 node.js 官方網站檢視當前長期支援版本

```
D:\teach\nodejs_ai_basics>nvm install lts
Downloading node.js version 22.17.1 (64-bit)...
Extracting node and npm...
Complete
Installation complete.
If you want to use this version, type:

nvm use 22.17.1
```

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)

```
D:\teach\nodejs_ai_basics>nvm use 22.17.1
Now using node v22.17.1 (64-bit)
```

圖：確認是否切換到當前安裝的版本

```
D:\teach\nodejs_ai_basics>nvm list
* 22.17.1 (Currently using 64-bit executable)
```

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）

```
D:\teach\nodejs_ai_basics>nvm uninstall 22.17.1
Uninstalling node v22.17.1... done
```

圖：反安裝指定的版本（有需要再執行這個指令，上課先別刪...）

```
D:\teach\nodejs_ai_basics>node -v
v22.17.1
```

圖：確認目前的 node.js 版本

```
D:\teach\nodejs_ai_basics>npm -v
10.9.2
```

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：

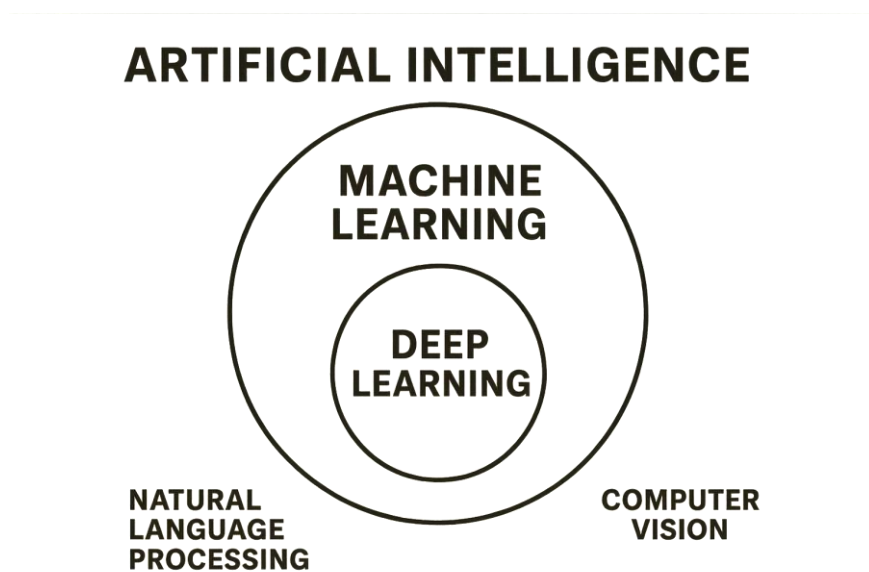
- 最外層：Artificial Intelligence
  - 表示 AI 的整體範疇，涵蓋所有智慧化技術與應用。



- 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。



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。

時間	里程碑與事件
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	<p>專家系統與符號 AI 盛行 (如 Dendral、ELIZA、Shakey 機器人)。</p> <p>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 &amp; 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。</p> <p>第二次 AI 冬天 (Second AI Winter): 約 1987 – 1993。</p> <p>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。</p>
1980 年代後期	Jude 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。

