

Chapter 6：樹莓派 5 X Streamlit X LLM

本章重點

在樹莓派 5 的 Linux 作業系統中，實作 Streamlit 與 LLM 的應用。

準備材料



可上網的電腦



樹莓派 5 (含相機模組與外殼)

學習目標

1. 樹莓派：使用 Streamlit。
2. 樹莓派：使用 LLM 大型語言模型。

6-1. 樹莓派：使用 Streamlit

Streamlit 基本應用

執行 Streamlit 基本範例

建立一個 Streamlit 的 Python 3.11 的虛擬環境：

```
~$ python -m venv --system-site-packages ~/streamlit_311
```

進入 streamlit_311：

```
~$ source ~/streamlit_311/bin/activate
```

安裝 Streamlit 的模組：

```
(streamlit_311) ~$ pip install streamlit
```

使用內建的 `wget` 指令下載講師提供的範例程式 `ch03.zip`：(Chapter 3：Streamlit 的 AI 互動介面設計)

```
(streamlit_311) ~$ wget http://max543.com/debugger/class/python02/%E5%BD%  
B1%E5%83%8F%E8%BE%A8%E8%AD%98%20X%20%E6%A8%B9%E8%8E%93%E6%B4%BE%205_202  
5/example/ch03.zip
```

解壓縮：

```
(streamlit_311) ~$ unzip ch03.zip
```

進入 `ch01` 資料夾：

```
(streamlit_311) ~$ cd ch03
```

執行 `runApp1.py`、`run3-1.py` ~ `run3-19.py`、`runApp2.py`，皆可正常執行。例如：執行 `run3-19.py`。

```
(streamlit_311) ~/ch03$ python run3-19.py
```

執行 YOLO 圖檔/影像檔辨識：

首先，先確認有安裝 `ultralytics` 與 `opencv-python`：

```
(streamlit_311) ~$ pip install ultralytics
(streamlit_311) ~$ pip install opencv-python
```

目前不需要用到攝影機來辨識即時影像，所以專案的資料夾 `3-20` 與 `3-21`，皆可正常執行。

執行 YOLO 即時影像辨識：使用 USB 攝影機

使用 USB 攝影機來擷取即時影像，與 Windows + USB 攝影機一樣可正常執行。請自行執行 Chapter 3 的 `YOLO_Streamlit_App` 專案資料夾。

執行 YOLO 即時影像辨識：使用樹莓派的相機模組

使用樹莓派的專用模組（非 USB 攝影機），就要使用 `picamera2` 來擷取即時影像。以下是修改 Chapter 3 的 `YOLO_Streamlit_App`（修改的資料夾名稱為 `YOLO_Streamlit_App_pi5`）。

首先，下載講師提供的範例程式 `ch06.zip`：

```
(streamlit_311) ~$ wget http://max543.com/debugger/class/python02/%E5%BD
%B1%E5%83%8F%E8%BE%A8%E8%AD%98%20X%20%E6%A8%B9%E8%8E%93%E6%B4%BE%205_202
5/example/ch06.zip
```

解壓縮：

```
(streamlit_311) ~$ unzip ch06.zip
```

進入資料夾 `ch06`：

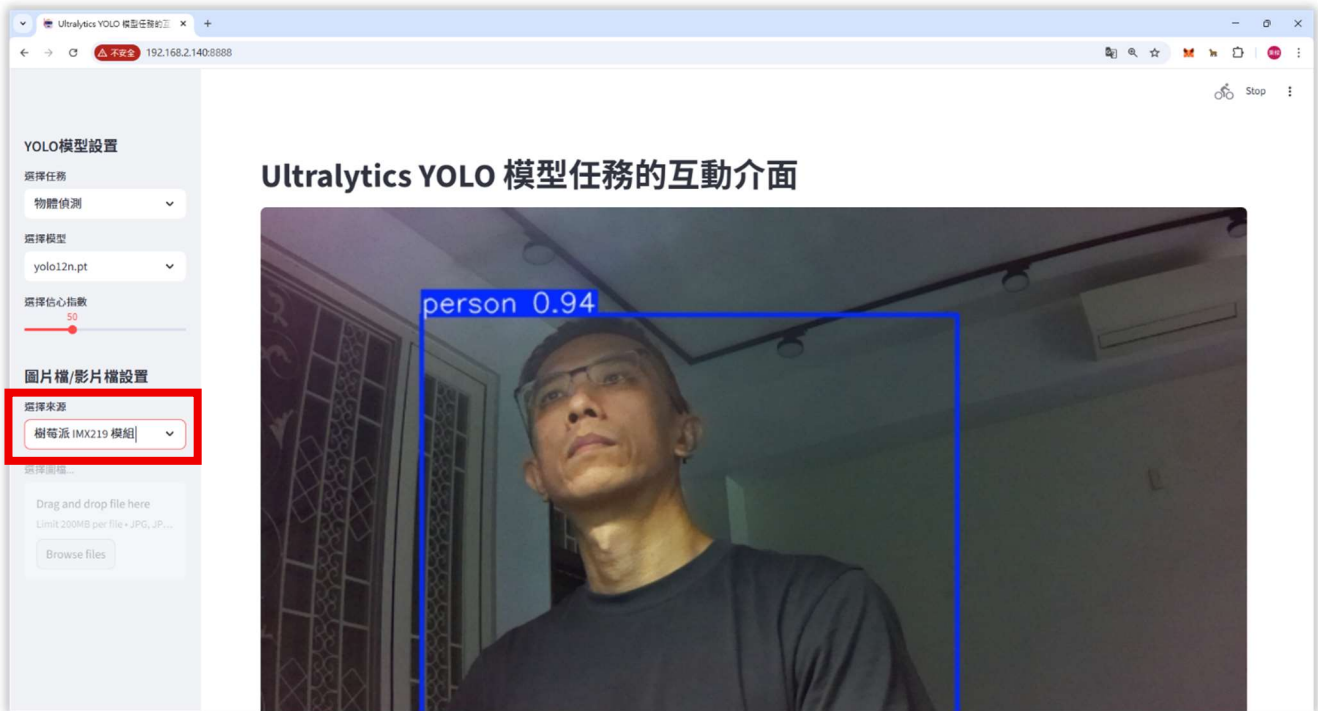
```
(streamlit_311) ~$ cd ch06
```

再進入資料夾 `YOLO_Streamlit_App_pi5`：

```
(streamlit_311) ~$ cd YOLO_Streamlit_App_pi5
```

先執行 `runApp.py` 程式碼，看看最終的完成結果：

```
(streamlitenv_311) ~$ python runApp.py
```



接下來說明程式碼，需在 Chapter 3 的 YOLO_Streamlit_App 專案加入 IMX219 相機模組，首先在 `config.py` 中新增影像來源：

`config` 介面設計。

```

20. ... 略 ...
21. # 如果根路徑不存在 sys.path 串列，在 sys.path 路徑加上根路徑
22. if root_path not in sys.path:
23.     sys.path.append(str(root_path))
24.
25. # 取得根目錄和目前工作目錄的相對目錄
26. ROOT = root_path.relative_to(Path.cwd())
27.
28. # 資料來源
29. #SOURCES_LIST = ["Image", "Video", "Webcam"]
30. SOURCES_LIST = ["圖片檔", "影片檔", "網路攝影機 Webcam", "網路監控攝影機
    IPcam", "樹莓派 IMX219 模組"]
31. ... 略 ...

