奕瑞科技

AI Box小試身手

# 加速邊際運算落地運用

奕瑞科技AI研發部門

主講人 工程師 吳宇鴻、謝明原

01. Xavier 介紹

02. Xavier NX介紹

03. 場景應用

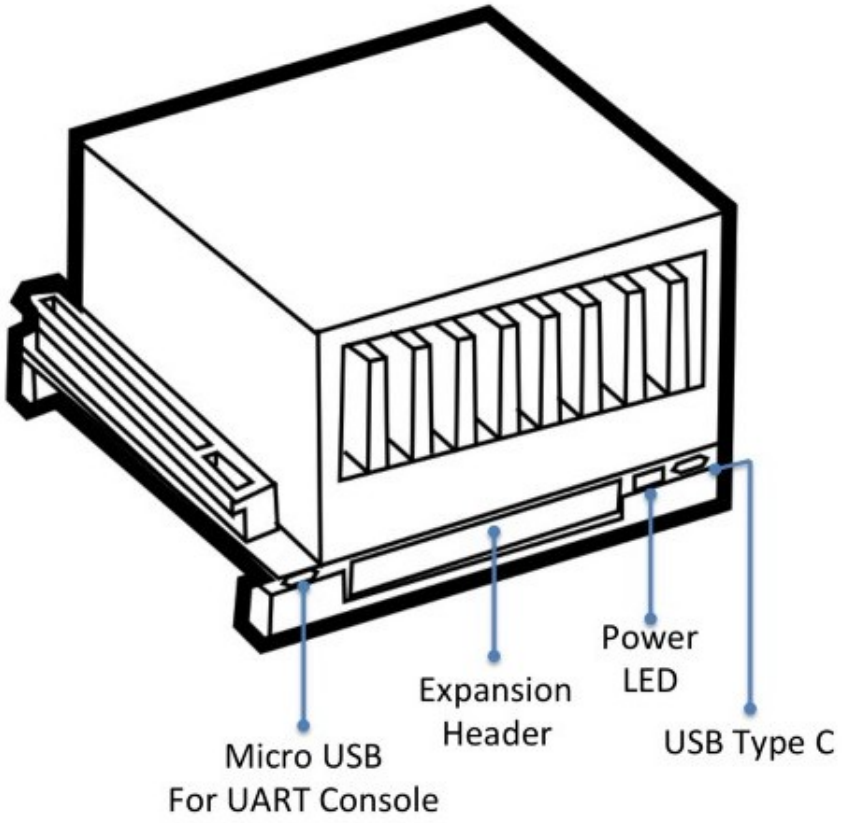
04. AI BOX小試身手

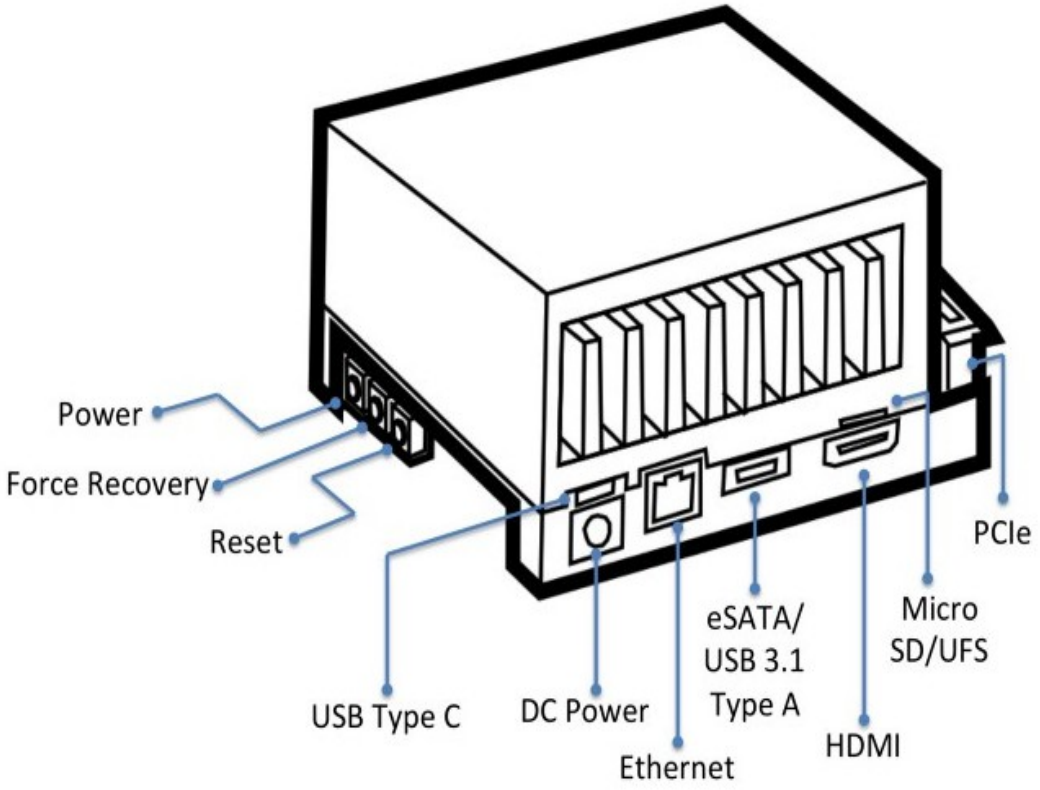
奕瑞科技

奕瑞科技

# 01. Xavier 介紹

# Xavier介紹


Micro USB
For UART Console
Expansion
Header
Power
LED
USB Type C


Power
Force Recovery
Reset
USB Type C
DC Power
Ethernet
eSATA/
USB 3.1
Type A
HDMI
Micro
SD/UFS
PCIe

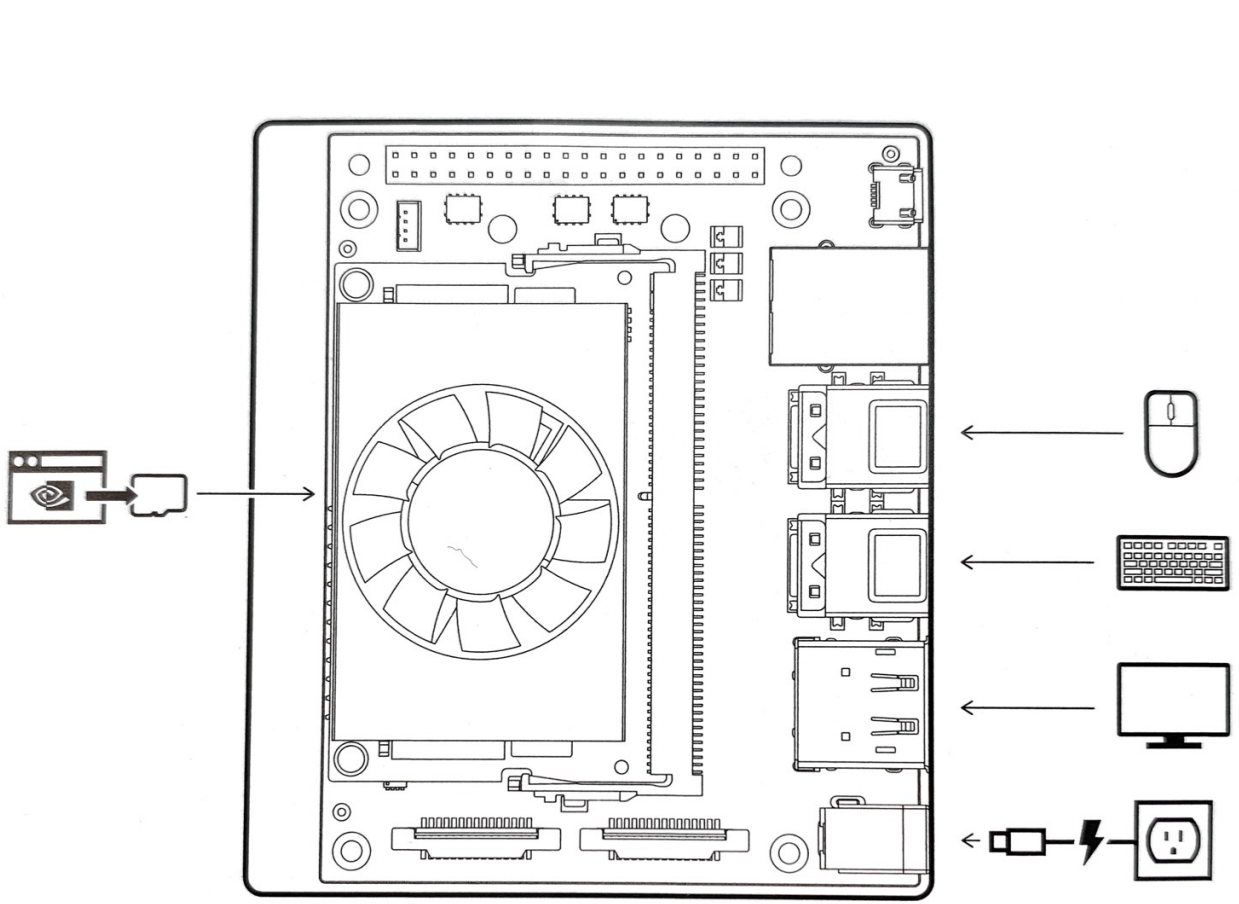
# Xavier介紹

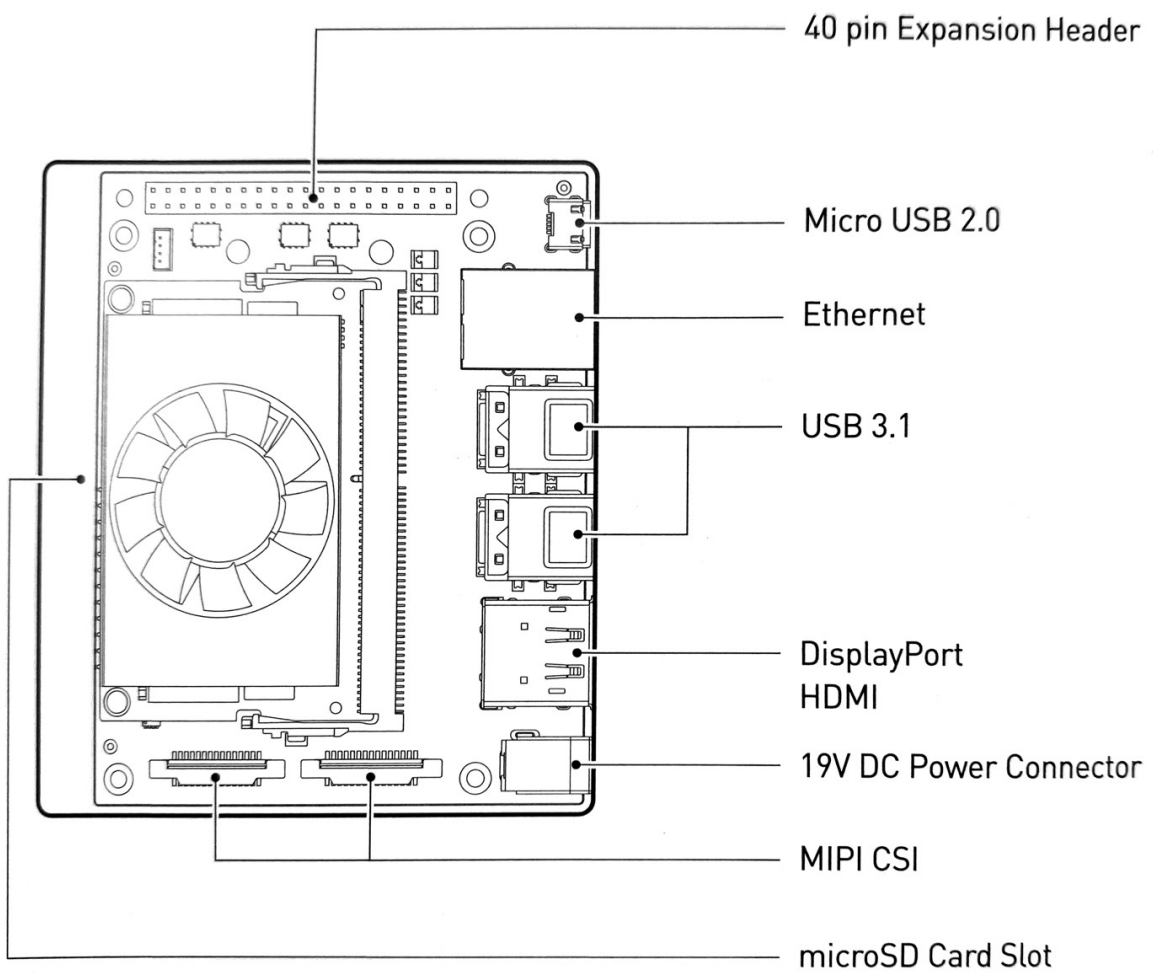
- 8核CPU
- 16G記憶體(32G)
- 32G儲存空間
- 體積小
- 省電
- 容易維護容易擴大規模
- 可適應較嚴苛的工廠環境
- DeepStream SDK
- 同樣算力下可使用到的RAM較大