時間	里程碑與事件
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>

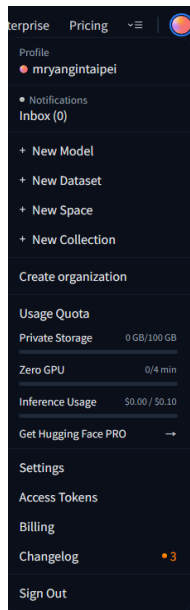
## 四、AI 應用概論

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。

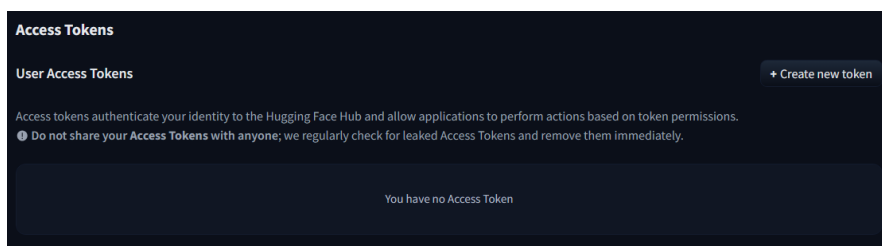
取得 Hugging Face 的 access token（非必要）

連結：<https://huggingface.co/>

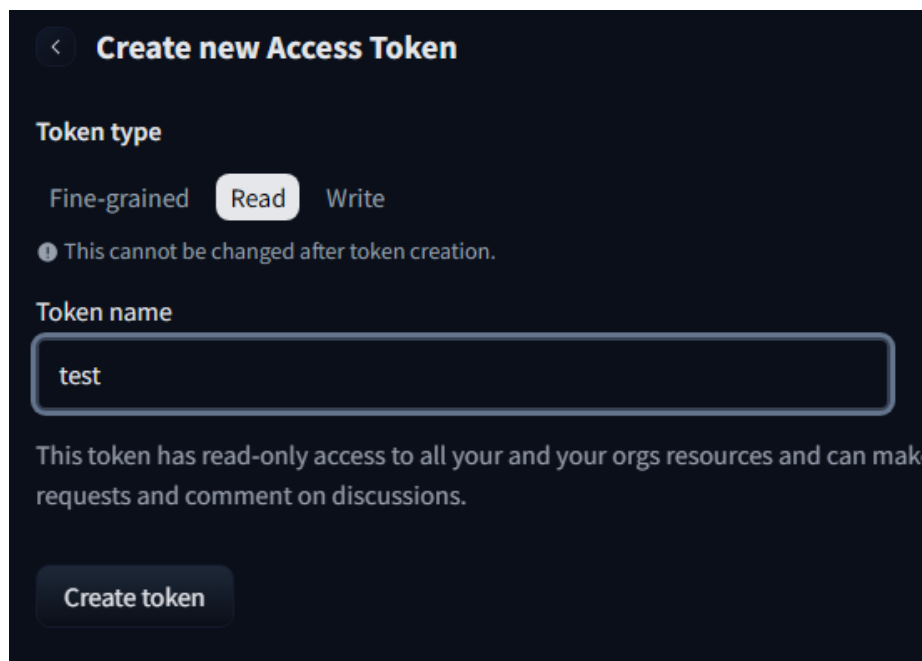
備註：請先註冊帳號（課程結束後，沒有其它用途的話，強烈建議一律登出）。



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

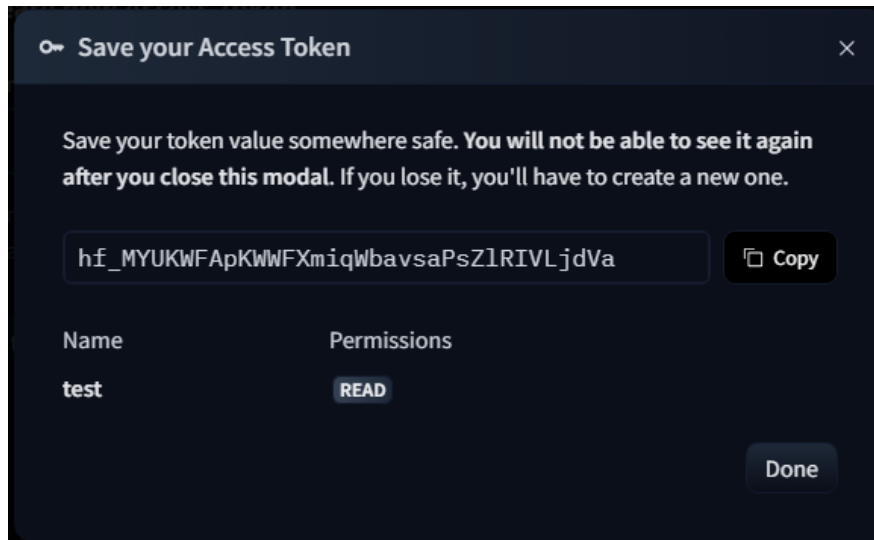


圖：尚未建立 token 的話，按下右上角「Create new token」

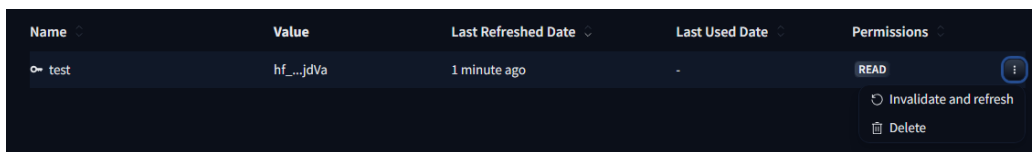


圖：因為我們只是希望下載、讀取開放權重的模型，所以 Token type 選擇「Read」，Token name 可以任意填寫，例如「test」，之後按下「Create

token」



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



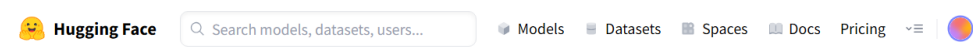
圖：忘了暫存 token，可以按下右邊的功能鈕，再按下「Invalidate and refresh」連結，它會重新提供新的 token 給你

#### 安裝套件

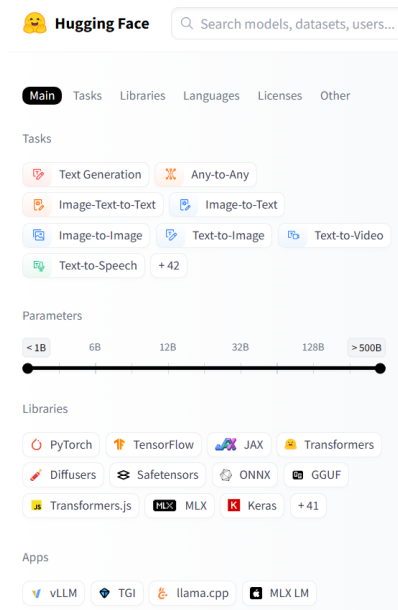
npm 網站連結：<https://www.npmjs.com/>

- `npm i @huggingface/transformers @google/genai canvas wavefile --save`
- `npm i --save` (確認目錄內有 package.json)

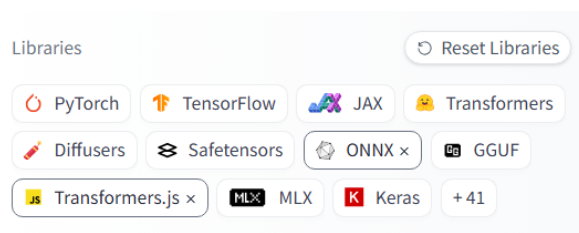
註：@huggingface/transformers 套件 目前只支援 onnx 模型格式。



圖：進入 Hugging Face 網站，按下中間的連結「Models」



圖：頁籤列表中，檢視「Libraries」



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>

## （一）電腦視覺

電腦視覺（Computer Vision）是 AI 的一個子領域，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
<pre> /**  * 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 }); </pre>

```

// 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：
  //  $\cos(\theta) = \frac{A \cdot B}{||A|| * ||B||}$ 
  // 其中  $A \cdot B$  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);

// 建立文字特徵擷取管道
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

語音處理是人工智慧的一個子領域，主要讓電腦能理解、產生或分析人類語音。常見應用包括語音辨識（將語音轉文字）、語音合成（文字轉語音）、語者識別與語音情緒分析等，廣泛應用於語音助理、客服系統與無障礙輔助技術中。

3-1.js

```
/**
 * 音訊特徵擷取與相似度計算
 *
 * 參考模型: https://huggingface.co/Xenova/clap-htsat-unfused
 * CLAP: Contrastive Language-Audio Pretraining
 */

// 匯入模組
import { AutoProcessor, ClapAudioModelWithProjection,
read_audio } from '@huggingface/transformers';
import wavefile from 'wavefile';
import { readFileSync } from 'fs';

// 定義模型 ID
const model_id = 'Xenova/clap-htsat-unfused';
```

```

// 初始化音訊特徵擷取管道
const processor = await AutoProcessor.from_pretrained(model_id);
const audio_model = await
ClapAudioModelWithProjection.from_pretrained(model_id, {
  dtype: 'auto',
  device: 'cpu'
});

// 建立函式
function processAudio(buffer) {
  // 將 Buffer 轉換為 WaveFile
  let wav = new wavefile.WaveFile(buffer);
  wav.toBitDepth('32f'); // 將輸入的音訊轉換為 32 位元浮點數格式
  (Float32)
  wav.toSampleRate(16000); // 轉換成 16kHz 的採樣率
  let audioData = wav.getSamples();

  // 如果音訊是多通道的，則將其轉換為單通道
  if (Array.isArray(audioData)) {
    if (audioData.length > 1) {
      const SCALING_FACTOR = Math.sqrt(2);

      // 合併頻道來節省記憶體
      for (let i = 0; i < audioData[0].length; ++i) {
        audioData[0][i] = SCALING_FACTOR * (audioData[0][i] +
audioData[1][i]) / 2;
      }
    }

    // 選擇第一個頻道的音訊數據
    audioData = audioData[0];
  }

  return audioData;
}

// 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));
}

// 定義音訊路徑
const audioPath = [
    './audios/3-1_0.wav', // Audio 0
    './audios/3-1_1.wav', // Audio 1
    './audios/3-1_2.wav', // Audio 2
    './audios/3-1_3.wav', // Audio 3
];

```

```

// 如果需要從網路下載語音檔案，可以使用以下程式碼
// let buffer = Buffer.from(await fetch(url).then(x =>
x.arrayBuffer()))

// 指定音訊的特徵 (zero-based index)
let index = 0; // 這裡可以修改為 0, 1, 2 等，代表要用來比較的音訊索引

// 如果語音檔案已經存在於本地，可以直接讀取
let buffer_main = readFileSync(audioPath[index]);

// 讀取音訊檔案
let audio_main = await processAudio(buffer_main);

// 將音訊數據轉換為模型輸入格式
const audio_inputs_main = await processor(audio_main);

// 取得音訊特徵
const audio_embeds_main = await audio_model(audio_inputs_main);

// 檢視變數內容
// console.log(audio_embeds_main)

for (let i = 0; i < audioPath.length; i++) {
  // 跳過用來比較的音訊索引
  if (i === index) {
    continue;
  }

  // 讀取第 i 張音訊檔案
  let buffer_other = readFileSync(audioPath[i]);
  let audio_other = await processAudio(buffer_other);

  // 將音訊數據轉換為模型輸入格式
  const audio_inputs_other = await processor(audio_other);

  // 取得音訊特徵
  const audio_embeds_other = await audio_model(audio_inputs_other);

```



```

// 計算第 1 張與其它張音訊的餘弦相似度
const similarity = cos_sim(
    audio_embeds_main.audio_embeds.ort_tensor.cpuData,
    audio_embeds_other.audio_embeds.ort_tensor.cpuData
);