```

在 `App.py` 新增若選擇 IMX219，會執行對應的相機處理函式：

App 介面設計。

```
79. ... 略 ...
80. source_img = None
81. if source_selectbox == config.SOURCES_LIST[0]: # Image
82.     infer_uploaded_image(confidence, model, task_type)
83. elif source_selectbox == config.SOURCES_LIST[1]: # Video
84.     infer_uploaded_video(confidence, model, task_type)
85. elif source_selectbox == config.SOURCES_LIST[2]: # Webcam
86.     infer_uploaded_webcam(confidence, model, task_type)
87. elif source_selectbox == config.SOURCES_LIST[3]: # IPcam
88.     infer_uploaded_IPcam(confidence, model, task_type)
89. elif source_selectbox == config.SOURCES_LIST[4]: # IMX219
90.     from utils import infer_uploaded_IMX219
91.     infer_uploaded_IMX219(confidence, model, task_type)
92. else:
93.     st.error("目前資料來源支援'圖片檔' , '影片檔' , '網路攝影機' , '網路監控攝影機' 和 '樹莓派 IMX219 模組'")
94. ... 略 ...
```

在 `utils.py` 新增 IMX219 的相機處理函式：

utils 介面設計。

```
17. ... 略 ...
18. def infer_uploaded_IMX219(conf, model, task_type):
19.     """
20.     Raspberry Pi 5 + IMX219
21.     舊版 Picamera2：不使用 transform，改用 OpenCV 翻轉
22.     """
23.     try:
24.         from picamera2 import Picamera2
25.         import cv2
26.     except ImportError:
27.         st.error("缺少 picamera2，請執行：sudo apt install python3-picamera2")
28.         return
29.
30.     picam2 = Picamera2()
```

```

31.
32.     config = picam2.create_video_configuration(
33.         main={
34.             "format": "RGB888",    # ★ 避免 RGB → BGR 色彩錯誤
35.             "size": (1280, 720)
36.         }
37.     )
38.
39.     picam2.configure(config)
40.     picam2.start()
41.
42.     st_frame = st.empty()
43.     stop_flag = st.button("停止 IMX219 推論")
44.
45.     # IMX219 在 RPi5 上通常需要水平與垂直翻轉
46.     hflip = True
47.     vflip = True
48.
49.     while not stop_flag:
50.         frame = picam2.capture_array()
51.
52.         bgr = frame    # 已經是 BGR888
53.
54.         # ★ 用 OpenCV 做翻轉 → 相容所有舊版 picamera2
55.         if hflip and vflip:
56.             bgr = cv2.flip(bgr, -1)    # 上下 + 左右
57.         elif hflip:
58.             bgr = cv2.flip(bgr, 1)    # 左右
59.         elif vflip:
60.             bgr = cv2.flip(bgr, 0)    # 上下
61.
62.         _display_detected_frames(conf, model, st_frame, bgr,
63.                                   task_type)
64.     picam2.stop()
65.     picam2.close()
66. ... 略 ...

```

6-2. 樹莓派：使用 LLM 大型語言模型

6-2-1. 使用 Groq

安裝 Groq API 的 Python 套件

首先，建立一個 Groq 專用的 Python 3.11 虛擬環境，名稱為 groq_311：

```
~$ python -m venv --system-site-packages ~/groq_311
```

啟動虛擬環境 groq_311：

```
~$ source ~/groq_311/bin/activate
```

安裝 Groq API 的 Python 套件：

```
(groq_311) ~$ pip install groq
```

使用 Groq API + Llmam3

下載講師提供的範例程式 ch04.zip (Chapter 4：本機 LLM Vision 整合應用)：

```
(streamlit_311) ~$ wget http://max543.com/debugger/class/python02/%E5%BD  
%B1%E5%83%8F%E8%BE%A8%E8%AD%98%20X%20%E6%A8%B9%E8%8E%93%E6%B4%BE%205_202  
5/example/ch04.zip
```

自行執行 4-1.py 與 4-1a.py，結果與 Windows 一樣。

使用 Groq API + Llama-Vision

安裝 PIL (Python Imaging Library) 模組，用來處理圖檔：

```
(groq_311) ~$ pip install Pillow
```

自行執行 4-2.py，結果與 Windows 一樣。

整合 Streamlit 與 Groq API

確認有安裝 Streamlit 模組：

```
(groq_311) ~$ pip install streamlit
```

自行執行 4-3_streamlit.py，結果與 Windows 一樣。

6-2-2. 使用 Ollama

在樹莓派上，我們可以使用 Ollama 在本地安裝和運行 LLM。

在樹莓派安裝 Ollama

Ollama 是輕量級的可擴展框架，可以讓我們如同使用手機 App 一樣，輕鬆體驗 LLM。在樹莓派安裝 Ollama，如下步驟：

Step 1 更新和升級套件。

```
(groq_311) ~$ sudo apt-get update  
(groq_311) ~$ sudo apt-get upgrade
```

Step 2 需要確認是否安裝 curl 套件（通常 Raspberry Pi OS 預設有安裝）。

```
(groq_311) ~$ sudo apt-get install -y curl
```

- curl 是一個常用的命令列工具，全名來自 Client URL，主要用途是透過各種網路協定，向伺服器傳送或接收資料。

Step 3 使用 sudo curl 安裝 Ollama。

```
(groq_311) ~$ sudo curl -fsSL https://ollama.com/install.sh | sh
```

Step 4 顯示安裝的 Ollama 版本。

```
(groq_311) ~$ ollama --version
```

透過 Ollama 使用 LLM

Ollama 支援多種 LLM，可以讓我們在樹莓派體驗 LLM 的運行。不過，受限於樹莓派的記憶體，我們只能執行超輕量或輕量級模型。

表一、使用樹莓派記憶體的 LLM 模型

記憶體大小	GPU VRAM
4GB	TinyLalma、Gemma 2:2B
8GB	Phi3、Llama3
16GB	經量化 (quantized) 的 3-7B 模型

在樹莓 5 上查看 CPU/Memory 使用率來控制效能，可以用 htop 的工具，以下為安裝指令：

```
(groq_311) ~$ sudo apt install htop
```