奕瑞科技

# 02. Xavier NX 介紹

# Xavier NX介紹





# Xavier NX介紹

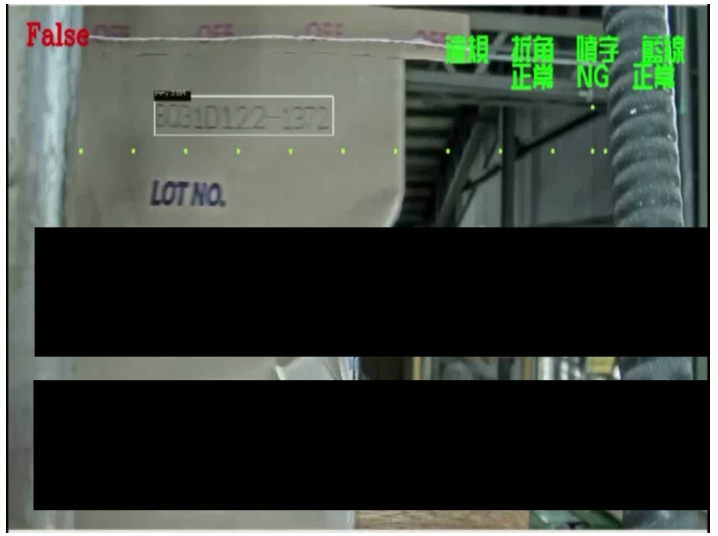
- 6核CPU
- 8G記憶體
- 沒有記憶體儲存空間，但可透過SD卡擴充
- 體積比xavier更小，效能為xavier的一半
- 省電
- 容易維護容易擴大規模
- 可適應較嚴苛的工廠環境
- DeepStream SDK
- 同樣算力下可使用到的RAM較大

奕瑞科技

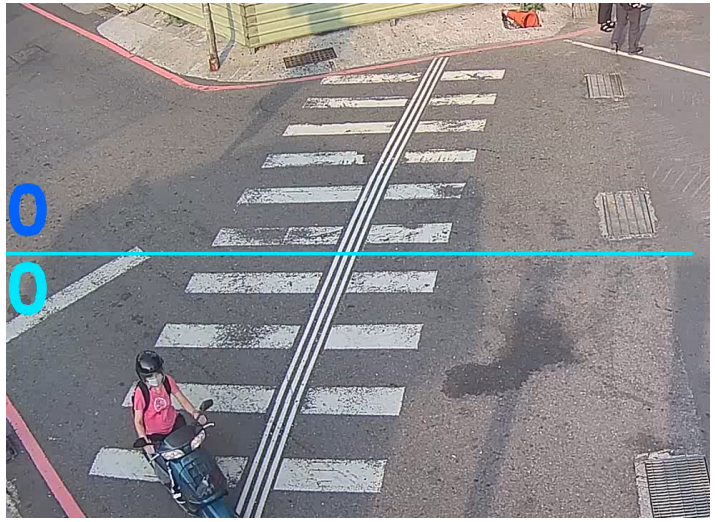
# 03. 場 景 應 用

# 場景應用


BEM3953
BEM3953
BEM3953
BEM3953


False
LOT NO.