// 輸出相似度結果
console.log(`Audio ${index} 與 Audio ${i} 的相似度:
${similarity}`);
}

```

### 3-2.js

```

/**
 * 語音轉文字
 *
 * 參考模型: https://huggingface.co/Xenova/whisper-medium
 * 語音測試網址: https://translate.google.com/translate\_tts?ie=UTF-8&client=tw-ob&tl=zh-TW&q=你的自訂文字
 * 範例: 如果說再見...是妳唯一的消息...我彷彿可以預見我自己...越往遠處飛去...妳越在我心裡...而我卻是妳不要的回憶
 */

// 匯入模組
import { pipeline } from '@huggingface/transformers';
import wavefile from 'wavefile';
import { readFileSync } from 'fs';

// 定義模型 ID
const model_id = 'Xenova/whisper-medium';

// 初始化語音轉文字管道 (如果用 CPU 可能會很久, 建議使用 GPU)
const pipe = await pipeline('automatic-speech-recognition', model_id, {
  dtype: 'auto',
  device: 'cpu'
});

```

```

// 取得語音檔案 (如果是 mp3，需要先轉換成 wav 格式)
const url = "./audios/3-2_0.wav";

// 如果需要從網路下載語音檔案，可以使用以下程式碼
// let buffer = Buffer.from(await fetch(url).then(x =>
x.arrayBuffer()))

// 如果語音檔案已經存在於本地，可以直接讀取
let buffer = readFileSync(url);

// 將 Buffer 轉換為 WaveFile
let wav = new wavefile.WaveFile(buffer);
wav.toBitDepth('32f'); // 將輸入的音訊轉換為 32 位元浮點數格式 (Float32)
wav.toSampleRate(16000); // 轉換成 16kHz 的採樣率，給 Whisper 模型使用
let audioData = wav.getSamples();

// 如果音訊是多通道的，則將其轉換為單通道
if (Array.isArray(audioData)) {
    if (audioData.length > 1) {
        const SCALING_FACTOR = Math.sqrt(2);

        // 合併頻道來節省記憶體
        for (let i = 0; i < audioData[0].length; ++i) {
            audioData[0][i] = SCALING_FACTOR * (audioData[0][i] +
audioData[1][i]) / 2;
        }
    }

    // 選擇第一個頻道的音訊數據
    audioData = audioData[0];
}

// 計算處理時間 - 開始計時
const start = Date.now();

/**
 * language 列表:

```

```

* ["english","chinese","german","spanish",
*  "russian","korean","french","japanese",
*  "portuguese","turkish","polish","catalan",
*  "dutch","arabic","swedish","italian",
*  "indonesian","hindi","finnish","vietnamese",
*  "hebrew","ukrainian","greek","malay","czech",
*  "romanian","danish","hungarian","tamil",
*  "norwegian","thai","urdu","croatian",
*  "bulgarian","lithuanian","latin","maori",
*  "malayalam","welsh","slovak","telugu",
*  "persian","latvian","bengali","serbian",
*  "azerbaijani","slovenian","kannada","estonian",
*  "macedonian","breton","basque","icelandic",
*  "armenian","nepali","mongolian","bosnian",
*  "kazakh","albanian","swahili","galician",
*  "marathi","punjabi","sinhala","khmer",
*  "shona","yoruba","somali","afrikaans",
*  "occitan","georgian","belarusian","tajik",
*  "sindhi","gujarati","amharic","yiddish",
*  "lao","uzbek","faroese","haitian creole",
*  "pashto","turkmen","nynorsk","maltese",
*  "sanskrit","luxembourgish","myanmar","tibetan",
*  "tagalog","malagasy","assamese","tatar",
*  "hawaiian","lingala","hausa","bashkir","javanese",
*  "sundanese"]
*/

// 取得語音轉文字的結果
let results = await pipe(audioData, {
  return_timestamps: true,    // 回傳時間戳記
  language: 'chinese'        // 輸出語系設定為中文
});

// 計算處理時間 - 結束計時
const end = Date.now();

// 計算總處理時間
const duration = end - start;

```

```

// 輸出處理時間
console.log(`處理時間: ${duration / 1000} 秒`);

// 檢視變數內容
console.log(results);

// 輸出語音轉文字的結果
console.log(`語音轉文字結果: ${results.text}`);

// 輸出 chunks 的內容
console.log('語音轉文字的 chunks:');
results.chunks.forEach((chunk, index) => {
    console.log('-----');
    console.log(`Chunk ${index + 1}:`);
    console.log(`[start: ${chunk.timestamp[0]}, end:
${chunk.timestamp[1]}] text: ${chunk.text}`);
});

```

### 3-3.js

```

/**
 * 文字轉音樂
 *
 * 參考模型: https://huggingface.co/Xenova/musicgen-small
 */

// 匯入模組
import { AutoTokenizer, MusicgenForConditionalGeneration, RawAudio }
from '@huggingface/transformers';

// 定義模型 ID
const model_id = 'Xenova/musicgen-small';

// 初始化模型
const tokenizer = await AutoTokenizer.from_pretrained(model_id);
const model = await
MusicgenForConditionalGeneration.from_pretrained(model_id, {

```

```

dtype: {
  text_encoder: 'q8', // 文本編碼器使用 8 位元整數
  decoder_model_merged: 'q8', // 解碼器合併使用 8 位元整數
  encodec_decode: 'fp32' // 編碼器解碼器使用 32 位元浮點數
},
device: 'cpu' // 使用 CPU
});

// 準備輸入文字
// const prompt = 'a light and cheerly EDM track, with syncopated
drums, aery pads, and strong emotions bpm: 130';
const prompt = 'lo-fi music with a soothing melody';
const inputs = await tokenizer(prompt);

// 計算處理時間 - 開始計時
const start = Date.now();

// 生成音樂
const audio_values = await model.generate({
  ...inputs,          // 「...」是展開運算子，將 inputs 物件的屬性展開為
                      // 函數參數
  do_sample: true,     // 使用隨機抽樣法
  temperature: 0.5,    // 溫度控制生成的隨機性
  max_new_tokens: 500, // 最大生成的 token 數量
  guidance_scale: 7.0, // 引導尺度，控制生成的多樣性，愈小愈多樣化，最小
                      // 值為 1.0
});

// 計算處理時間 - 結束計時
const end = Date.now();

// 計算總處理時間
const duration = end - start;

// 輸出處理時間
console.log(`處理時間: ${duration / 1000} 秒`);

// 儲存成 wav 檔案

```

```
const audio = new RawAudio(audio_values.data,  
model.config.audio_encoder.sampling_rate);  
audio.save('./audios/3-3_output.wav');
```

### 3-4.js

```
/**  
 * 文字轉語音  
 *  
 * 參考模型: https://huggingface.co/Xenova/speecht5\_tts  
 * 參考聲碼器: https://huggingface.co/Xenova/speecht5\_hifigan  
 */  
  
// ===== 簡易版本 =====  
import { pipeline } from '@huggingface/transformers';  
import wavefile from 'wavefile';  
import { writeFileSync } from 'fs';  
  
// 定義模型 ID  
const model_id = 'Xenova/speecht5_tts';  
  
// 初始化文字轉語音管道  
const pipe = await pipeline('text-to-speech', model_id, { dtype:  
'fp32' });  
  
// 取得發聲者嵌入 (這是預先訓練好的發聲者嵌入)  
const speaker_embeddings =  
'https://huggingface.co/datasets/Xenova/transformers.js-  
docs/resolve/main/speaker_embeddings.bin';  
  