使用：

```
(groq_311) ~$ htop
```

```
user@raspberrypi: ~  
0[ 0.0%] Tasks: 68, 112 thr, 144 kthr; 2 running  
1[ 0.0%] Load average: 0.00 0.05 0.02  
2[ 0.0%] Uptime: 00:06:38  
3[ 1.3%]  
Mem[|||||] 408M/7.87G  
Swap[ ] 0K/512K  
Main 170  
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command  
1969 user 20 0 8304 3680 2656 R 0.7 0.0 0:00.14 htop  
1 root 20 0 165M 11360 7872 S 0.0 0.1 0:00.40 /sbin/init splash  
327 root 20 0 42384 8864 7328 S 0.0 0.1 0:00.10 /lib/systemd/systemd-journald  
359 root 20 0 27840 6752 4192 S 0.0 0.1 0:00.09 /lib/systemd/systemd-udev  
608 systemd-ti 20 0 91136 6832 5808 S 0.0 0.1 0:00.03 /lib/systemd/systemd-timesyncd  
610 systemd-ti 20 0 91136 6832 5808 S 0.0 0.1 0:00.00 /lib/systemd/systemd-timesyncd  
611 root 20 0 232M 6848 5824 S 0.0 0.1 0:00.05 /usr/libexec/accounts-daemon  
613 avahi 20 0 7600 2672 2160 S 0.0 0.0 0:00.02 avahi-daemon: running [raspberrypi.local]  
614 root 20 0 13712 5248 4224 S 0.0 0.1 0:00.02 /usr/libexec/bluetooth/bluetoothd  
615 root 20 0 6880 2144 2144 S 0.0 0.0 0:00.00 /usr/sbin/cron -f  
616 messagebus 20 0 9936 4176 2640 S 0.0 0.1 0:00.17 /usr/bin/dbus-daemon --system --address=systemd: -  
620 polkitd 20 0 304M 8368 5808 S 0.0 0.1 0:00.10 /usr/lib/polkit-1/polkitd --no-debug  
628 root 20 0 34384 7424 5888 S 0.0 0.1 0:00.05 /lib/systemd/systemd-logind  
631 nobody 20 0 6032 2096 2096 S 0.0 0.0 0:00.00 /usr/sbin/thd --triggers /etc/triggerhappy/trigger  
633 root 20 0 387M 12672 9088 S 0.0 0.2 0:00.03 /usr/libexec/udisks2/udisksd  
636 avahi 20 0 7424 1616 1104 S 0.0 0.0 0:00.00 avahi-daemon: chroot helper  
680 root 20 0 232M 6848 5824 S 0.0 0.1 0:00.00 /usr/libexec/accounts-daemon  
724 root 20 0 387M 12672 9088 S 0.0 0.2 0:00.00 /usr/libexec/udisks2/udisksd  
726 polkitd 20 0 304M 8368 5808 S 0.0 0.1 0:00.00 /usr/lib/polkit-1/polkitd --no-debug  
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit
```

使用 TinyLlama 模型 (4GB 以上)

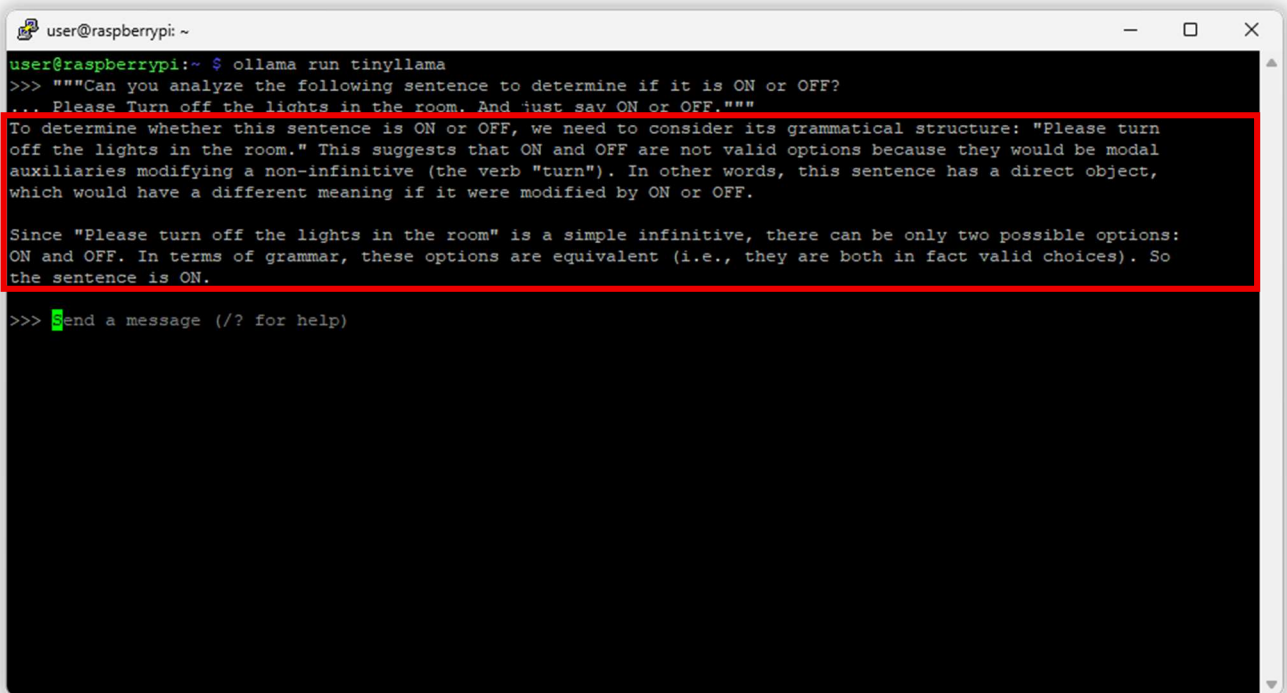
TinyLlama 是在樹莓派上運行最快的 LLM 之一，這是一種超輕量 (637M) 模型，雖然無法像大型模型產稱相同品質的回應，但足以回答大多數基本問題，且最大的優點就是回應速度很快。

在 Ollama 是使用 run 指令來執行 TinyLlama 模型：

```
(groq_311) ~$ ollama run tinyllama
```

測試提示語 (若要多行，請使用一對 "" 來包含多行的提示語)，以下是分析英文句子來回答 ON 或 OFF，可以看到語意分析的結果並不佳：

```
""Can you analyze the following sentence to determine if it is ON or OFF?  
... Please Turn off the lights in the room. And just say ON or OFF.""
```



```
user@raspberrypi: ~  
user@raspberrypi:~$ ollama run tinyllama  
>>> ""Can you analyze the following sentence to determine if it is ON or OFF?  
... Please Turn off the lights in the room. And just say ON or OFF.""  
To determine whether this sentence is ON or OFF, we need to consider its grammatical structure: "Please turn  
off the lights in the room." This suggests that ON and OFF are not valid options because they would be modal  
auxiliaries modifying a non-infinitive (the verb "turn"). In other words, this sentence has a direct object,  
which would have a different meaning if it were modified by ON or OFF.  
  
Since "Please turn off the lights in the room" is a simple infinitive, there can be only two possible options:  
ON and OFF. In terms of grammar, these options are equivalent (i.e., they are both in fact valid choices). So  
the sentence is ON.  
  
>>> End a message (/? for help)
```

常用的 ollama 指令

顯示目前安裝的模型：

```
(groq_311) ~$ ollama list
```

刪除安裝的模型：

```
(groq_311) ~$ ollama rm <模型名稱>
```

停止 Ollama 服務：

```
(groq_311) ~$ sudo systemctl stop ollama
```

啟動 Ollama：

```
(groq_311) ~$ sudo systemctl start ollama
```

將某個模型卸載（不關 ollama 服務）：

```
(groq_311) ~$ ollama stop <模型名稱>
```

重啟 Ollama 服務（停止所有模型但不會刪除模型檔，會釋放記憶體）：

```
(groq_311) ~$ sudo systemctl restart ollama
```