Warning: Mask


0
0

奕瑞科技

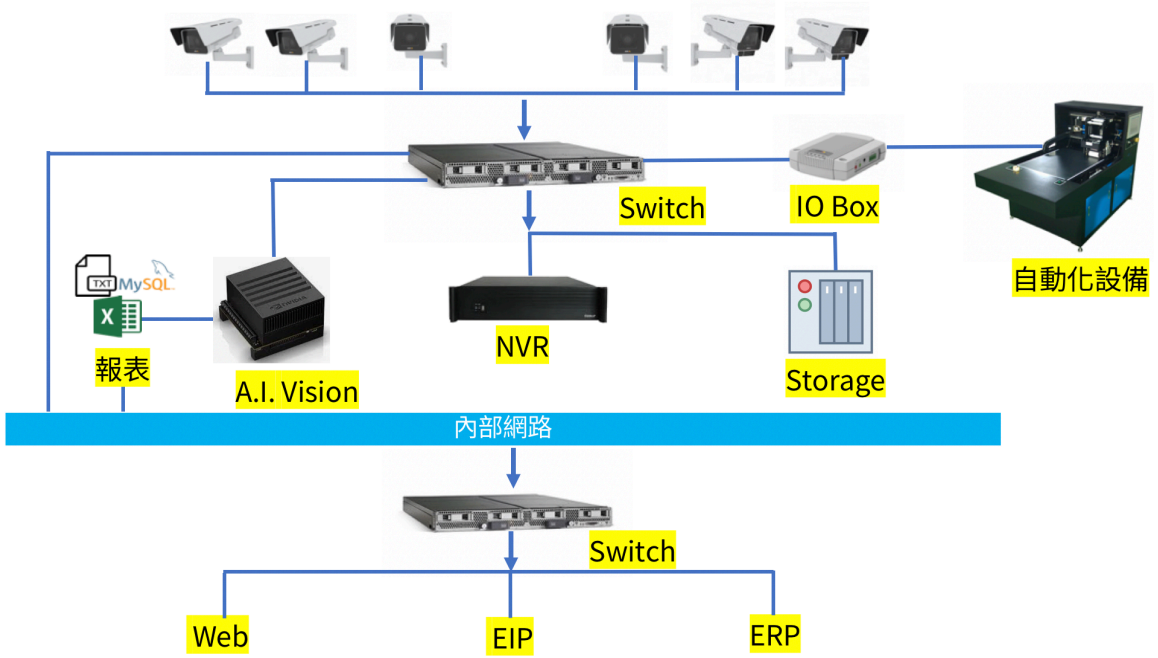
# 場景應用

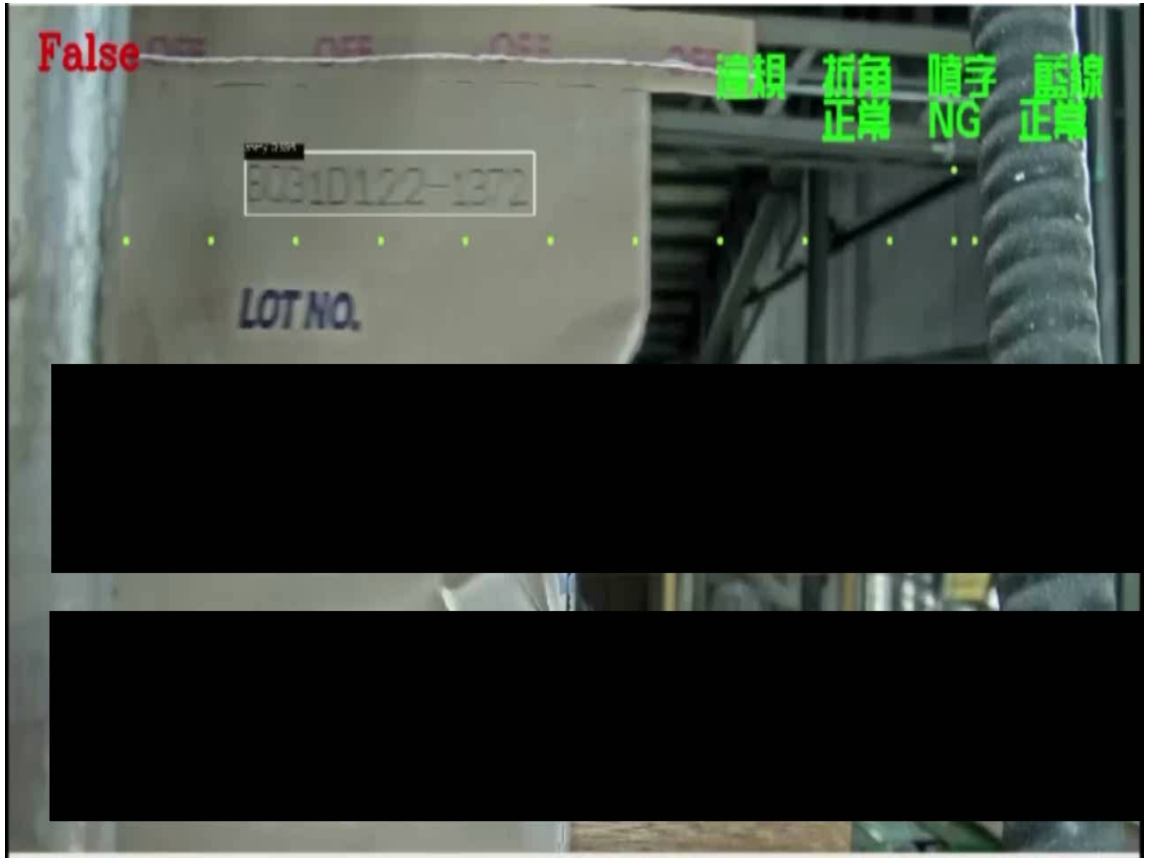
- 道路贓車、通緝車輛辨識
- 克服車上無法放置大型電腦


奕瑞科技

# 場景應用

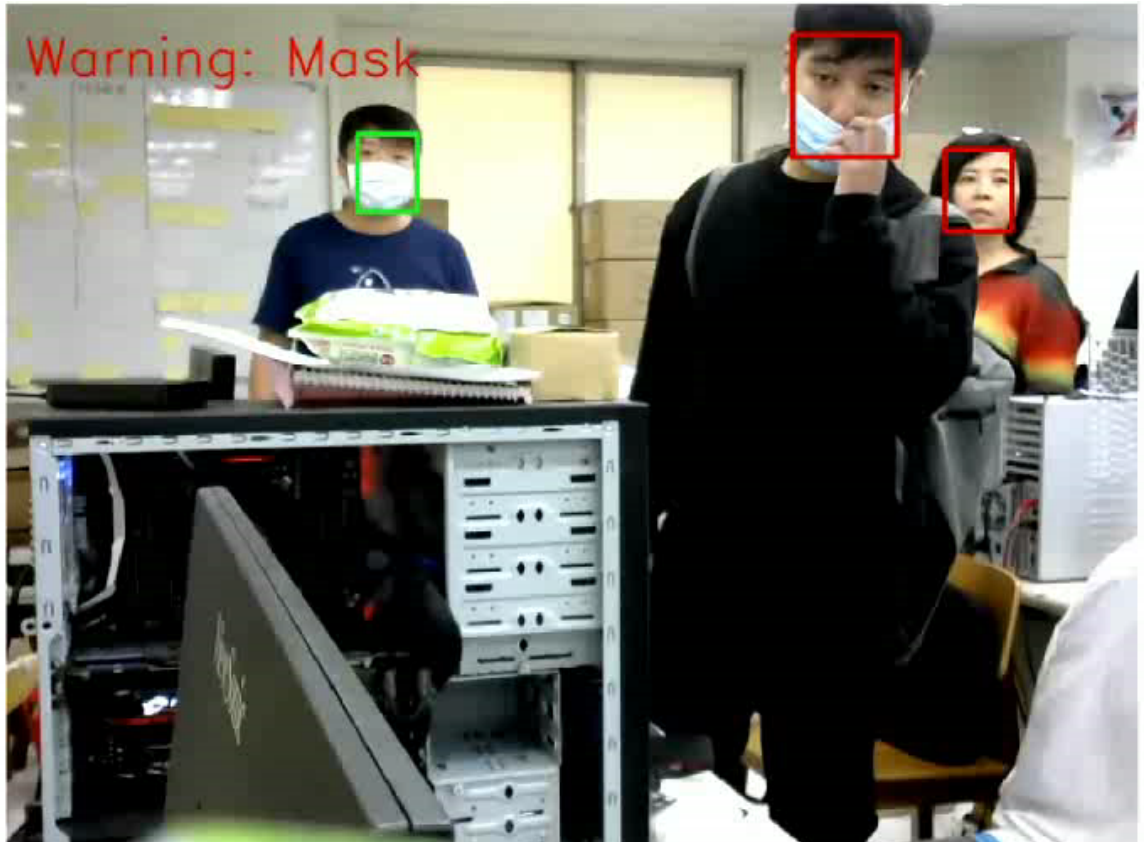
- 辨識不合格的包裝，發出警告後，使用機器手臂推掉
- 協助員工檢查包裝縫線



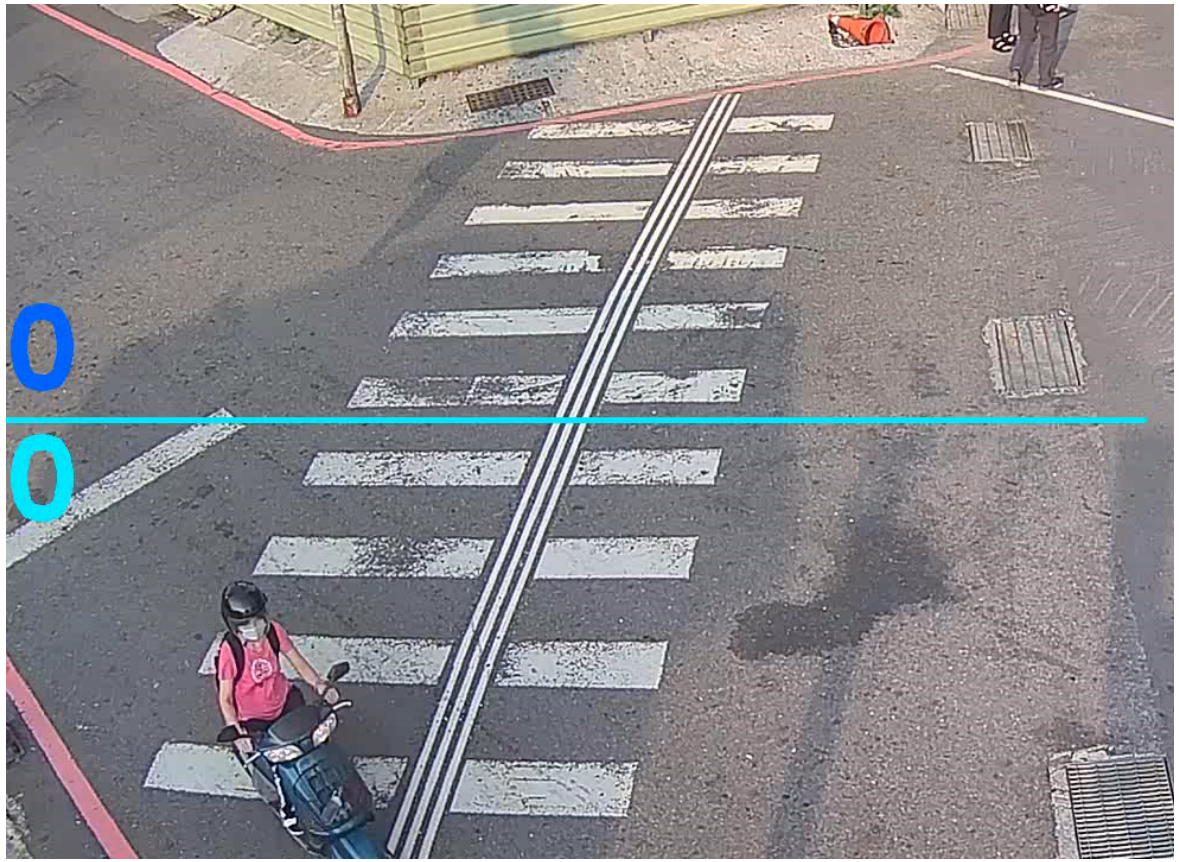
# 場景應用

- 運用在大眾運輸、人多的公共場合
- 可搭配蜂鳴器



# 場景應用

- 計算人流
- 行人違規穿越馬路

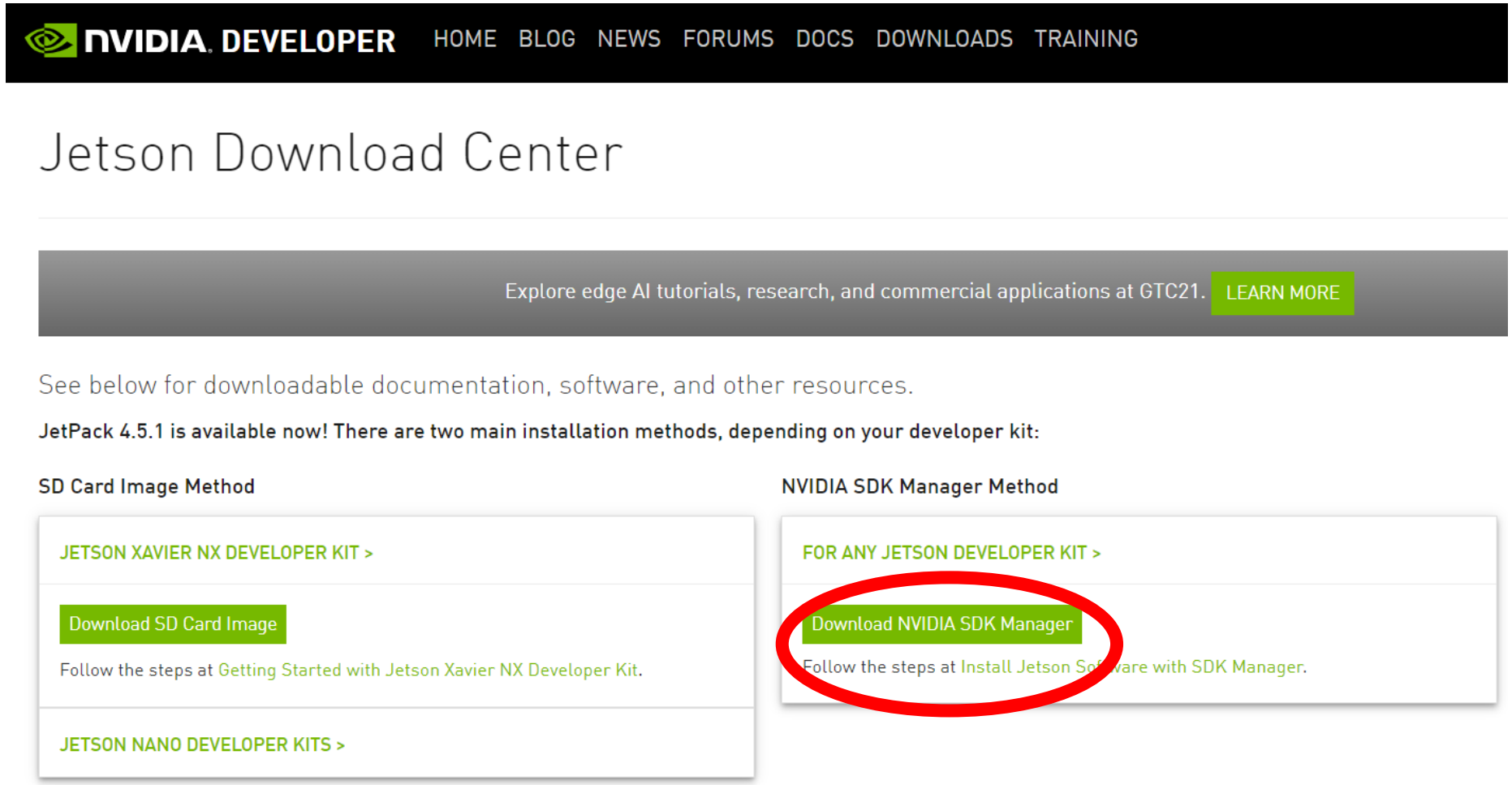




奕瑞科技

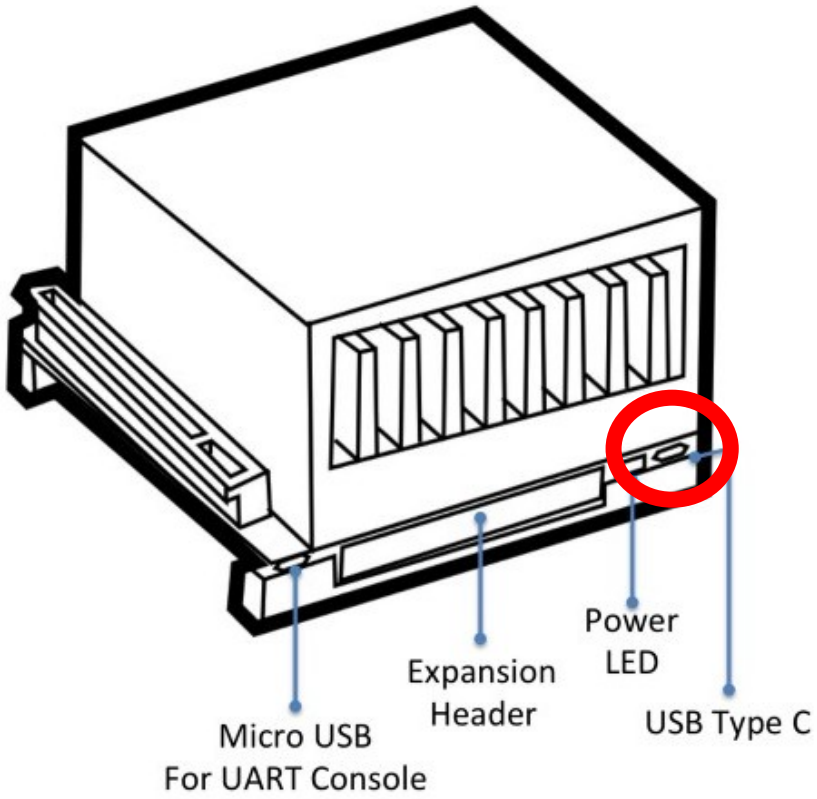