// 建立輸入文本  
let input_text = `Hello, my name is Melody, it's a pleasure to meet  
you!  
Today, I will be your guide through the world of speech synthesis.`;  
  
// 生成語音  
const results = await pipe(input_text, { speaker_embeddings });
```

```

// 檢視變數內容
// console.log(results);

// 將生成的語音轉換為 WAV 格式
const wav = new wavefile.WaveFile();
wav.fromScratch(1, results.sampling_rate, '32f', results.audio);
writeFileSync('./audios/3-4_output.wav', wav.toBuffer());

// ===== 進階版本 =====

// 匯入模組
import {
    AutoTokenizer, AutoProcessor,
    SpeechT5ForTextToSpeech, SpeechT5HifiGan,
    Tensor
} from '@huggingface/transformers';
import wavefile from 'wavefile';
import { writeFileSync } from 'fs';

// 定義模型 ID
const model_id = 'Xenova/speecht5_tts';
const model_vocoder_id = 'Xenova/speecht5_hifigan';

// 讀取模型和處理器
// NOTE: 這裡使用的是 unquantized 版本，因為它們在精度上更好
// 如果需要更快的推理速度，可以考慮使用 quantized 版本
const tokenizer = await AutoTokenizer.from_pretrained(model_id);
const processor = await AutoProcessor.from_pretrained(model_id);

// 讀取模型和聲碼器
const model = await SpeechT5ForTextToSpeech.from_pretrained(model_id,
{
    quantized: false,
    dtype: 'auto'
});
const vocoder = await
SpeechT5HifiGan.from_pretrained(model_vocoder_id, {

```

```

    quantized: false,
    dtype: 'auto'
  });

  // 讀取發聲者嵌入（因為模型需要發聲者嵌入來生成語音）
  // 這裡使用的是一個預先訓練好的發聲者嵌入
  // 如果需要使用自定義的發聲者嵌入，可以參考模型的文檔來生成自己的發聲者嵌入
  const speaker_embeddings_data = new Float32Array(
    await (
      await fetch(
        'https://huggingface.co/datasets/Xenova/transformers.js-
docs/resolve/main/speaker_embeddings.bin'
      )
    ).arrayBuffer()
  );

  // 將發聲者嵌入轉換為 Tensor
  const speaker_embeddings = new Tensor(
    'float32',
    speaker_embeddings_data,
    [1, speaker_embeddings_data.length]
  )

  // 建立輸入文本
  let input_text = `Hello, my name is Melody, it's a pleasure to meet
you!
Today, I will be your guide through the world of speech synthesis.`;

  // 將輸入文本轉換為模型需要的格式，例如進行分詞和編碼
  const inputs = await tokenizer(input_text);

  // 生成語音（waveform 變數名稱是固定的）
  const { waveform } = await model.generate_speech(
    inputs['input_ids'],
    speaker_embeddings,
    { vocoder }
  );

```



```
// 檢查變數內容
// console.log(waveform)

// 將生成的語音轉換為 WAV 格式
const wav = new wavefile.WaveFile();
wav.fromScratch(
  1,
  processor.feature_extractor.config.sampling_rate,
  '32f',
  waveform.data
);
writeFileSync('./audios/3-4_output.wav', wav.toBuffer());
```

## （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。

在這裡會使用到向量資料庫 lancedb，網址如下：

[1] LanceDB JavaScript SDK

<https://www.npmjs.com/package/@lancedb/lancedb>

[2] lancedb

<https://github.com/lancedb/lancedb>

[3] LanceDB Documentation / Core Concepts

<https://lancedb.com/docs/concepts/>

範例使用的音效來源：

mixkit - Free assets for your next video project

<https://mixkit.co/>

4-1.js

```
/**
 * 建立向量資料庫
 *
```

```

* 參考資料：
* https://www.npmjs.com/package/@lancedb/lancedb
* https://lancedb.github.io/lancedb/basic/
* https://github.com/lancedb/lancedb
* https://lancedb.com/docs/concepts/
*
* Audio 來源：
* https://mixkit.co/
*
*/

// 匯入模組
import { AutoProcessor, ClapAudioModelWithProjection, pipeline } from
  '@huggingface/transformers';
import wavefile from 'wavefile';
import { readFileSync } from 'fs';
import * as lancedb from "@lancedb/lancedb";
import {
  Schema as ArrowSchema,
  Field,
  FixedSizeList,
  Int64,
  Utf8,
  Float32
} from 'apache-arrow';

// 建立文字特徵擷取管道
const pipe = await pipeline('feature-extraction', "Xenova/bge-m3",
  { dtype: 'auto' });

// 初始化音訊特徵擷取模型
const audio_processor = await
  AutoProcessor.from_pretrained("Xenova/clap-htsat-unfused");
const audio_model = await
  ClapAudioModelWithProjection.from_pretrained("Xenova/clap-htsat-
  unfused", {
    dtype: 'auto',
    device: 'cpu'
  })

```

```

});

// 建立函式
function processAudio(buffer) {
    // 將 Buffer 轉換為 WaveFile
    let wav = new wavefile.WaveFile(buffer);
    wav.toBitDepth('32f'); // 將輸入的音訊轉換為 32 位元浮點數格式 (Float32)
    wav.toSampleRate(16000); // 轉換成 16kHz 的採樣率
    let audioData = wav.getSamples();

    // 如果音訊是多通道的，則將其轉換為單通道
    if (Array.isArray(audioData)) {
        if (audioData.length > 1) {
            const SCALING_FACTOR = Math.sqrt(2);

            // 合併頻道來節省記憶體
            for (let i = 0; i < audioData[0].length; ++i) {
                audioData[0][i] = SCALING_FACTOR * (audioData[0][i] +
audioData[1][i]) / 2;
            }
        }

        // 選擇第一個頻道的音訊數據
        audioData = audioData[0];
    }

    return audioData;
}

// 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];
        }
    }

    // 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;
    }

    return pooled;
}

// 準備寫入向量資料庫的資料
const arr_data = [
    {
        'id': 1,
        'text_desc': '熱烈的掌聲，可以感受到現場觀眾的歡呼與慶賀，氣氛熱烈
歡騰，可能正在進行某種儀式或表演活動，人們正在為表演者或活動本身喝采。',
        'text_vector': null,
        'audio_path': './audios/3-1_0.wav',
        'audio_vector': null,
    }
];

```

```

    },
    {
      'id': 2,
      'text_desc': '清晰的掌聲，密集而響亮，充滿了現場的活力和興奮。可能正在進行表演、演講或其他活動，觀眾正以熱烈的掌聲表達支持、讚賞或歡呼的情緒，渲染現場氣氛。',
      'text_vector': null,
      'audio_path': './audios/3-1_1.wav',
      'audio_vector': null,
    },
    {
      'id': 3,
      'text_desc': '清澈悅耳的吉他聲，伴隨著悠揚的旋律。吉他的音色溫柔而略帶憂鬱，可能是一段輕柔的演奏或背景音樂。整體給人一種放鬆、舒適，甚至有些許懷舊的氛圍。',
      'text_vector': null,
      'audio_path': './audios/3-1_2.wav',
      'audio_vector': null,
    },
    {
      'id': 4,
      'text_desc': '熱烈的歡呼聲，人們大聲尖叫、吶喊，氣氛相當興奮和激動。可以判斷現場可能正在進行某種激烈的活動、比賽，或者是一個非常精彩的表演，觀眾們的情緒高漲。',
      'text_vector': null,
      'audio_path': './audios/3-1_3.wav',
      'audio_vector': null,
    },
  ],
];