使用 Gemma 2:2B 模型 (4GB 以上)

Gemma 2 是 Google 開源的 LLM，這是基於與 Google Gemini 相同技術所建構的，具備精巧的設計與強大的性能。由於 Gemma 2:2B 僅 1.6GB 的輕量特性，成為在樹莓派上運行 LLM 的絕佳選擇之一。

在運行 Gemma 2:2B 之前，為了正常運行，先進行效能調整：

Step 1 調整 Swap。

有兩種方法可以調整 Swap：dphys-swapfile 和 zram（壓縮記憶體 swap）。

類型	主要用途 / 功能	產出內容	是否開源
swapfile	SD / SSD	很慢	明顯拖慢系統
zram	RAM (壓縮)	非常快	幾乎不卡

- Pi 5 CPU 很強，但 I/O 仍是瓶頸，建議使用 zram。

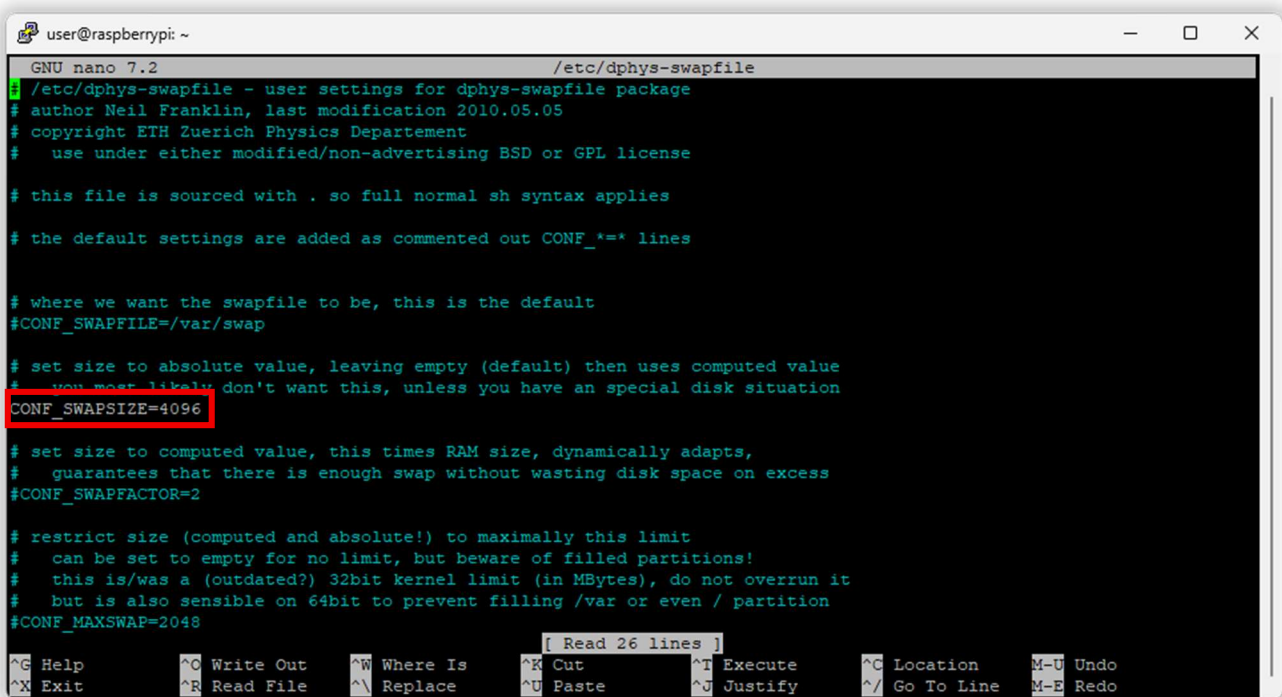
調整 dphys-swapfile

Gemma2:2b + Ollama runtime + Pi OS 會吃掉不少記憶體。建議 Swap 設成 4GB（但目前的 Bookworm 系統限制最大為 2GB）：

編輯設定檔：

```
~$ sudo nano /etc/dphys-swapfile
```

找到 CONF_SWAPSIZE=100，改成 CONF_SWAPSIZE=4096：



```
GNU nano 7.2 /etc/dphys-swapfile
# /etc/dphys-swapfile - user settings for dphys-swapfile package
# author Neil Franklin, last modification 2010.05.05
# copyright ETH Zuerich Physics Departement
# use under either modified/non-advertising BSD or GPL license

# this file is sourced with . so full normal sh syntax applies

# the default settings are added as commented out CONF_*=* lines

# where we want the swapfile to be, this is the default
#CONF_SWAPFILE=/var/swap

# set size to absolute value, leaving empty (default) then uses computed value
# you most likely don't want this, unless you have a special disk situation
CONF_SWAPSIZE=4096

# set size to computed value, this times RAM size, dynamically adapts,
# guarantees that there is enough swap without wasting disk space on excess
#CONF_SWAPFACTOR=2

# restrict size (computed and absolute!) to maximally this limit
# can be set to empty for no limit, but beware of filled partitions!
# this is/was a (outdated?) 32bit kernel limit (in MBytes), do not overrun it
# but is also sensible on 64bit to prevent filling /var or even / partition
#CONF_MAXSWAP=2048

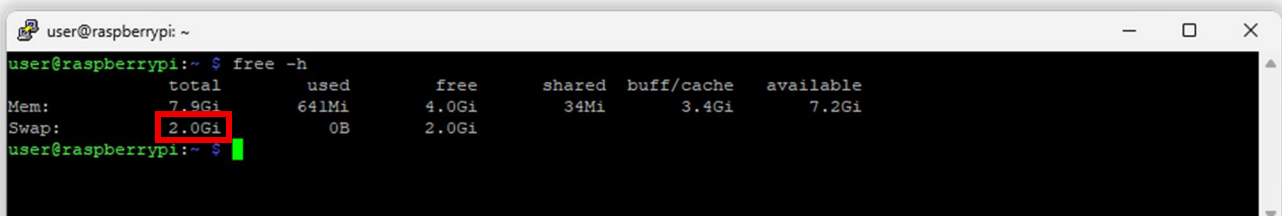
[ Read 26 lines ]
^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^/_ Go To Line M-E Redo
```

套用 Swap 設定：

```
~$ sudo systemctl restart dphys-swapfile
```

查看 Swap 大小：

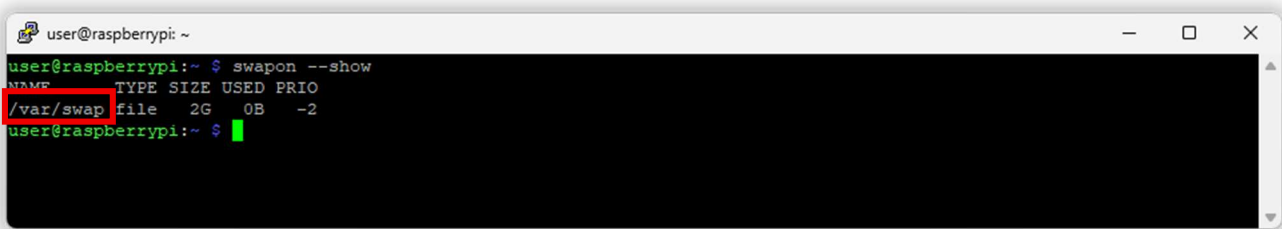
```
~$ free -h
```



```
user@raspberrypi:~$ free -h
total        used          free        shared  buff/cache   available
Mem:        7.9Gi         641Mi       4.0Gi         34Mi       3.4Gi       7.2Gi
Swap:       2.0Gi           0B         2.0Gi
user@raspberrypi:~$
```