奕瑞科技

# 04. AI BOX小試身手

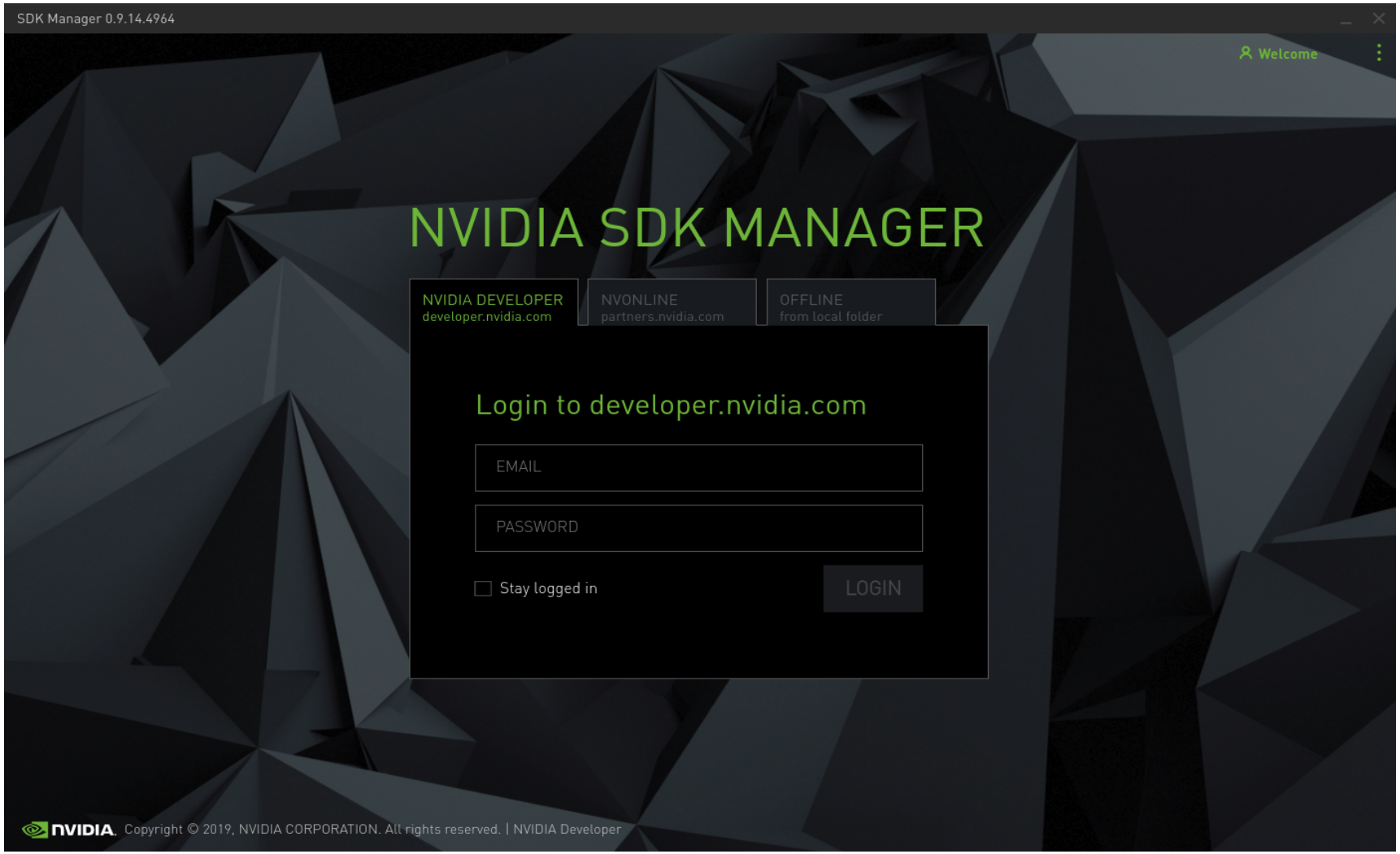
# Xavier 刷機



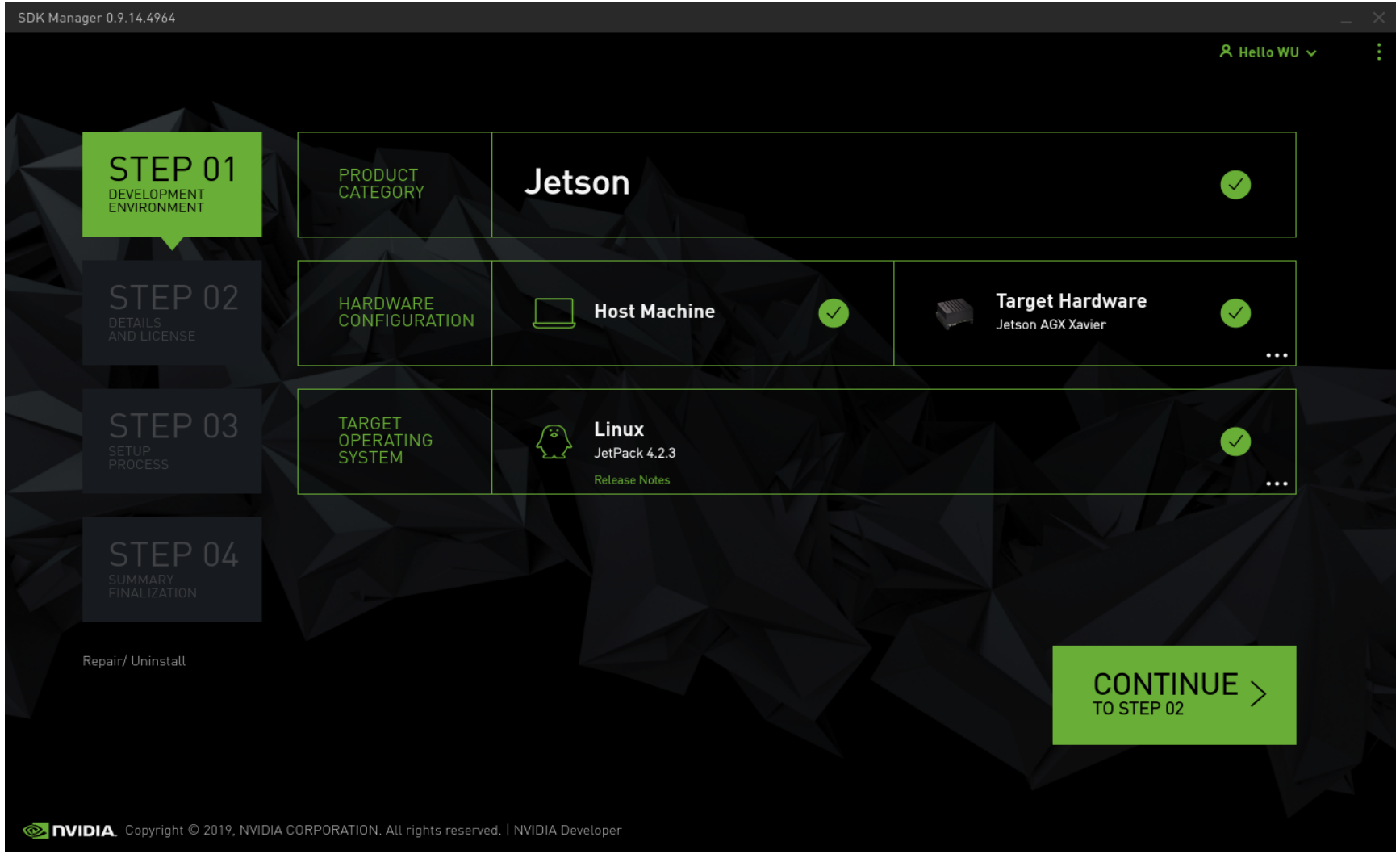
# Xavier 刷機


Micro USB
For UART Console
Expansion
Header
Power
LED
USB Type C

# Xavier 刷機



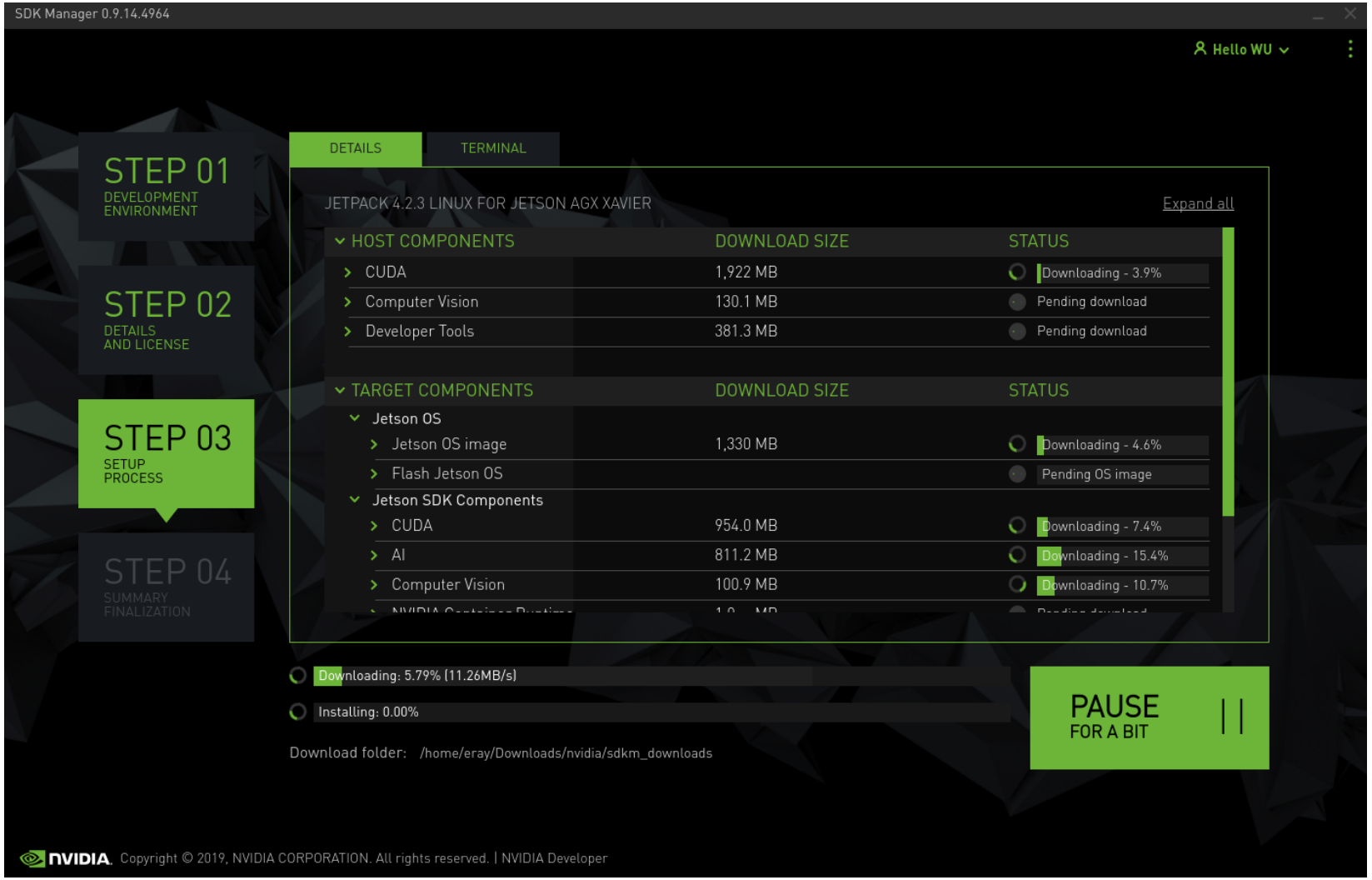
# Xavier 刷機



# Xavier 刷機

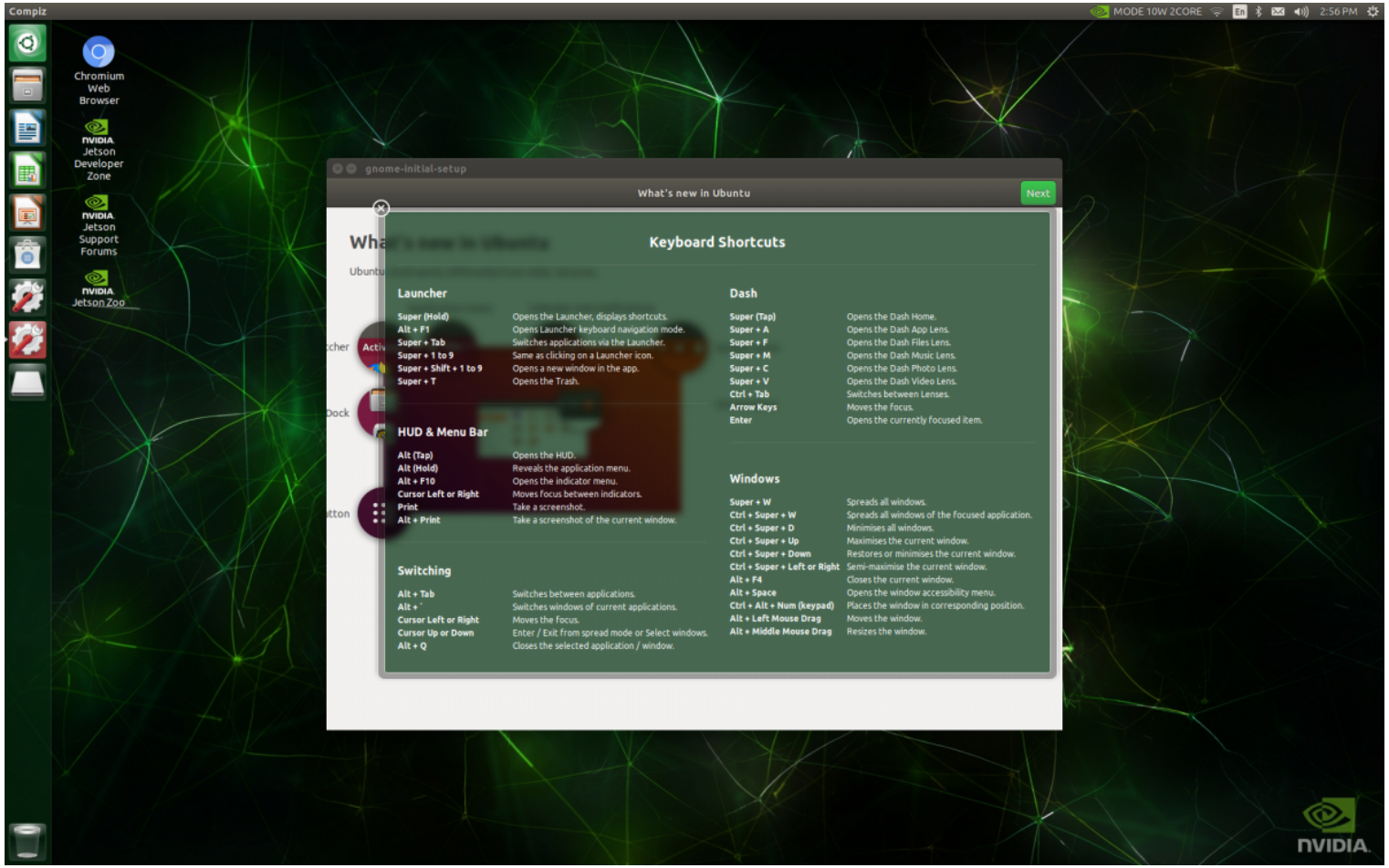

奕瑞科技

# Xavier 刷機

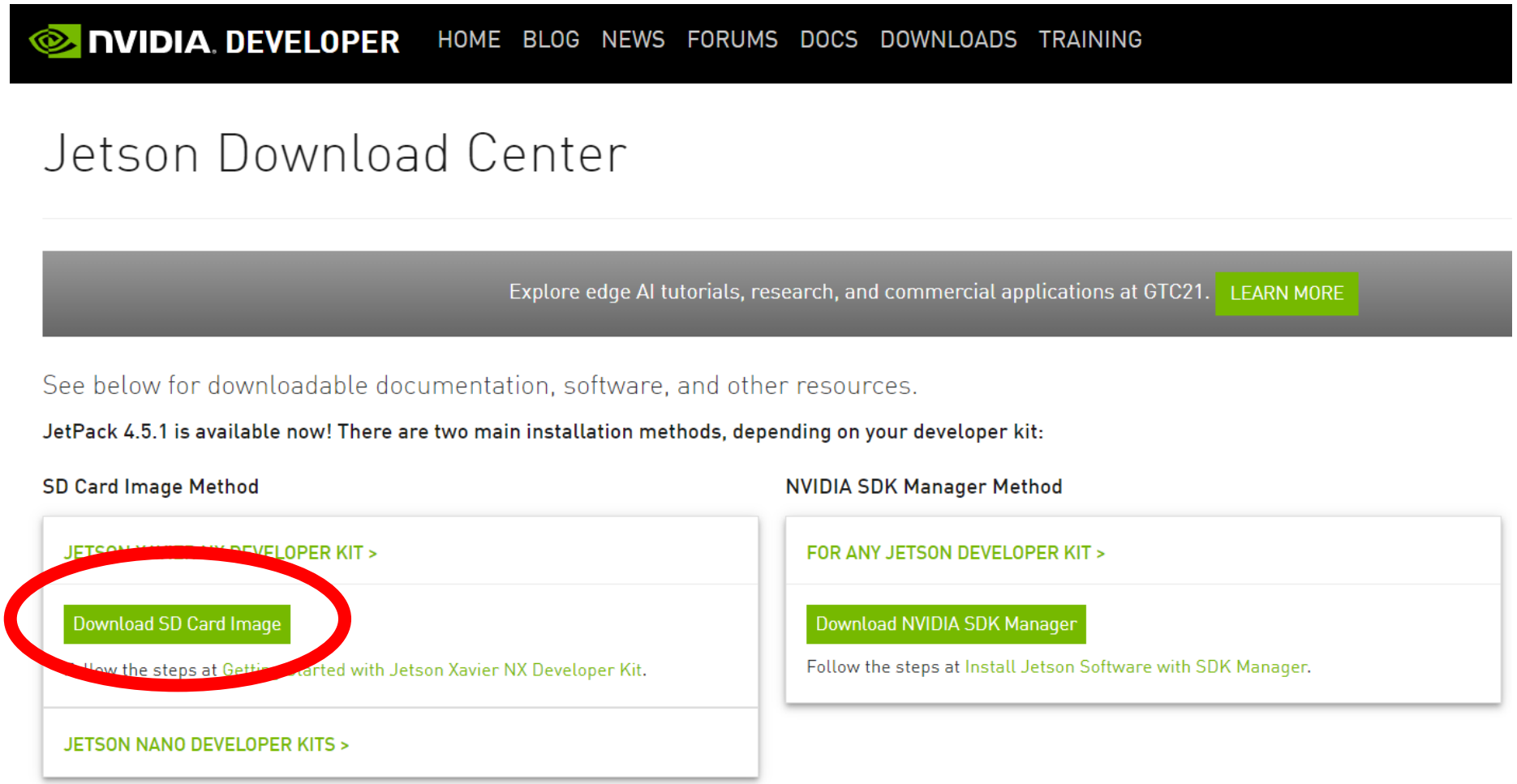



奕瑞科技