// 處理每個資料項目（補上音訊特徵與文字特徵）
for (let i = 0; i < arr_data.length; i++) {
  const item = arr_data[i];

  // 讀取音訊檔案
  const buffer = readFileSync(item.audio_path);
  const audio_data = processAudio(buffer);

```

```

// 將音訊數據轉換為模型輸入格式
const audio_inputs = await audio_processor(audio_data);

// 取得音訊特徵
const audio_embeds = await audio_model(audio_inputs);

// 儲存音訊特徵
arr_data[i].audio_vector =
Array.from(audio_embeds.audio_embeds.ort_tensor.cpuData);

// 執行特徵擷取
const text_feature = await pipe(item.text_desc);

// 對文字特徵進行 mean pooling
const pooled = mean_pooling(text_feature.ort_tensor.cpuData,
text_feature.ort_tensor.dims);

// 儲存文字特徵
arr_data[i].text_vector = Array.from(pooled);
}

// 檢視變數內容
console.log(arr_data[0]);
console.table(arr_data[0]);

// 建立向量資料庫
const db = await lancedb.connect("./db");

// 定義資料表結構
const schema = new ArrowSchema([
  new Field('id', new Int64(), false),
  new Field('text_desc', new Utf8(), false),
  new Field(
    'text_vector',
    new FixedSizeList(1024, new Field('item', new Float32(),
false)),
    false // 代表欄位為非空值 (non-null)
  ),
]);

```

```

    new Field('audio_path', new Utf8(), true),
    new Field(
        'audio_vector',
        new FixedSizeList(512, new Field('item', new Float32(),
false))),
        false // 代表欄位為非空值 (non-null)
    )
]);

// 建立資料表
const tbl = await db.createEmptyTable("knowledge", schema);

// 將資料寫入資料表
await tbl.add(arr_data);

console.log("資料表清單：", await db.tableNames());

```

#### 4-2.js

```

/**
 * 向量檢索
 *
 * 參考資料：
 * https://www.npmjs.com/package/@lancedb/lancedb
 * https://lancedb.com/docs/concepts/search/vector-search/
 *
 */

// 匯入模組
import { pipeline } from '@huggingface/transformers';
import * as lancedb from "@lancedb/lancedb";

// 建立文字特徵擷取管道
const pipe = await pipeline('feature-extraction', "Xenova/bge-m3",
{ dtype: 'auto' });

// 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];
    }
}

// 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;
}

return pooled;
}

// 開啟向量資料庫
const db = await lancedb.connect("./db");

```



```

// 取得資料表
const tbl = await db.openTable("knowledge");

// 檢視資料筆數
const total = await tbl.countRows();
console.log(`資料總筆數: ${total}`);

// 將整張表查出來 (請對大表加上 .limit() 以避免記憶體爆掉)
const rows = await tbl.query().limit(total).toArray();
console.table(rows);

// 初始化搜尋文字
// let search_text = "清澈悅耳的吉他聲，伴隨著悠揚的旋律。吉他的音色溫柔而
// 略帶憂鬱，可能是一段輕柔的演奏或背景音樂。整體給人一種放鬆、舒適，甚至有些許
// 懷舊的氛圍。";
let search_text = "柔和清亮的吉他聲輕輕響起，旋律悠揚而帶有一點淡淡的哀愁，
彷彿一段輕音樂，營造出放鬆又懷舊的氣息。";

// 使用文字進行向量檢索
const text_inputs = await pipe(search_text);
const text_vector = mean_pooling(text_inputs.ort_tensor.cpuData,
text_inputs.ort_tensor.dims);

// 執行向量檢索
const results = await tbl
    .search(text_vector)
    .distanceType('cosine') // 預設是 'l2'，設定為 'cosine' 的話，要改成
    「1 - 距離」
    .select(['id', 'text_desc', 'audio_path', '_distance'])
    .limit(3)
    .toArray();

// 顯示檢索結果
console.log(`檢索到 ${results.length} 筆資料: `);
console.table(results);

// 顯示檢索結果
for (const result of results) {

```

```
console.log(`文件編號: ${result.id}`);
console.log(`文字描述: ${result.text_desc}`);
console.log(`音訊路徑: ${result.audio_path}`);
console.log(`相似度: ${1 - result._distance}`);
}
```

#### 4-3.js

```
/**
 * 向量檢索 (使用音訊檔案)
 *
 * 參考資料:
 * https://www.npmjs.com/package/@lancedb/lancedb
 * https://lancedb.com/docs/concepts/search/vector-search/
 *
 */

// 匯入模組
import { AutoProcessor, ClapAudioModelWithProjection } from
  '@huggingface/transformers';
import wavefile from 'wavefile';
import { readFileSync } from 'fs';
import * as lancedb from "@lancedb/lancedb";

// 初始化音訊特徵擷取模型
const audio_processor = await
  AutoProcessor.from_pretrained("Xenova/clap-htsat-unfused");
const audio_model = await
  ClapAudioModelWithProjection.from_pretrained("Xenova/clap-htsat-
  unfused", {
    dtype: 'auto',
    device: 'cpu'
  });

// 建立函式
function processAudio(buffer) {
  // 將 Buffer 轉換為 WaveFile
  let wav = new wavefile.WaveFile(buffer);
}
```

```

    wav.toBitDepth('32f'); // 將輸入的音訊轉換為 32 位元浮點數格式
    (Float32)
    wav.toSampleRate(16000); // 轉換成 16kHz 的採樣率
    let audioData = wav.getSamples();

    // 如果音訊是多通道的，則將其轉換為單通道
    if (Array.isArray(audioData)) {
        if (audioData.length > 1) {
            const SCALING_FACTOR = Math.sqrt(2);

            // 合併頻道來節省記憶體
            for (let i = 0; i < audioData[0].length; ++i) {
                audioData[0][i] = SCALING_FACTOR * (audioData[0][i] +
audioData[1][i]) / 2;
            }
        }

        // 選擇第一個頻道的音訊數據
        audioData = audioData[0];
    }

    return audioData;
}

// 開啟向量資料庫
const db = await lancedb.connect("./db");

// 取得資料表
const tbl = await db.openTable("knowledge");

// 檢視資料筆數
const total = await tbl.countRows();
console.log(`資料總筆數: ${total}`);

// 將整張表查出來 (請對大表加上 .limit() 以避免記憶體爆掉)
const rows = await tbl.query().limit(total).toArray();
console.table(rows);

```

```

// 取得範例音訊檔案
const audio_path = "./audios/4-3_0.wav";

// 使用音訊檔案進行向量檢索
let buffer = readFileSync(audio_path);
let audio = await processAudio(buffer);

// 將音訊數據轉換為模型輸入格式
const audio_inputs = await audio_processor(audio);

// 取得音訊特徵
const audio_embeds = await audio_model(audio_inputs);

// 執行向量檢索
const results = await tbl
  .search(audio_embeds.audio_embeds.ort_tensor.cpuData)
  .distanceType('cosine') // 預設是 'l2'，設定為 'cosine' 的話，要改成
  「1 - 距離」
  .select(['id', 'text_desc', 'audio_path', '_distance'])
  .limit(3)
  .toArray();

// 顯示檢索結果
console.log(`檢索到 ${results.length} 筆資料：`);
console.table(results);