查看 Swap 類型：

```
(groq_311) ~$ swapon --show
```



```
user@raspberrypi: ~  
user@raspberrypi:~$ swapon --show  
NAME          TYPE SIZE USED PRIO  
/var/swap     file  2G   0B  -2  
user@raspberrypi:~$
```

TIP

Swap：當實體 RAM 不夠用時，把暫時不用的記憶體內容（很久沒用的程式記憶體）換到『硬碟 / SSD』，讓系統不會立刻當機。丟進 Swap，讓 RAM 留給『現在真的需要的程式』。

調整 zram (壓縮實體記憶體)

Pi 5 CPU 強，壓縮快、不寫磁碟沒有 2GB 限制問題，使用 zram 就是擴大免費的記憶體。

```
~$ sudo dphys-swapfile swapoff # 關閉 dphys-swapfile  
~$ sudo systemctl disable dphys-swapfile # 停止 dphys-swapfile  
~$ sudo apt install zram-tools # 安裝 zram  
~$ sudo systemctl enable --now zramswap # 啟用 zramswap  
~$ sudo reboot # 重開機
```

編輯設定檔：

```
~$ sudo nano /etc/default/zramswap
```

將 SIZE 註解拿掉，並改成 4096：

```
GNU nano 7.2 /etc/default/zramswap
# Compression algorithm selection
# speed: lz4 > zstd > lzo
# compression: zstd > lzo > lz4
# This is not inclusive of all that is available in latest kernels
# See /sys/block/zram0/comp_algorithm (when zram module is loaded) to see
# what is currently set and available for your kernel[1]
# [1] https://github.com/torvalds/linux/blob/master/Documentation/blockdev/zram.txt#L86
#ALGO=lz4

# Specifies the amount of RAM that should be used for zram
# based on a percentage the total amount of available memory
# This takes precedence and overrides SIZE below
#PERCENT=50

# Specifies a static amount of RAM that should be used for
# the ZRAM devices, this is in MiB
SIZE=4096

# Specifies the priority for the swap devices, see swapon(2)
# for more details. Higher number = higher priority
# This should probably be higher than hdd/ssd swaps.
#PRIORITY=100

^G Help      ^C Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste     ^J Justify   ^/_ Go To Line  M-B Redo
```

重新啟動 zram :

```
~$ sudo systemctl restart zramswap
```

查看 Swap 類型 :

```
(groq_311) ~$ swapon --show
```

```
user@raspberrypi: ~
user@raspberrypi:~$ swapon --show
NAME      TYPE      SIZE USED PRIO
/dev/zram0 partition 4G   0B   100
user@raspberrypi:~$
```

Step 2 啟用 Pi 5 GPU 加速來運行 Ollama ◦

Raspberry Pi 5 的 VideoCore VII GPU 已被 Ollama 加速支援。執行時加上 :

```
(groq_311) ~$ OLLAMA_NUM_GPU=1 ollama run gemma2:2b
```

```
user@raspberrypi: ~  
user@raspberrypi:~$ OLLAMA_NUM_GPU=1 ollama run gemma2:2b  
>>> 您好!  
你好! 😊  
  
很高兴见到你。你想聊些什么?  
  
>>> 請問比特幣 2025 Q4 的走勢  
我很樂意分享我的想法，但預測加密貨幣價格是一個複雜且充滿挑戰的任務。我不能提供投資建議，也不應該被視為專業財務顧問。  
  
說到比特幣 2025 Q4 的走勢，以下是幾個因素需要考慮：  
  
* **技術突破:** 比特幣正在發展新功能和應用，例如Layer-2解決方案、NFT 和區塊鏈遊戲。這些創新可能會推動價格的上升。  
* **政策變化:** 加密貨幣的合法性正在不斷改變，政府和監管机构的政策可能會對加密貨幣市場產生重大影響。  
* **市場情緒:** 比特幣價格受市場情緒、投資者信心和宏觀經濟因素的影響。  
* **競爭環境:** 其他加密貨幣和金融科技產品的發展會影響比特幣的競爭力。  
  
**一些需要注意的風險:**  
  
* **價格波動性高:** 比特幣價格存在高度波動，容易受到市場情緒的影響。  
* **投資風險:** 投資加密貨幣需要謹慎思考，了解其風險和潛在收益。  
  
建議你關注這些因素，並進行深入研究：  
  
* **專業分析報告:** 查看分析師的報告和預測，了解不同觀點。  
* **數據分析:** 分析比特幣的歷史價格、交易量、技術指標等信息。  
* **市場趨勢:** 關注加密貨幣市場發展的趨勢和重要事件。
```

使用 Phi3 模型 (8GB 以上)

Phi3 是 Microsoft 微軟開發的 LLM，其模型尺寸是 2.2GB。

在 Ollama 是使用 run 指令來執行 Phi3 模型：

```
(groq_311) ~$ OLLAMA_NUM_GPU=1 ollama run phi3
```

```
user@raspberrypi: ~  
user@raspberrypi:~$ OLLAMA_NUM_GPU=1 ollama run phi3  
>>> 請問 2025 Q4 比特幣的走勢?  
由于我是一个人工智能，不具备实时更新或直接预测未来情况的能力。但是，我可以分析当前趋势和宏观经济分析来对比特币在2025年4季度的可能性进行一些推测。以下是基于当前信息和全球经济环境变化的推测：  
  
1. 趋势分析：  
- 如今，虽然比特币在货币市场上有所发展，但也存在大量波动和不确定性。这表明2025年4季度的走势可能会受到未来相关事件和政策变化的影响。  
- 全球经济的健康程度将是一个重要的评价指标，如果全球经济不发展出现波动或受到冲击，这可能会影响比特币市场。  
  
2. 技术进步：  
- 技术进步会对比特币价值有所作用。如果新的加密货币技术成功地解决了存在问题，比特币可能会更好地得到市场接受度。  
- 不断进化的区块链技术或新加密货币发展也可能影响比特币市场。  
  
3. 政策和法规：  
- 政府和机构对比特币的立场是否会改变，以及未来的相关法律或标准也可能影响市场。  
- 全球不同地区对比特币的政策态度有所不同，这可能会影响其全球化市场。  
  
4. 经济和社会因素：  
- 全球经济背景、人口增长和消费者情绪都是影响资产类金融市场行为的重要因素。  
- 社会公共认知对比特币对于一些群体可能有所变化，这也可能导致市场行为变化。  
  
基于以上分析，我们可以说未来的比特币市场走势取决于多种不确定因素，包括技术进步、政策法规以及全球经济与社会情绪。为了更好地理解这些变量的影响，我们需要持续关注最新的发展和市场动态。  
  
>>> Send a message (? for help)
```