# Xavier 刷機



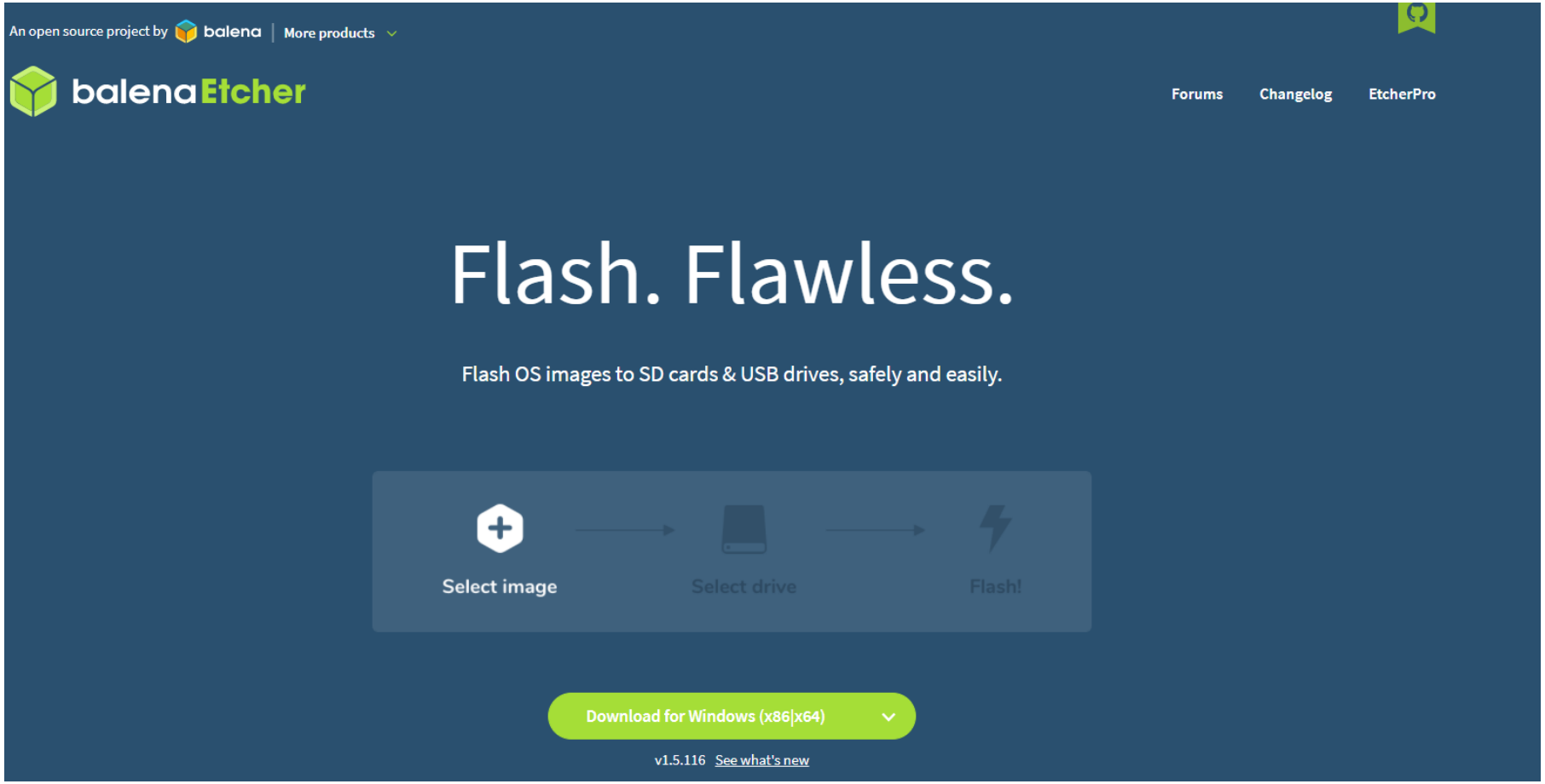
# Xavier NX 刷機



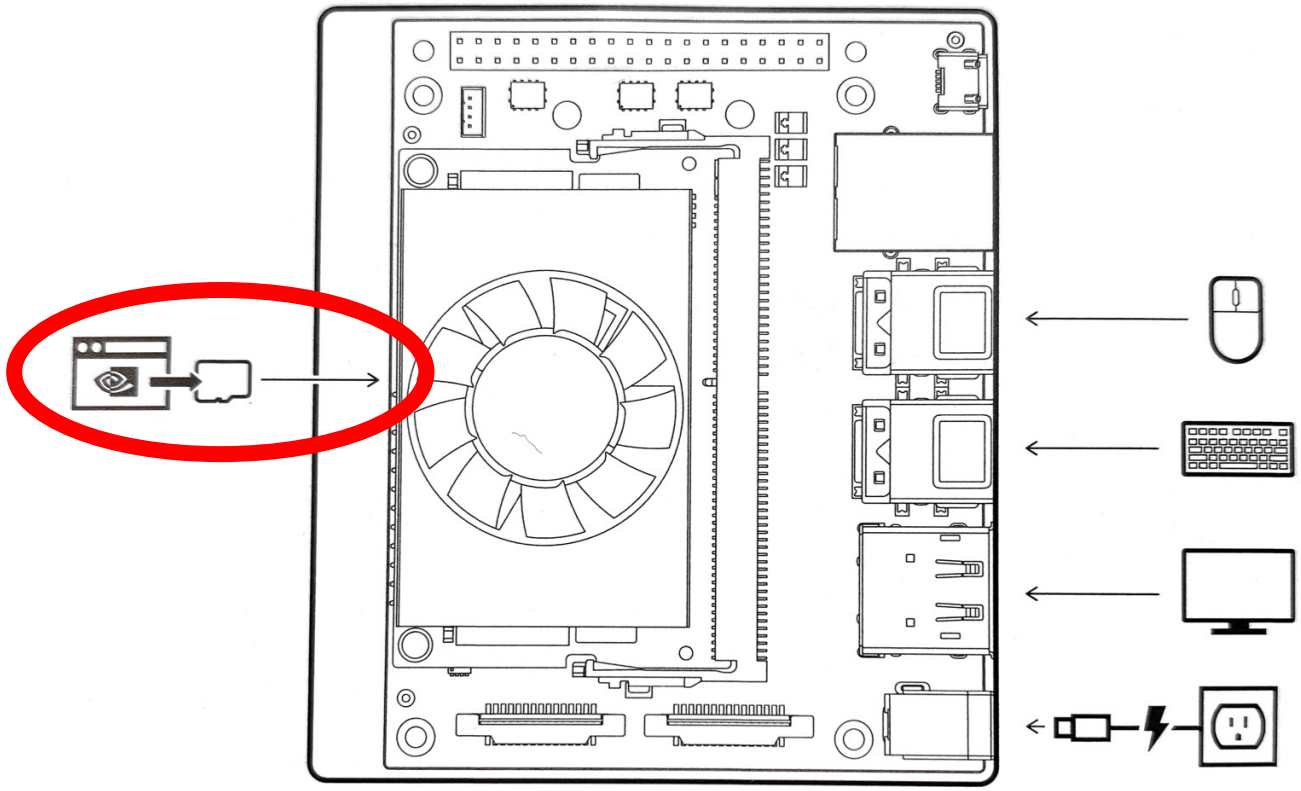
# Xavier NX 刷機


奕瑞科技

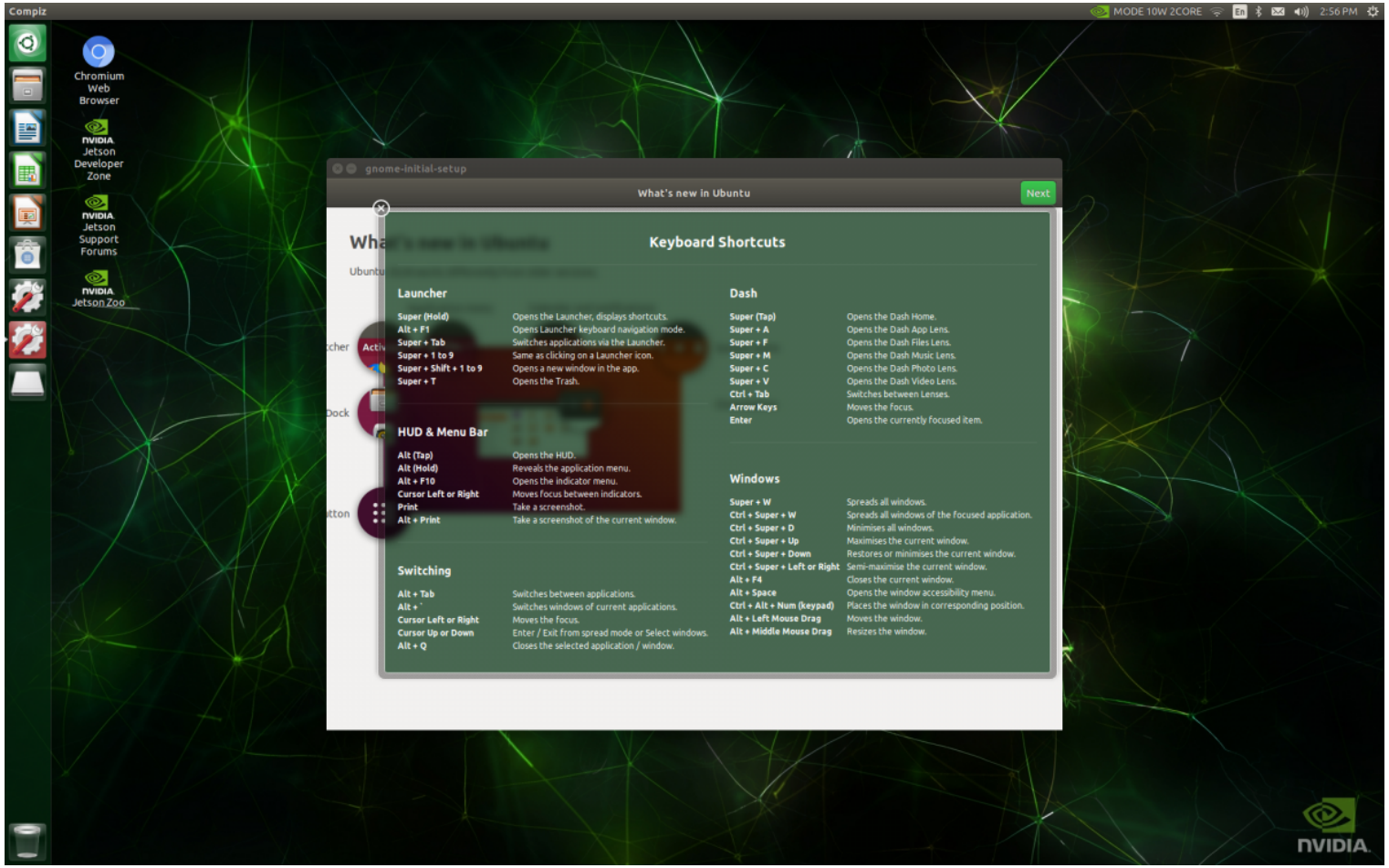
# Xavier NX 刷機


An open source project by balena | More products
balenaEtcher
Forums
Changelog
EtcherPro
Flash. Flawless.
Flash OS images to SD cards & USB drives, safely and easily.
Select image
Select drive
Flash!
Download for Windows (x86|x64)
v1.5.116 See what's new