// 顯示檢索結果
for (const result of results) {
  console.log(`文件編號：${result.id}`);
  console.log(`文字描述：${result.text_desc}`);
  console.log(`音訊路徑：${result.audio_path}`);
  console.log(`相似度：${1 - result._distance}`);
}

```

## （五）智慧文件處理

智慧文件處理（Intelligent Document Processing，IDP）是利用 AI、OCR、NLP 與機器學習，從 PDF、影像、表單等文件中自動擷取文字、欄位資訊、表格與結構內容，並可進行文件分類、摘要與問答，將非結構化或半結構化資料轉成可操作、可整合的結構化資訊，顯著提升處理速度與準確率。

5-1.js

```
/**
 * 圖片內容檢視
 *
 * 參考模型: https://huggingface.co/onnx-community/Florence-2-base-ft
 */

// 匯入模組
import {
    Florence2ForConditionalGeneration,
    AutoProcessor,
    load_image,
    RawImage,
} from '@huggingface/transformers';
import path from 'path';

// 讀取模型與處理器
const model_id = 'onnx-community/Florence-2-base-ft';
const model = await
    Florence2ForConditionalGeneration.from_pretrained(model_id, { dtype:
        'auto' });
const processor = await AutoProcessor.from_pretrained(model_id);

// 讀取網路上的圖片
// const url =
    'https://huggingface.co/datasets/huggingface/documentation-
    images/resolve/main/transformers/tasks/car.jpg';
// const image = await load_image(url);

// 或者從本地檔案系統讀取圖片
```

```

const image_path = './images/1-2_0.jpg'; // 替換為你的圖片路徑
const absPath = path.resolve(image_path);
const image = await RawImage.read(absPath);

// 定義任務描述（請參考以下列表中的一個選項）
/**
 * 參考連結：
 * https://huggingface.co/microsoft/Florence-2-base-ft
 *
 * <OD>
 * <CAPTION>
 * <DETAILED_CAPTION>
 * <MORE_DETAILED_CAPTION>
 * <CAPTION_TO_PHRASE_GROUNDING>（要在 prompts 變數後面加一些描述）
 * <DENSE_REGION_CAPTION>
 * <REGION_PROPOSAL>
 * <OCR>
 * <OCR_WITH_REGION>
 */
// const task = '<CAPTION_TO_PHRASE_GROUNDING>';
const task = '<CAPTION>';
const prompts = processor.construct_prompts(task);

// 前處理圖片與提示
// 備註：如果使用 <CAPTION_TO_PHRASE_GROUNDING> 任務，則需要在 prompts 之
// 後加上描述，例如：
// const inputs = await processor(image, prompts + 'A remote on the
// left side of the image.');
```

```

const inputs = await processor(image, prompts);

// 生成文字（尚未解碼）
const generated_ids = await model.generate({
  ...inputs,
  max_new_tokens: 100,
});

// 將生成的 ID 轉換為文字（解碼）

```

```

const generated_text = processor.batch_decode(generated_ids,
{ skip_special_tokens: false })[0];

// 處理後生成的文字
const results = processor.post_process_generation(generated_text,
task, image.size);

// 檢視變數內容
console.log(results);

```

## 5-2.js

```

/**
 * 語音內容檢視
 *
 * 參考模型: https://huggingface.co/onnx-community/ultravox-v0\_5-llama-3\_2-1b-ONNX
 */

// 匯入模組
import { UltravoxProcessor, UltravoxModel } from
"@huggingface/transformers";
import wavefile from 'wavefile';

// 模型 ID
const model_id = "onnx-community/ultravox-v0_5-llama-3_2-1b-ONNX";

// 讀取模型與處理器
const processor = await UltravoxProcessor.from_pretrained(model_id);
const model = await UltravoxModel.from_pretrained(model_id, {
  dtype: {
    embed_tokens: "q8", // "fp32", "fp16", "q8"
    audio_encoder: "q4", // "fp32", "fp16", "q8", "q4", "q4f16"
    decoder_model_merged: "q4", // "q8", "q4", "q4f16"
  },
});

// 建立函式

```

```

function processAudio(buffer) {
  // 將 Buffer 轉換為 WaveFile
  let wav = new wavefile.WaveFile(buffer);
  wav.toBitDepth('32f'); // 將輸入的音訊轉換為 32 位元浮點數格式
                           (Float32)
  wav.toSampleRate(16000); // 轉換成 16kHz 的採樣率
  let audioData = wav.getSamples();

  // 如果音訊是多通道的，則將其轉換為單通道
  if (Array.isArray(audioData)) {
    if (audioData.length > 1) {
      const SCALING_FACTOR = Math.sqrt(2);

      // 合併頻道來節省記憶體
      for (let i = 0; i < audioData[0].length; ++i) {
        audioData[0][i] = SCALING_FACTOR * (audioData[0][i] +
audioData[1][i]) / 2;
      }
    }

    // 選擇第一個頻道的音訊數據
    audioData = audioData[0];
  }

  return audioData;
}

// 讀取網路上的音訊檔案
const url = "http://huggingface.co/datasets/Xenova/transformers.js-
docs/resolve/main/mlk.wav";
let buffer = Buffer.from(await fetch(url).then(x => x.arrayBuffer()))

// 或者從本地檔案系統讀取音訊檔案
// const url = "./audios/5-2_0.wav";
// let buffer = readFileSync(url);

// 處理音訊
const audio = processAudio(buffer);

```



```

// 定義訊息
const messages = [
  {role: "system", content: "You are a helpful assistant."},
  {role: "user", content: "Transcribe this audio:<|audio|>"},
];

// 將訊息轉換為文字
const text = processor.tokenizer.apply_chat_template(messages, {
  add_generation_prompt: true,
  tokenize: false,
});

// 將音訊與文字結合
const inputs = await processor(text, audio);

// 生成文字（尚未解碼）
const generated_ids = await model.generate({
  ...inputs,
  max_new_tokens: 128,
});

// 將生成的 ID 轉換為文字（解碼）
const generated_texts = processor.batch_decode(
  generated_ids.slice(
    null,
    [inputs.input_ids.dims.at(-1), null]
  ),
  { skip_special_tokens: true },
);