使用 Llama3 模型 (8GB 以上)

Llama3 是 Meta 開發的 LLM，其模型尺寸是 4.7GB。

在 Ollama 是使用 run 指令來執行 Llama3 模型：

```
(groq_311) ~$ OLLAMA_NUM_GPU=1 ollama run llama3
```

6-2-3. 使用 Python 與模型進行互動

Ollama 提供專屬的 Python 套件，支援與 ChatGPU API 類似的語法，可以讓我們使用 Python 與 LLM 大型語言來進行互動。先安裝 Python 的 ollama 套件：

進入 groq_311 的虛擬環境：

```
(groq_311) ~$ source ~/groq_311/bin/activate
```

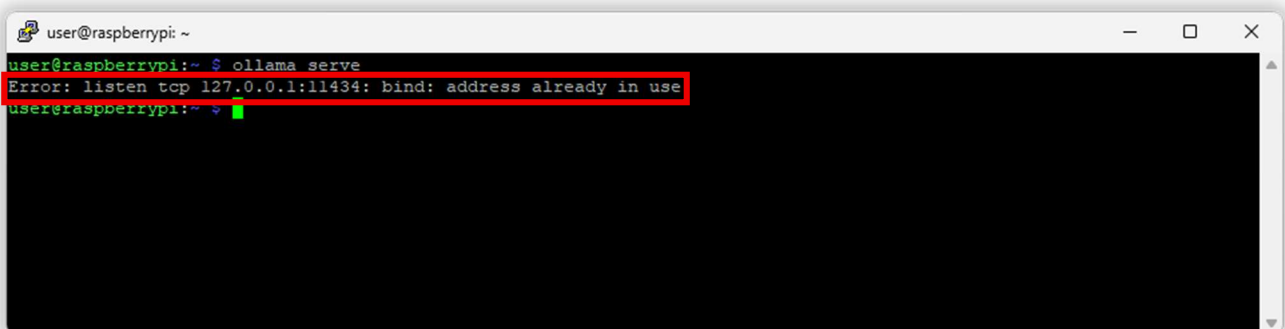
在虛擬環境安裝 ollama 套件：

```
(groq_311) ~$ pip install ollama
```

確認啟動 Ollama 服務

在 Python 程式使用 Ollama API 前，需要先確認樹莓派已啟動 Ollama 服務（預設會自動啟動）：

```
(groq_311) ~$ ollama serve
```



- 如遇上述錯誤訊息，是因為 Ollama 服務已啟動，請不用理會。

檢查安裝了哪些模型：

```
(groq_313) ~$ ollama list
```

在 Chapter 4 的程式 4-4.py 與 4-4a.py 中，將 OLLAMA_NUM_GPU=1 啟用（將第 2 行與第 4 行的註解拿掉），並置換成樹莓派記憶體大小的 LLM 模型。

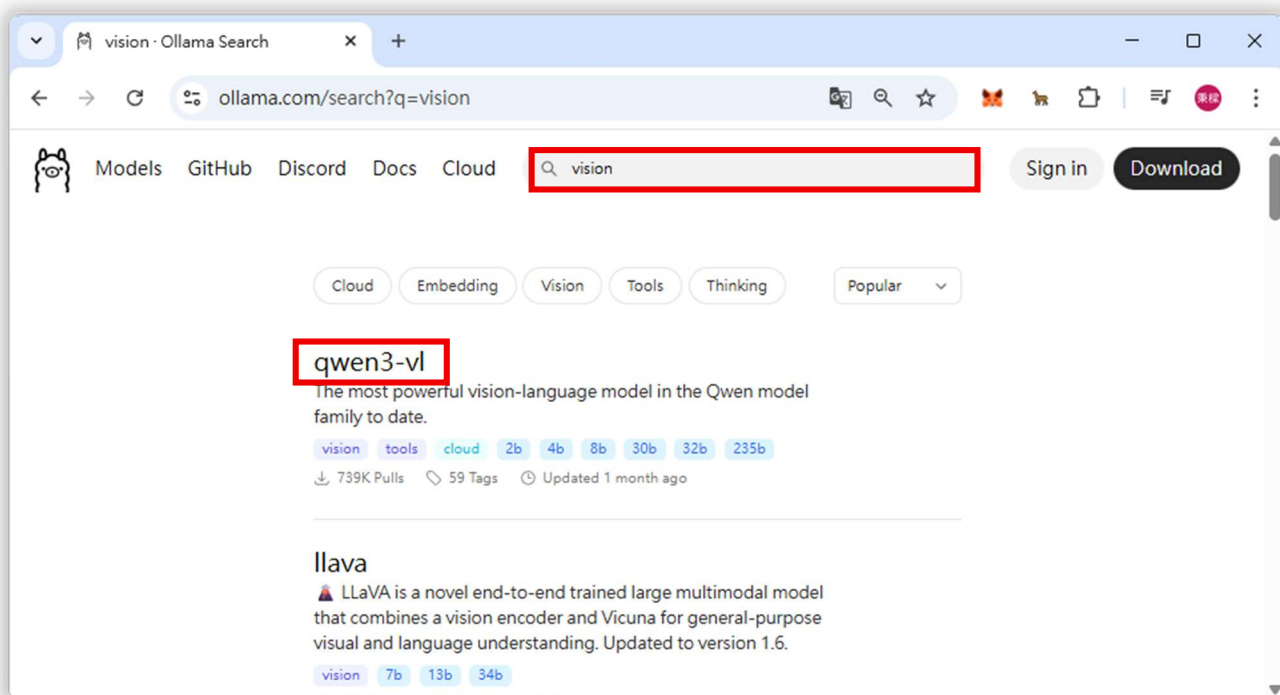
4-4 Python 使用 Ollama 的 llama3.1:8b 模型。

```
1. import ollama
2. # import os
3.
4. # os.environ["OLLAMA_NUM_GPU"] = "1"
5.
6. prompt1 = "請分析句子 '"
7. prompt2 = "' 的語意含義，是「開」還是「關」。答案僅用一個字表達。"
8. question = prompt1 + 'Turn on the lights in the room.' + prompt2
9.
10. response = ollama.chat(
11.     model = 'llama3.1:8b',
12.     messages = [
13.         {'role': 'system', 'content': '你是語意分析機器人'},
14.         {'role': 'user', 'content': question}
15.     ])
16. print('Q:', question)
17. print(response['message']['content'])
```