# Xavier NX 刷機



# Xavier NX 刷機



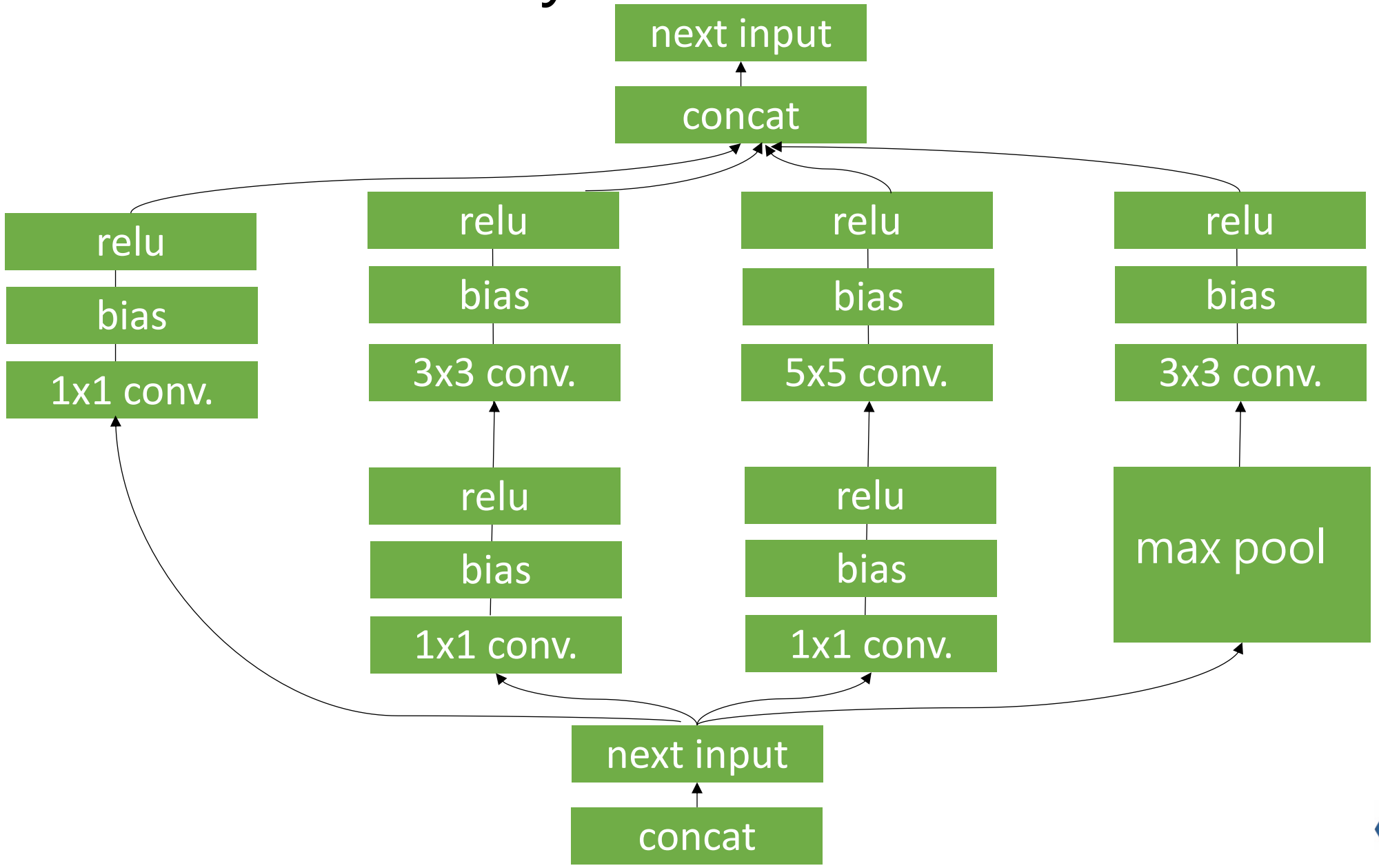
# Xavier 系統相關指令

- 輸入 sudo nvpmodel –m 0 將電力效能全開
- 輸入 sudo jetson_clocks 開啟風扇

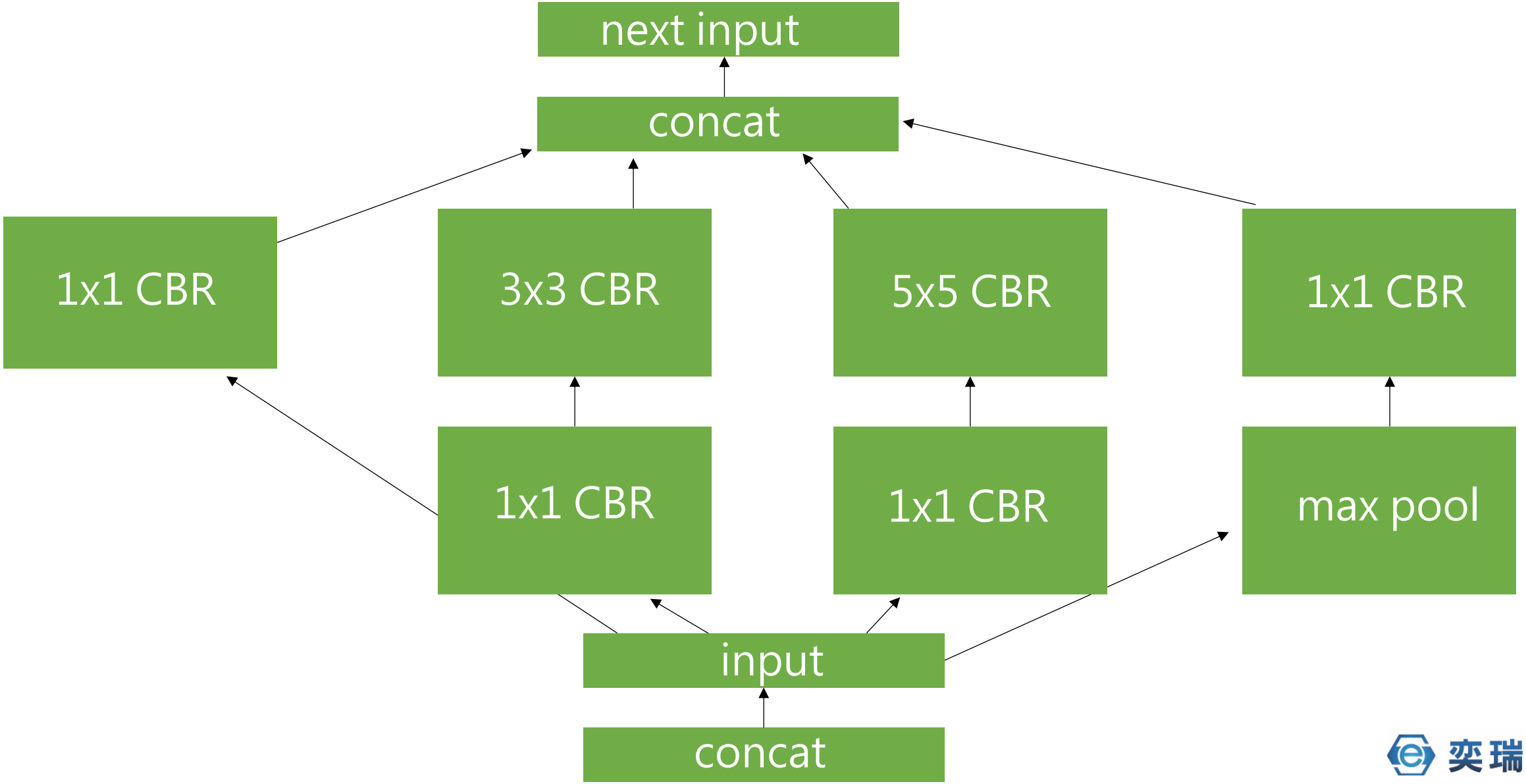
# TensorRT

- 主要將部分layer進行合併，並降低精度以提升速度。
- 優點:
  - 提升辨識速度
  - 節省記憶體使用量
- 缺點:
  - 精確度稍微下降

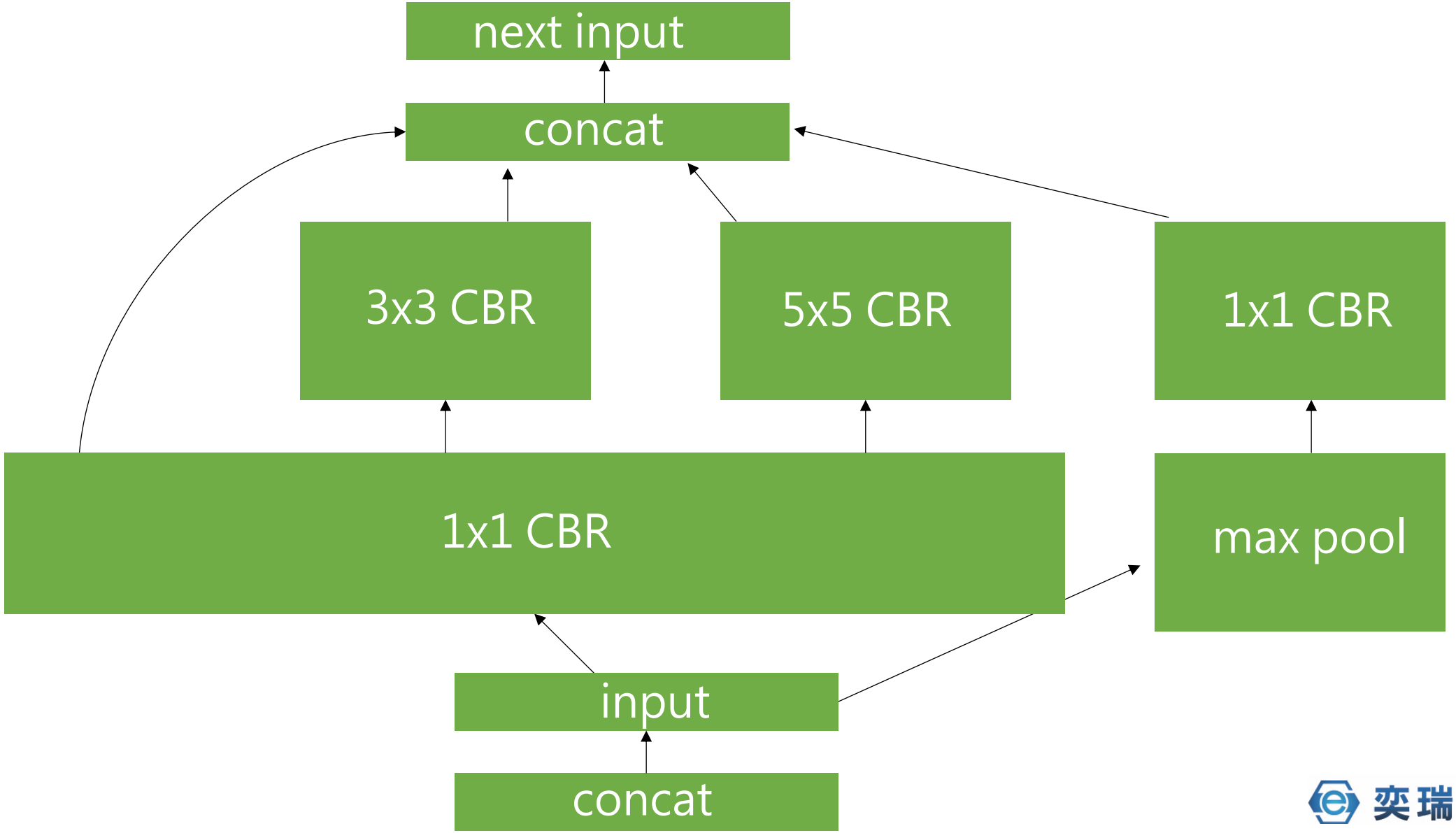
# Fuse network layers Inception structure in GoogleNet