// 檢視變數內容
console.log(generated_texts);

```

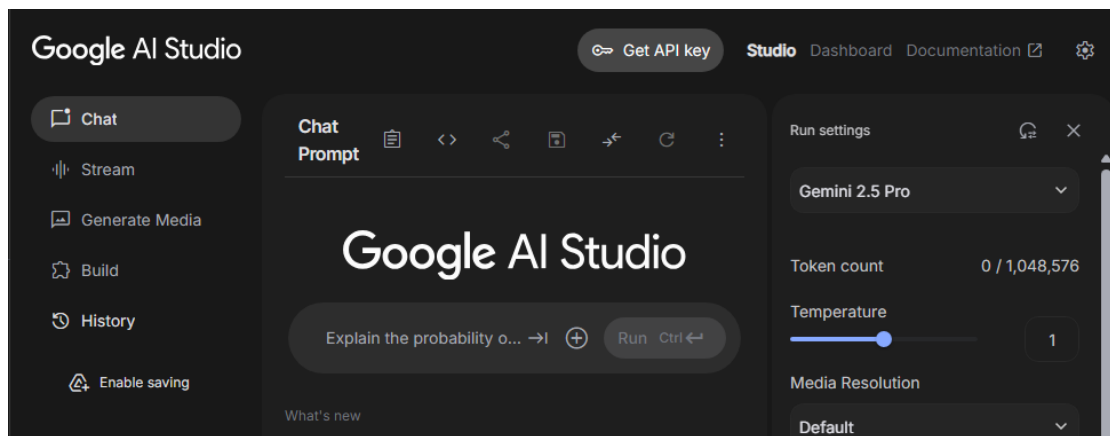
## 四、生成式 AI (Optional)

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。在本單元，嘗試使用 Google AI Studio 的 API key，在免費額度下，嘗試使用大型語言模型的能力。

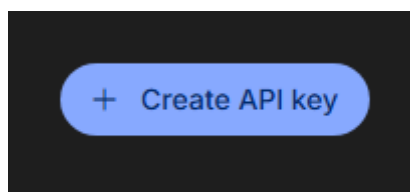
取得 Google AI Studio 的 API key

連結：<https://aistudio.google.com/>

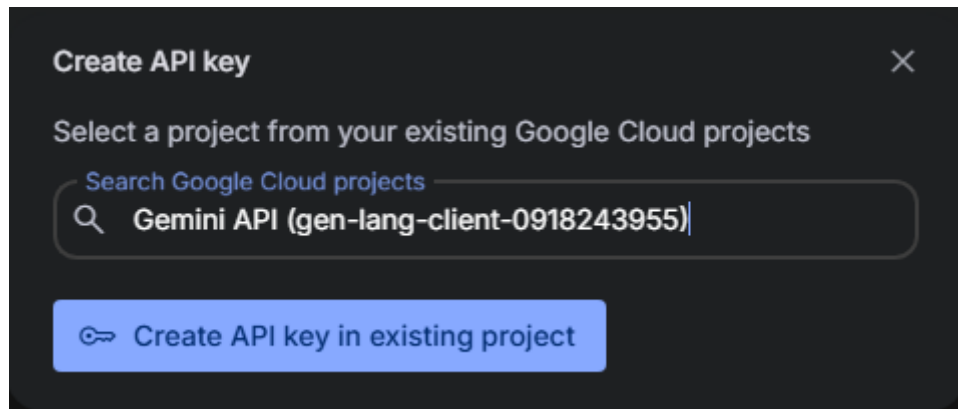
備註：需要在 Chrome 登入 Google 帳號（課程結束後，沒有其它用途的話，強烈建議一律登出）。



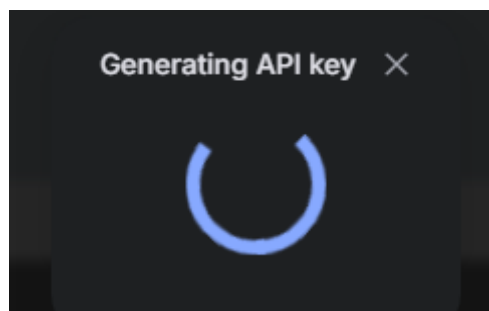
圖：進入 Google AI Studio 頁面，按下上方的「Get API key」



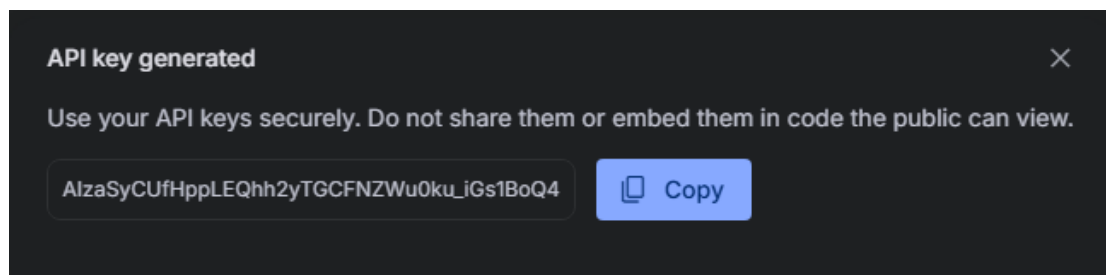
圖：若是先前尚未建立 API key，請點擊一下



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」



圖：產生 API key 的等待畫面



圖：API key 建立完成，請按下「Copy」後，貼在程式當中使用

Project number	Project name	API key	Created	Plan	
...6117	Gemini API <a href="#">🔗</a>	...BoQ4	Aug 1, 2025	Free	<a href="#">🗑️</a>
		...x3IM	Apr 21, 2025	<a href="#">Set up billing</a> <a href="#">View usage data</a>	<a href="#">🗑️</a>

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」

Gemini Developer API 參考文件

<https://ai.google.dev/gemini-api/docs?hl=zh-tw>

## 6-1.js

```
/**
 * 文字生成 - 一次性回應與串流回應
 *
 * Gemini API 說明文件:
 * https://ai.google.dev/gemini-api/docs/text-generation?hl=zh-tw
 *
 * config 參數說明:
 * https://ai.google.dev/api/generate-content?hl=zh-
tw#v1beta.GenerationConfig
 */

// 匯入模組
import { GoogleGenAI } from "@google/genai";
import { get_gemini_api_key } from "../modules/myModule.mjs";

// 建立 GoogleGenAI 物件，並提供 API 金鑰
const ai = new GoogleGenAI({ apiKey:get_gemini_api_key() });

// 使用 GoogleGenAI 物件生成內容
async function main() {
  /**
   * 一般回應
   */

  // 使用模型生成內容
  const response = await ai.models.generateContent({
    model: "gemini-2.0-flash-lite",
    contents: "用 100 個字來描述 AI 的運作方式。",
    config: {
      systemInstruction: "You are a helpful assistant.", // 系統
提示，設定模型的角色或任務
      thinkingConfig: {
        thinkingBudget: 0, // 0 表示不讓模型使用思考模式
      },
    },
  });
}
```

```

        maxOutputTokens: 150, // 最大輸出字數
        stopSequences: ["\n", "。"], // 停止生成的序列
        temperature: 0.7, // 控制生成文本的隨機性，越高越隨機
        topK: 50, // 前 K 個最可能的詞
        topP: 0.9, // 控制生成文本的多樣性（累進機率）
        seed: 42, // 隨機種子，用於生成可重複的結果
    }
});

// 輸出生成的內容
console.log(response.text);

/**
 * 串流回應（沒用到的話，可以先將以下程式碼註解掉）
 */

// 使用模型生成內容的串流回應
// 這樣可以逐步接收生成的文本，而不是一次性獲得完整文字
// const response = await ai.models.generateContentStream({
//     model: "gemini-2.0-flash-lite",
//     contents: "用一段話來描述 AI 的運作方式。",
// });

// 使用 for-await-of 迴圈來處理串流回應
// 每次迭代都會獲得一個新的文字片段
// for await (const chunk of response) {
//     console.log(chunk.text);
// }

}

// 執行主函式
await main();

```

6-2.js

/\*\*

```

* 文字生成 - 多輪對話
*
* Gemini API 說明文件:
* https://ai.google.dev/gemini-api/docs/text-generation?hl=zh-tw
*
* config 參數說明:
* https://ai.google.dev/api/generate-content?hl=zh-
tw#v1beta.GenerationConfig
*
*/

// 匯入模組
import { GoogleGenAI } from "@google/genai";
import { get_gemini_api_key } from "../modules/myModule.mjs";

// 建立 GoogleGenAI 物件，並提供 API 金鑰
const ai = new GoogleGenAI({ apiKey:get_gemini_api_key() });