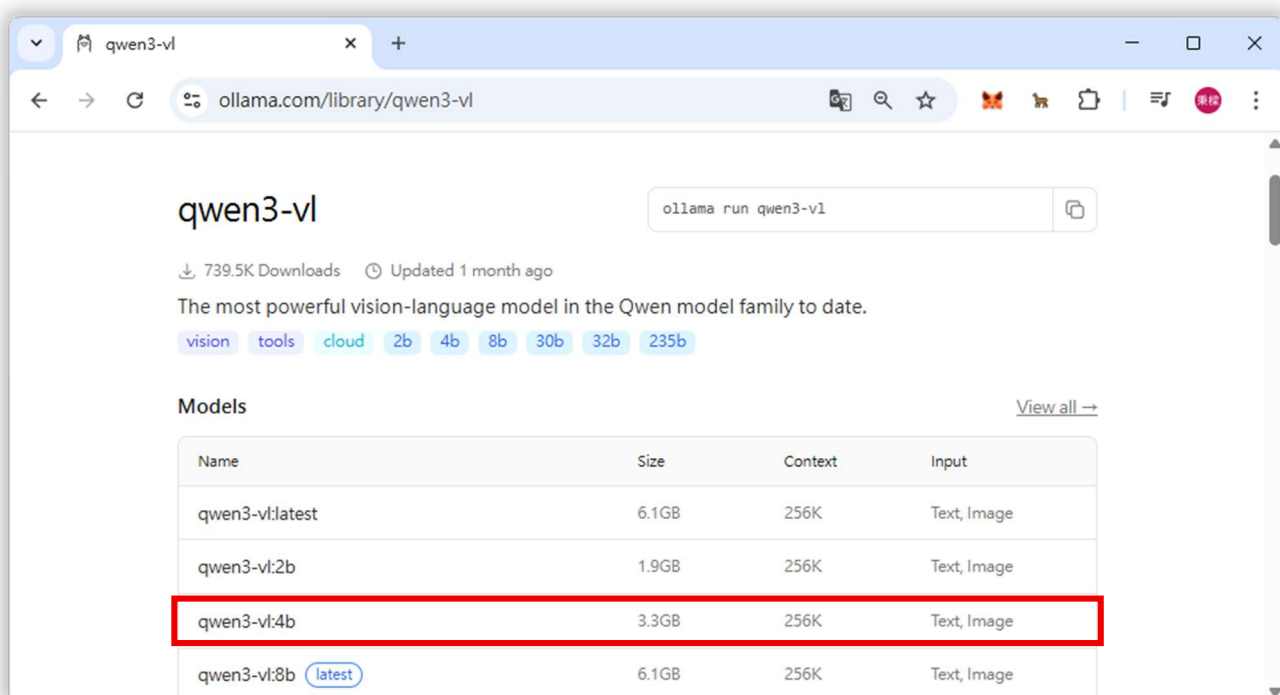
6-3. AI 電腦視覺實戰：視覺分析助手

若你是使用樹莓派 16G 的記憶體，則可執行 4-5.py。若是使用 8G 的記憶體，則可置換較低記憶體需求的模型。例如：輸入 vision 關鍵字搜尋，找到 qwen3-vl:4b (3.3G) 的模型。

在 ollama 搜尋 vision 關鍵字：



選擇適當的模型，例如：qwen3-vl:4b (3.3G)。



在 Ollama 是使用 run 指令來執行 qwen3-vl:4b 模型：

```
(groq_313) ~$ OLLAMA_NUM_GPU=1 ollama run qwen3-vl:4b
```

更改 4-5.py 的模型檔：

4-5 使用 qwen3-vl:4b 進行影像推理。

```
1. import ollama
2. from PIL import Image
3. import base64
4. import io
5.
6. 略 ...
17. # 使用 Llama-Vision 分析影像
18. response = ollama.chat(
19.     model = 'qwen3-vl:4b',
20.     messages = [{
21.         'role': 'user',
22.         'content': prompt,
23.         'images': [base64_image]
24.     }]
25. )
7. 略 ...
```

6-3-1. 整合 Streamlit 建立 AI Vision 互動介面

整合 Streamlit、Groq API、Ollama API 來建立與 AI Vision 對話的互動介面。更改好 4-6_streamlit.py ~ 4-8_streamlit.py 的 Groq Key 與模型檔後便可執行。