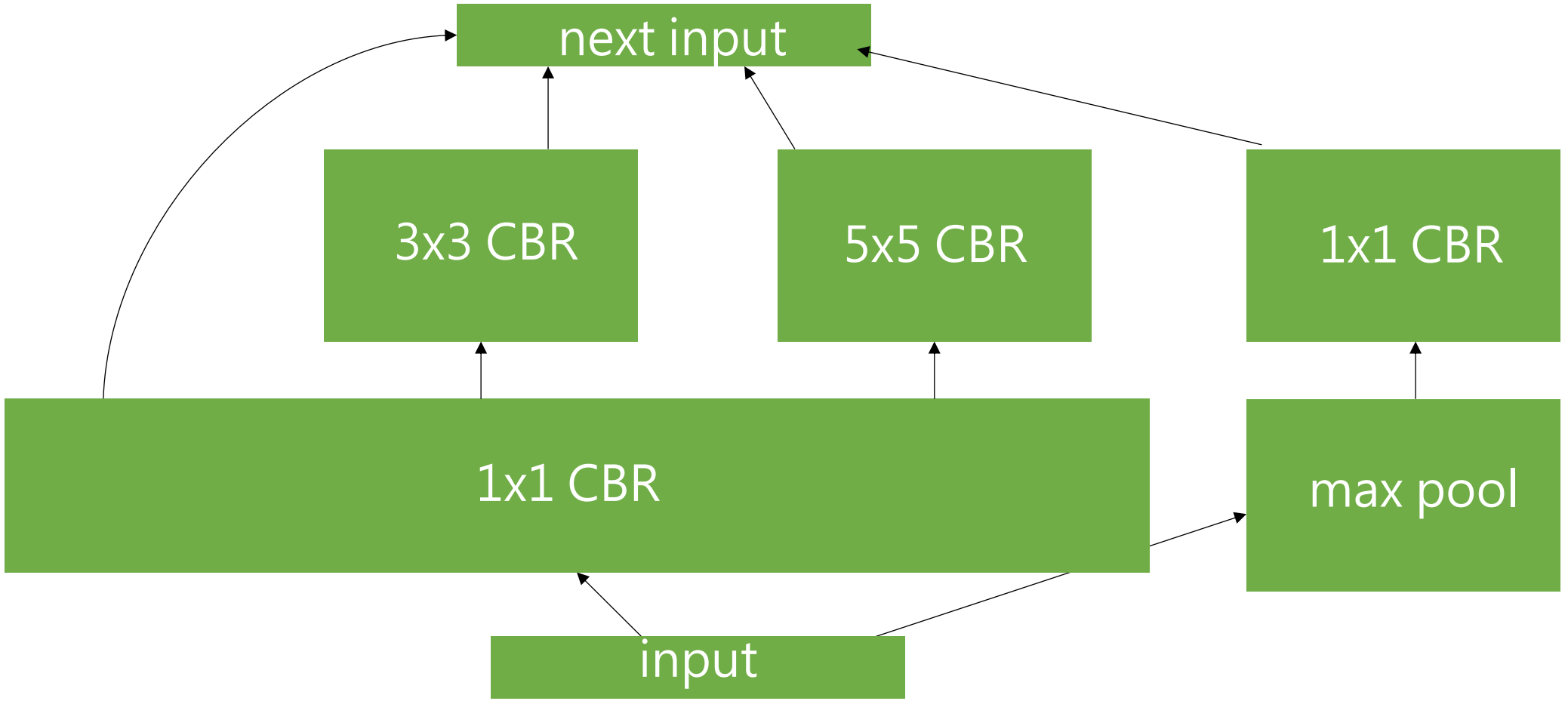
# Fuse network layers Vertical fusion



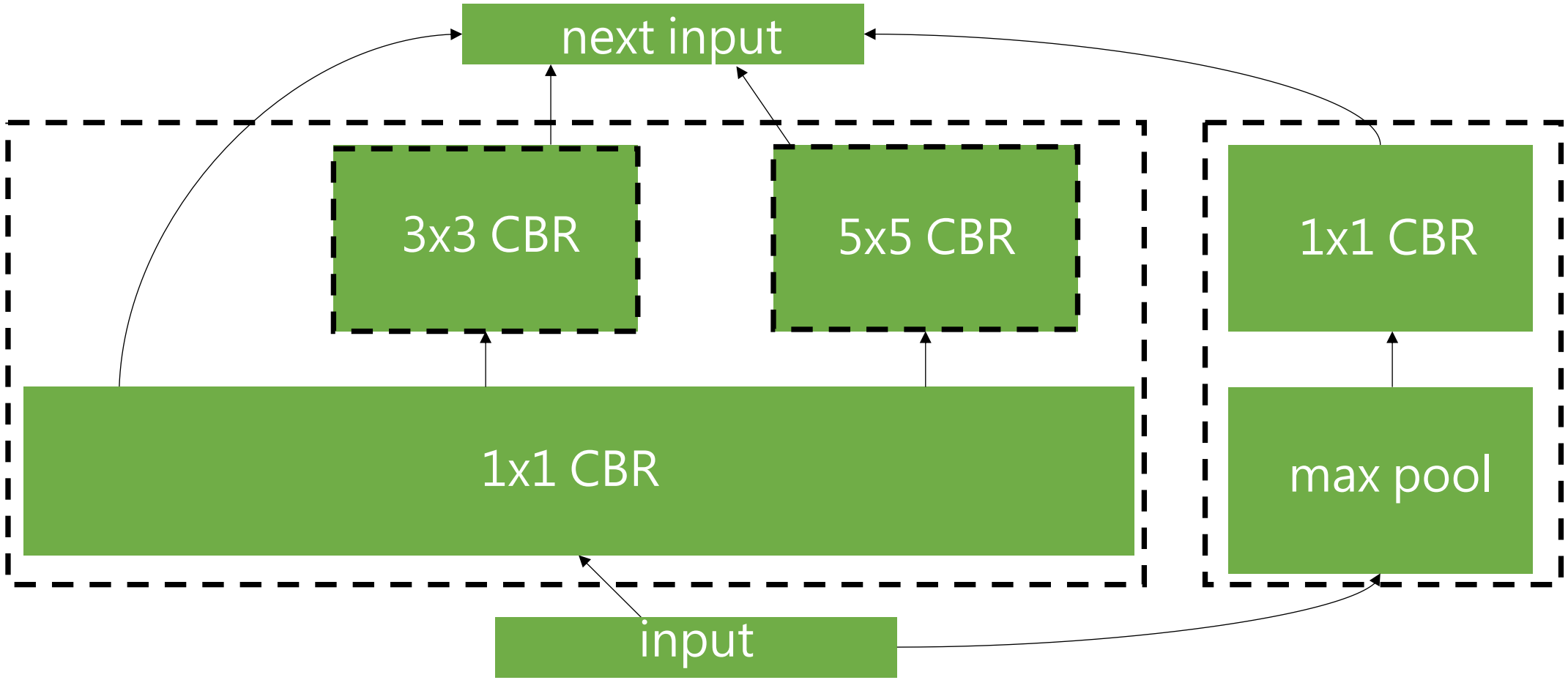
# Fuse network layers Horizontal fusion



# Fuse network layers Concat elision



# Fuse network layers Concurrency



# TensorRT 執行

1. 下載專案
2. 編譯以及安裝相依套件
3. 編譯tensorRT plugins(yolo layer)
4. 將model換成指定的檔案名稱
5. darknet model轉換成onnx
6. 將model轉換成TensorRT engine
7. TensorRT engine推論

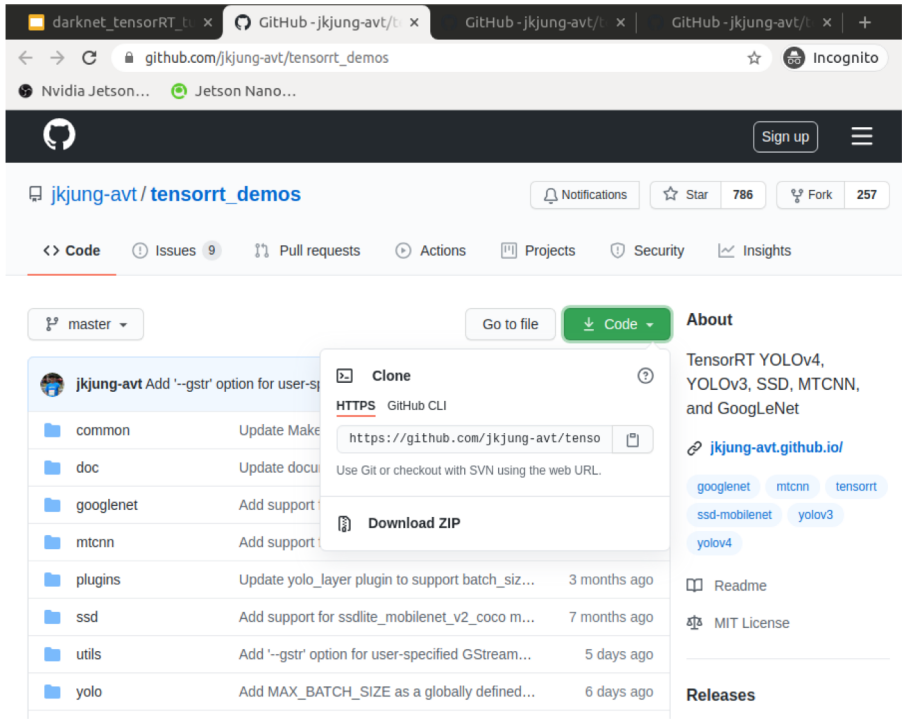
# 下載專案

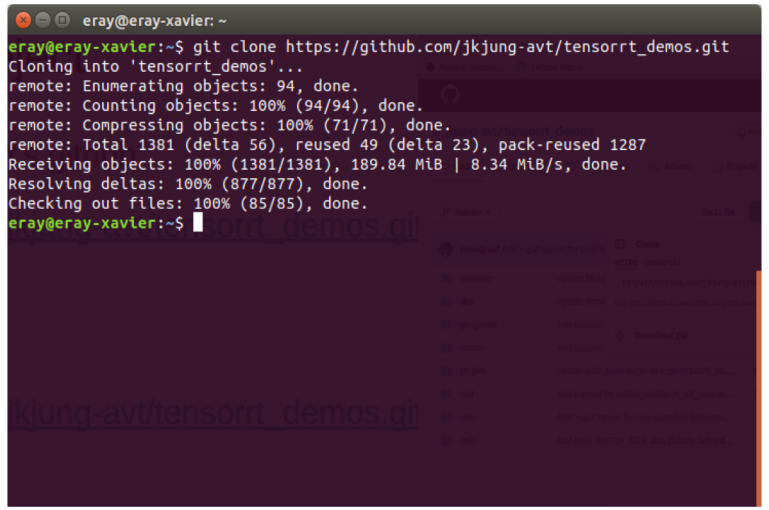
Project’ s github:

https://github.com/jkjung-avt/tensorrt_demos.git

Terminal:

$ git clone https://github.com/jkjung-avt/tensorrt_demos.git



# 編譯以及安裝相依套件

**1.Install protobuf 3.8.0**

```
$ wget https://raw.githubusercontent.com/jkjung-avt/jetson_nano/master/install_protobuf-3.8.0.sh
```

```
$ chmod +x ./install_protobuf-3.8.0.sh; ./install_protobuf-3.8.0.sh
```

**2.Install pycuda**

```
$ cd ${HOME}/project/tensorrt_demos/ssd
```

```
$ ./install_pycuda.sh
```

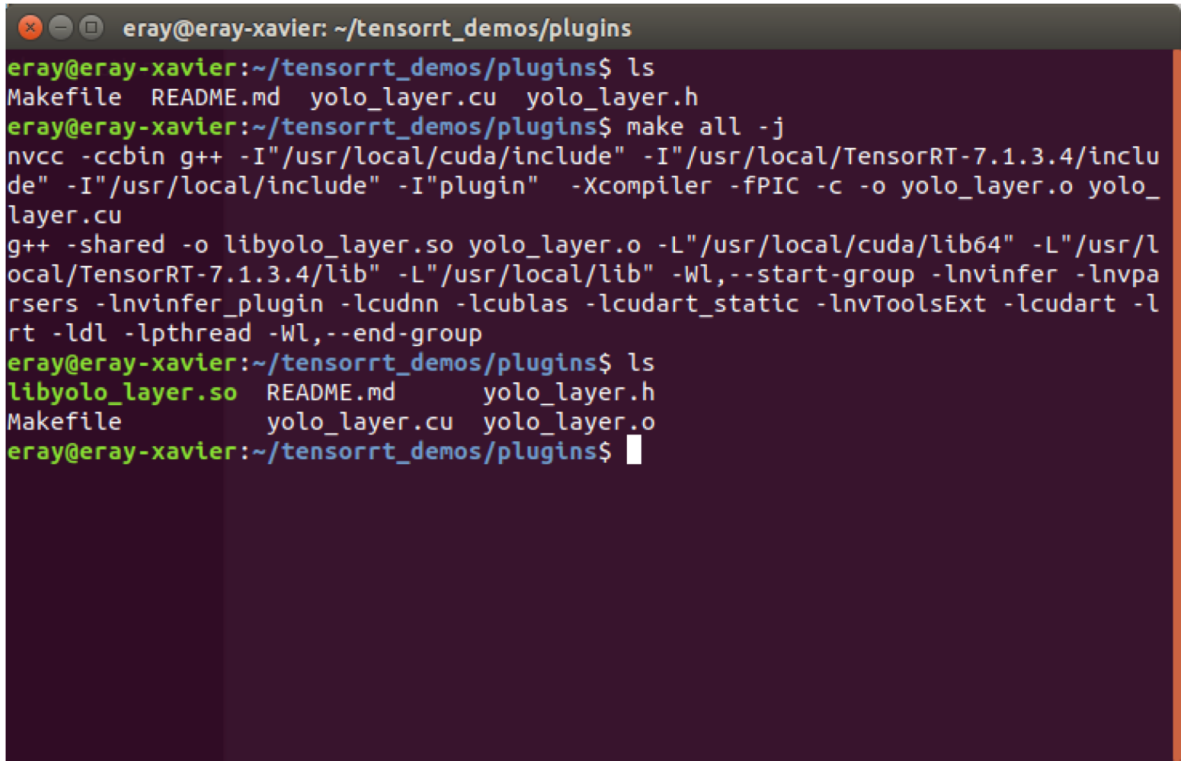
**3.Install onnx**

```
$ sudo pip3 install onnx==1.4.1
```

# 編譯tensorRT plugins(yolo layer)

$ cd ./tensorrt_demos/plugins

$ make all -j



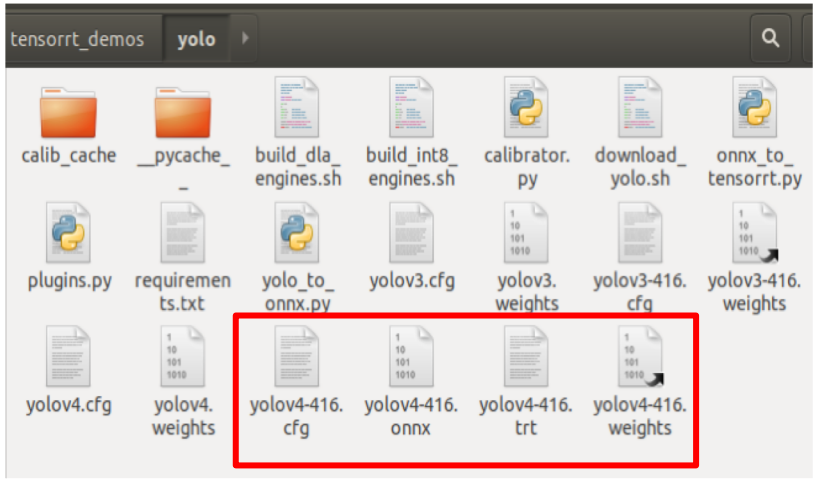
# 將model轉換成TensorRT engine

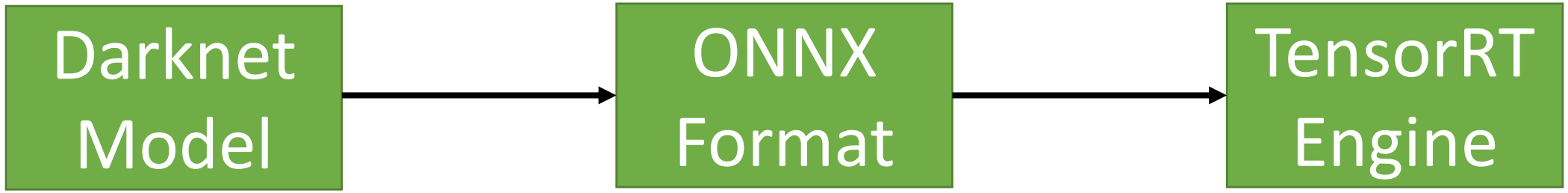
1.Prepare yolo models

2.Convert yolo models to tensorRT engines

```
$ python3 yolo_to_onnx.py -m yolov4-416
```

```
$ python3 onnx_to_tensorrt.py -m yolov4-416
```