// 使用 GoogleGenAI 物件生成內容
async function main() {
  const chat = ai.chats.create({
    model: "gemini-2.0-flash-lite",
    history: [
      {
        role: "user",
        parts: [{ text: "你好" }],
      },
      {
        role: "model",
        parts: [{ text: "很高興見到你。你想知道什麼?" }],
      },
    ],
  });

  const response1 = await chat.sendMessage({
    message: "我家裡有兩隻狗。",
  });

  console.log("聊天回應 1:", response1.text);
}

```

```

    const response2 = await chat.sendMessage({
      message: "我家裡的狗有多少隻腳？",
    });
    console.log("聊天回應 2:", response2.text);
  }

// 執行主函式
await main();

```

### 6-3.js

```

/**
 * 圖片生成
 *
 * Gemini API 說明文件:
 * https://ai.google.dev/gemini-api/docs/image-generation?hl=zh-tw
 *
 * config 參數說明:
 * https://ai.google.dev/api/generate-content?hl=zh-
 * tw#v1beta.GenerationConfig
 *
 */

// 匯入模組
import { GoogleGenAI, Modality } from "@google/genai";
import { get_gemini_api_key } from "../modules/myModule.mjs";
import * as fs from "node:fs";

// 建立 GoogleGenAI 物件，並提供 API 金鑰
const ai = new GoogleGenAI({ apiKey: get_gemini_api_key() });

async function main() {
  // 定義要生成的內容。如果要取得較好的結果，請使用下列語言：
  // 英文、西班牙文（墨西哥）、日文、中文（中國大陸）、印地文（印度）。
  const contents =
    "Hi, can you create a 3d rendered image of a pig " +
    "with wings and a top hat flying over a happy " +

```

```

    "futuristic scifi city with lots of greenery?";

// 設定 responseModalities 包含 "Image"，讓模型可以生成圖片
const response = await ai.models.generateContent({
  model: "gemini-2.0-flash-preview-image-generation",
  contents: contents,
  config: {
    responseModalities: [Modality.TEXT, Modality.IMAGE],
  },
});

// 輸出生成的內容
for (const part of response.candidates[0].content.parts) {
  // 基於部分類型，顯示文本或保存圖片
  if (part.text) {
    console.log(part.text);
  } else if (part.inlineData) {
    // 取得圖片數據
    const imageData = part.inlineData.data;

    // 將 base64 編碼的圖片數據轉換為 Buffer
    const buffer = Buffer.from(imageData, "base64");

    // 將圖片數據寫入檔案
    fs.writeFileSync("./images/gemini-native-image.png",
buffer);

    // 輸出圖片保存成功的訊息
    console.log("Image saved as ./images/gemini-native-
image.png");
  }
}

// 執行主函式
await main();

```



## 6-4.js

```
/**
 * 語音生成
 *
 * Gemini API 說明文件:
 * https://ai.google.dev/gemini-api/docs/speech-generation?hl=zh-tw
 *
 * 如果出現「Error [ERR_MODULE_NOT_FOUND]: Cannot find package 'wav'
imported from D:\teach\nodejs_ai_basics\6-4.js」
 * npm install wav --legacy-peer-deps
 */

// 匯入模組
import { GoogleGenAI } from "@google/genai";
import { get_gemini_api_key } from "../modules/myModule.mjs";
import wav from 'wav';

// 建立 GoogleGenAI 物件，並提供 API 金鑰
const ai = new GoogleGenAI({ apiKey: get_gemini_api_key() });

// 使用 wav 模組來寫入 PCM 資料到 WAV 檔案
async function saveWaveFile(filename, pcmData, channels = 1, rate =
24000, sampleWidth = 2) {
    return new Promise((resolve, reject) => {
        // 建立 WAV 檔案寫入器
        const writer = new wav.FileWriter(filename, {
            channels,
            sampleRate: rate,
            bitDepth: sampleWidth * 8,
        });

        // 當寫入完成時，解析 Promise
        writer.on('finish', resolve);

        // 當發生錯誤時，拒絕 Promise
        writer.on('error', reject);

        // 將 PCM 資料寫入 WAV 檔案
    });
}
```

```

        writer.write(pcmData);

        // 關閉寫入器
        writer.end();
    });
}

// 主函式，生成語音並儲存為 WAV 檔案
async function main() {
    // 呼叫 Gemini API 生成語音內容
    const response = await ai.models.generateContent({
        model: "gemini-2.5-flash-preview-tts",
        contents: [
            { parts: [
                { text: 'Say cheerfully: Have a wonderful day!' }
            ]}
        ],
        config: {
            responseModalities: ['AUDIO'],
            speechConfig: {
                voiceConfig: {
                    // Voice 選項:
                    // https://ai.google.dev/gemini-api/docs/speech-
generation#voices
                    prebuiltVoiceConfig: { voiceName: 'Kore' },
                },
            },
        },
    });

    // 取得音訊資料並轉換為 Buffer
    // 備註:「?.」是 Optional Chaining，確保在物件不存在時不會拋出錯誤，
    // 用來安全地存取巢狀物件的屬性，即使中間某一層是 null 或 undefined 也
    // 不會拋錯，
    // 而是直接傳回 undefined。
    const data =
response.candidates?.[0]?.content?.parts?.[0]?.inlineData?.data;
    const audioBuffer = Buffer.from(data, 'base64');

```

```

    // 儲存音訊為 WAV 檔案
    const fileName = './audios/out.wav';
    await saveWaveFile(fileName, audioBuffer);
}

// 執行主函式
await main();

```

## 6-5.js

```

/**
 * 文件解讀
 *
 * Gemini API 說明文件:
 * https://ai.google.dev/gemini-api/docs/document-processing?hl=zh-tw#inline\_data
 *
 */

// 匯入模組
import { GoogleGenAI } from "@google/genai";
import { get_gemini_api_key } from "../modules/myModule.mjs";
import { readFileSync } from 'fs';

// 建立 GoogleGenAI 物件，並提供 API 金鑰
const ai = new GoogleGenAI({ apiKey: get_gemini_api_key() });

// 建立主函式
async function main() {
    // 讀取 PDF 檔案並轉換為 Base64 編碼
    const contents = [
        { text: "對這份文件進行摘要" },
        {
            inlineData: {
                mimeType: 'application/pdf',
                data: Buffer.from(readFileSync("files/6-5_0.pdf")).toString("base64")
            }
        }
    ];
}

```

```

    }
  }
];

// 呼叫 Gemini API 生成內容
const response = await ai.models.generateContent({
  model: "gemini-2.0-flash-lite",
  contents: contents
});

// 輸出回應內容
console.log(response.text);
}

// 執行主函式
await main();

```

## 6-6.js

```

/**
 * 圖像解讀
 *
 * Gemini API 說明文件:
 * https://ai.google.dev/gemini-api/docs/image-understanding?hl=zh-tw
 *
 */

// 匯入模組
import { GoogleGenAI } from "@google/genai";
import { get_gemini_api_key } from "../modules/myModule.mjs";
import { readFileSync } from 'fs';

// 建立 GoogleGenAI 物件，並提供 API 金鑰
const ai = new GoogleGenAI({ apiKey: get_gemini_api_key() });

// 讀取圖片檔案並轉換為 base64 編碼
const base64ImageFile = readFileSync("./images/1-1_2.jpg", {encoding:
"base64"});

```

```
// 準備內容，包含圖片的 base64 編碼和描述文字
const contents = [
  {
    inlineData: {
      mimeType: "image/jpeg",
      data: base64ImageFile,
    },
  },
  { text: "描述這張圖片" },
];

// 呼叫 Gemini API 生成內容
const response = await ai.models.generateContent({
  model: "gemini-2.0-flash-lite",
  contents: contents,
});

// 輸出回應內容
console.log(response.text);
```