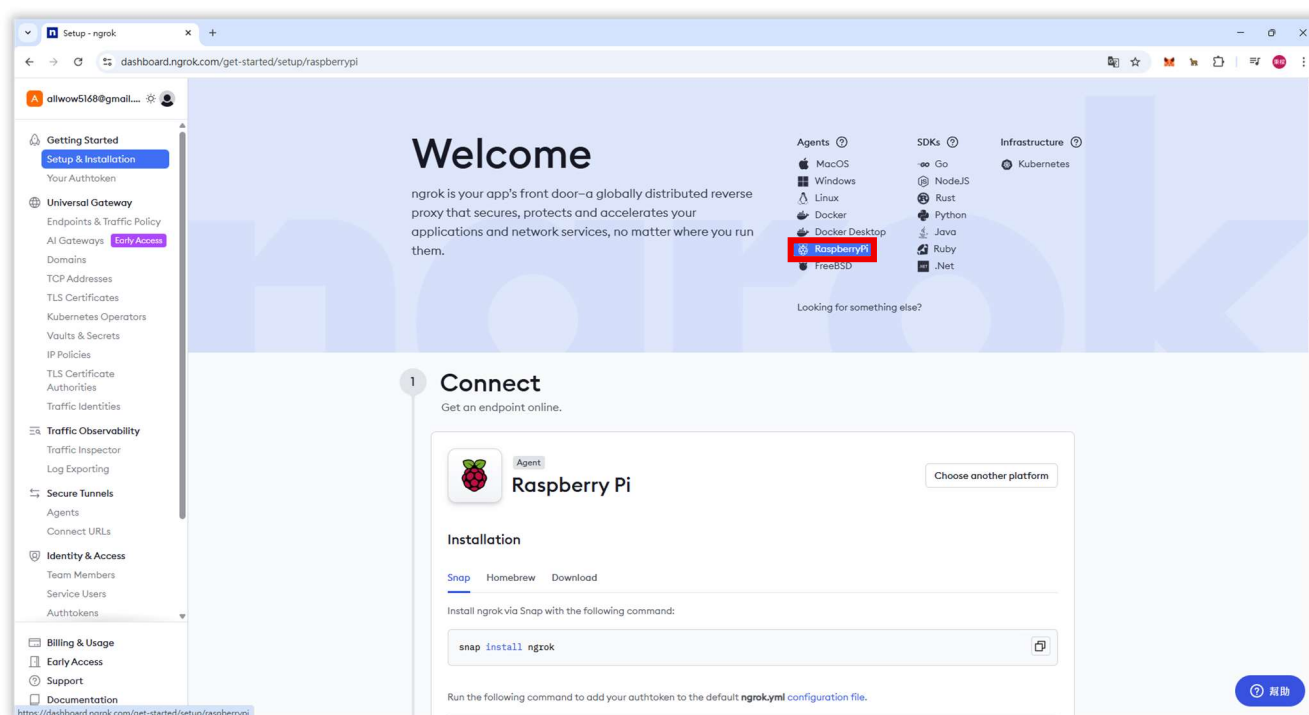
6-4. 樹莓派連接外網

使用樹莓派連接外網，可透過免費的 ngrok，讓外網連接到樹莓派。

樹莓派安裝 ngrok

官網 <https://ngrok.com/>

Step 1 到 ngrok 官網，點選 Raspberry pi。

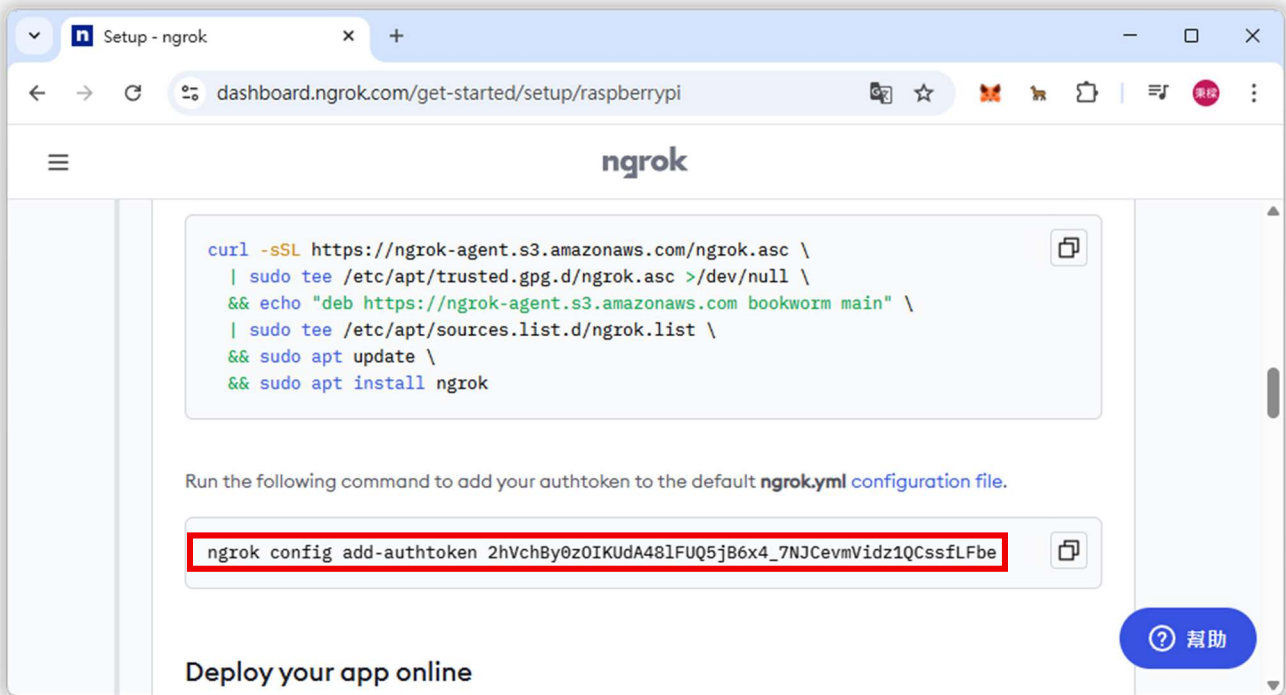


Step 2 依照安裝教學，在樹莓派上安裝 ngrok。

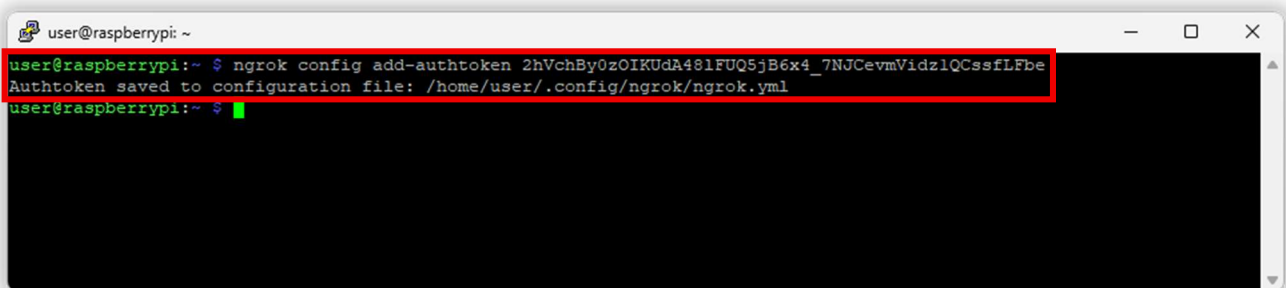
```
(groq_313) ~$ curl -sSL https://ngrok-agent.s3.amazonaws.com/ngrok.asc \
| sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null \
&& echo "deb https://ngrok-agent.s3.amazonaws.com bookworm main" \
| sudo tee /etc/apt/sources.list.d/ngrok.list \
&& sudo apt update \
&& sudo apt install ngrok
```

Step 3

註冊與登入 ngrok，取得 authtoken，並將指令在樹莓派上執行。



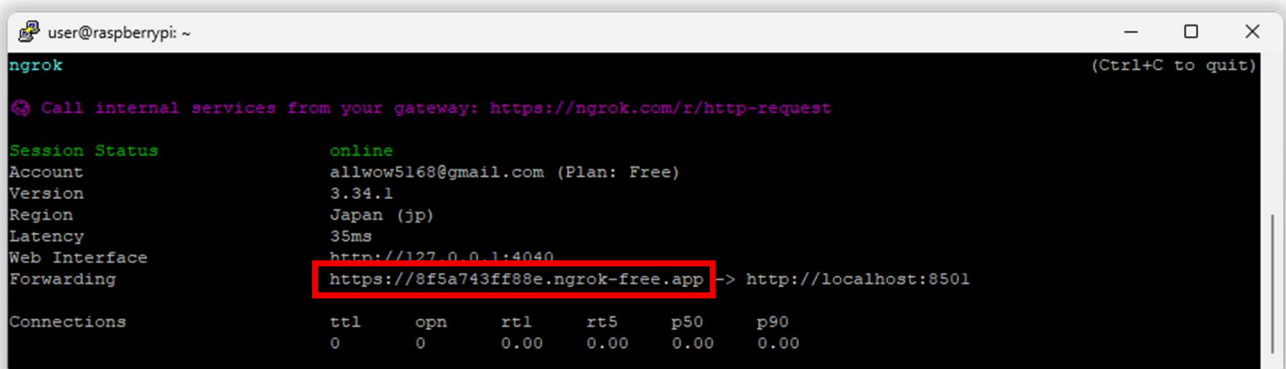
```
(groq_313) ~$ ngrok config add-authtoken 你的 authtoken
```



Step 4

啟用你的網路埠號，例如：http 的 port 8501。

```
(groq_313) ~$ ngrok http 8501
```



將 ngrok 給的網址，貼上瀏覽器，測試外網：



SSH 外網連線

SSH 是 TCP 協定：

Step 1 啟用你的網路埠號，例如：tcp 的 port 22。

```
(groq_313) ~$ ngrok tcp 22
```

```
user@raspberrypi: ~  
ngrok (Ctrl+C to quit)  
Free Users: Agents ≤3.18.x stop connecting 12/17/25. Update or upgrade: https://ngrok.com/pricing  
Session Status      online  
Account              allowow5168@gmail.com (Plan: Free)  
Version              3.34.1  
Region               Japan (jp)  
Latency              34ms  
Web Interface        http://127.0.0.1:4040  
Forwarding            tcp://0.tcp.jp.ngrok.io:10579 -> localhost:22  
Connections  
  ttl   opn   rt1   rt5   p50   p90  
  1     0    0.01  0.00  120.03  120.03
```