# TensorRT engine推論

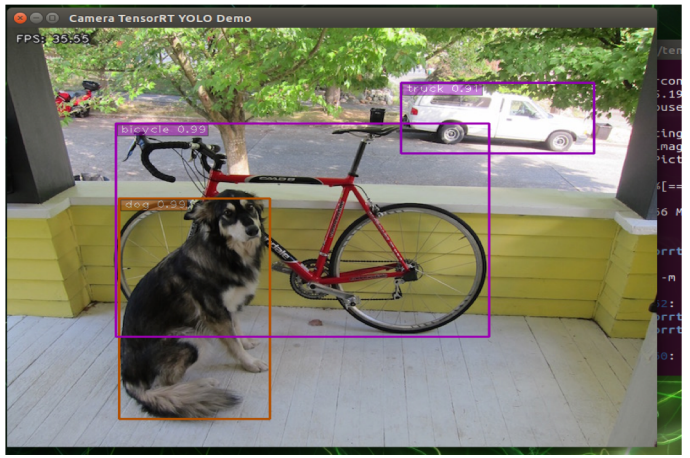
1.open your project directory

2.Prepare image

3.Open terminal and Detect image

```
$ python3 trt_yolo.py --image ${HOME}/Pictures/dog.jpg \  -m yolov4-416
```

p.s tensorRT engine不可以跨平台使用，每個編譯出來的tensorRT engine都會根據平台進行最佳化的動作



# Xavier未使用TensorRT

# Xavier使用TensorRT

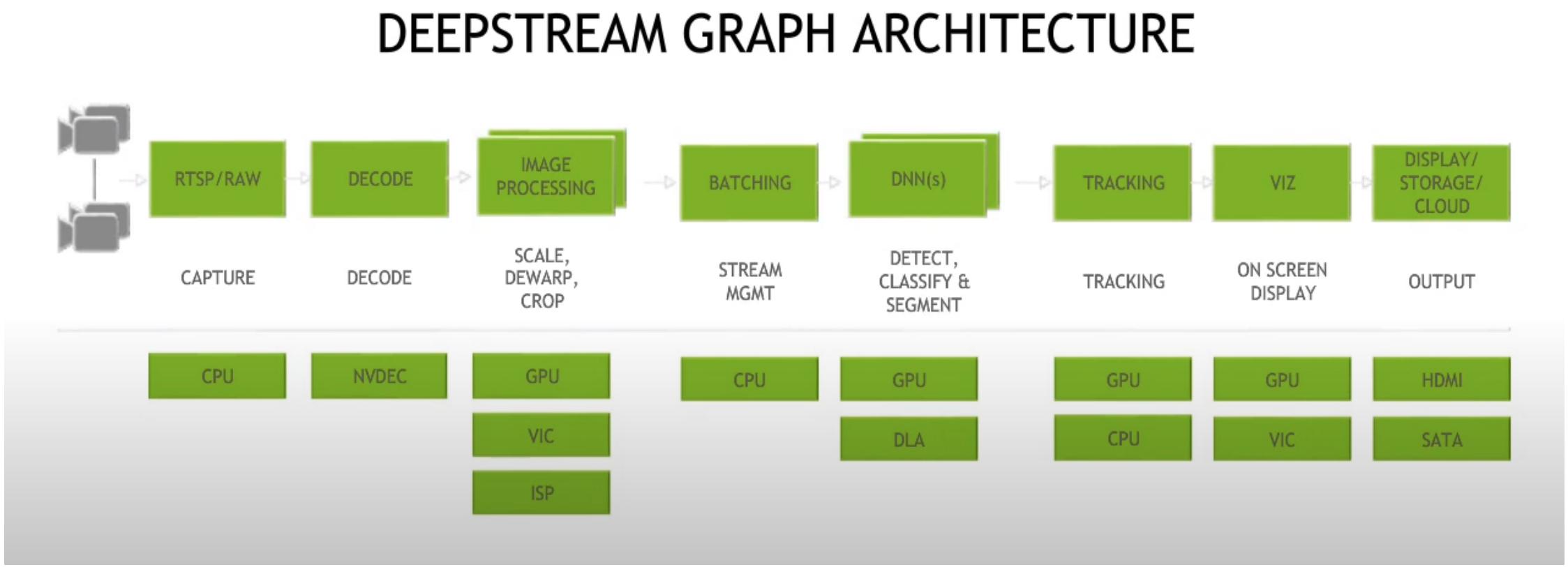
# Xavier使用TensorRT差異

| 未使用TensorRT | 使用TensorRT |
| --- | --- |
| 20 FPS | 36-42 FPS |

# DeepStream

- DeepStream是一套NVIDIA針對Video處理的一個FrameWork，它可以對多個輸入源的解碼、推論、顯示進行非同步及平行處理。
優點：
    1. 使用者只需要專心撰寫推論部份即可。
    2. 所有動作都在GPU上運作，省下CPU傳輸至GPU的時間。
    3. 內建在GPU上運行的追蹤演算法，節省偵測時間。
    4. 透過非同步、及平行處理加速運算效能。
- 缺點：
    1. 若要做較精細的處理，需要了解GStreamer的運作以及撰寫其plugins

# DeepStream



# Deepstream 實作 YOLOV3 Model

# Step

1. 安裝deepStream
2. 編輯檔案(更改class_count, anchors) and 編譯
3. 在要運行的config中指定已經編譯完成的.so的路徑運行程式
4. 開啟python檔案進行設定，

# 編輯檔案(更改class_count,anchors)and 編譯-1

- **Enter to the folder:**
- $ cd /opt/nvidia/deepstream/deepstream/sources/objectDetector_Yolo
- **Download model and config**
- $ sudo chmod+x prebuild.sh; sudo ./prebuild.sh
- **Edit the yolo layer file(class count, anchors)**
- $ sudo vim nvdsinfer_custom_impl_Yolo/nvdsparsebbox_Yolo.cpp
- **Compiler .so**
- $ cd nvdsinfer_custom_impl_Yolo; sudo make all

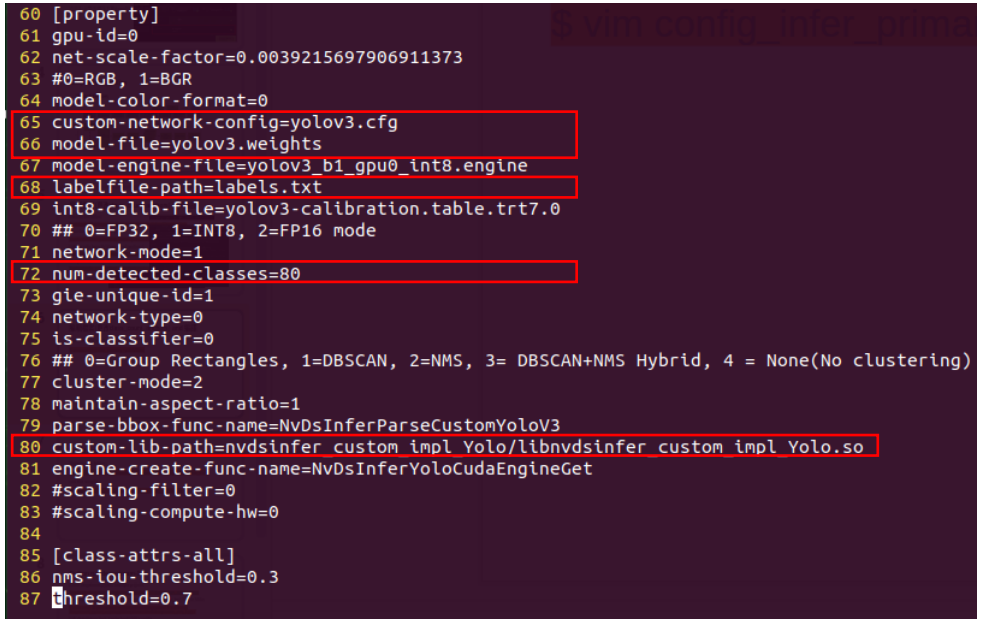
```
606
607 [yolo]
608 mask = 6,7,8
609 anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
610 classes=80
611 num=9
612 jitter=.3
613 ignore_thresh = .7
```

```
33 static const int NUM_CLASSES_YOLO = 80;
```

```
274 /* C-linkage to prevent name-mangling */
275 extern "C" bool NvDsInferParseCustomYoloV3(
276     std::vector<NvDsInferLayerInfo> const& outputLayersInfo,
277     NvDsInferNetworkInfo const& networkInfo,
278     NvDsInferParseDetectionParams const& detectionParams,
279     std::vector<NvDsInferParseObjectInfo>& objectList)
280 {
281     static const std::vector<float> kANCHORS = {
282         10.0, 13.0, 16.0,  30.0,  33.0, 23.0,  30.0,  61.0,
62.0,
283         45.0, 59.0, 119.0, 116.0, 90.0, 156.0, 198.0, 373.0,
326.0};
284     static const std::vector<std::vector<int>> kMASKS = {
285         {6, 7, 8},
286         {3, 4, 5},
287         {0, 1, 2}};
288     return NvDsInferParseYoloV3 (
289         outputLayersInfo, networkInfo, detectionParams,
objectList,
290         kANCHORS, kMASKS);
291 }
```

# 編輯 PGIE(Primary GPU Inference Engines) config

- **Edit the pgie config**
- $ vim config_infer_primary_yoloV3.txt
- **parameter should be changed**
- custom-network-config=
- model-file=
- labelfile-path= (network's class_name)
- num-detected-classes= network's class count
- custom-lib-path= Please enter the yolo layer library you build.



```
[property]
gpu-id=0
net-scale-factor=0.0039215697906911373
#0=RGB, 1=BGR
model-color-format=0
custom-network-config=yolov3.cfg
model-file=yolov3.weights
model-engine-file=yolov3_b1_gpu0_int8.engine
labelfile-path=labels.txt
int8-calib-file=yolov3-calibration.table.trt7.0
## 0=FP32, 1=INT8, 2=FP16 mode
network-mode=1
num-detected-classes=80
gie-unique-id=1
network-type=0
is-classifier=0
## 0=Group Rectangles, 1=DBSCAN, 2=NMS, 3= DBSCAN+NMS Hybrid, 4 = None(No clustering)
cluster-mode=2
maintain-aspect-ratio=1
parse-bbox-func-name=NvDsInferParseCustomYoloV3
custom-lib-path=nvdsinfer_custom_impl_Yolo/libnvdsinfer_custom_impl_Yolo.so
engine-create-func-name=NvDsInferYoloCudaEngineGet
#scaling-filter=0
#scaling-compute-hw=0

[class-attrs-all]
nms-iou-threshold=0.3
threshold=0.7
```

# Test the proejcts

- **Enter to the folder:**
- $ cd /opt/nvidia/deepstream/deepstream/sources/objectDetector_Yolo
- **run the scripts:**
- $ deepstream-app -c deepstream_app_config_yoloV3.txt

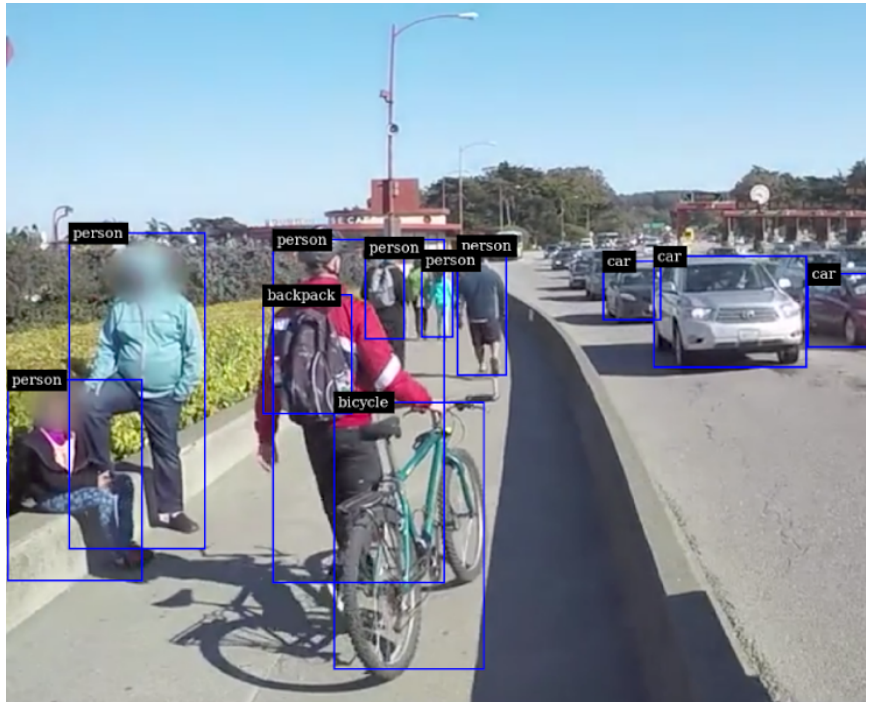
# Python run deepstream-1

- **Clone deepstream-python example projects**
- $ sudo git clone https://github.com/NVIDIA-AI-IOT/deepstream_python_apps
- **Copy apps/common and apps/deepstream-test1/deepstream_test_1.py to folder**
- $ sudo copy -R deepstream_python_apps/apps/common ./; sudo copy deepstream_python_apps/apps/deepstream-test1/deepstream_test_1.py ./
- **Edit deepstream_test_1.py to load PGIE's config**
- $sudo vim deepstream_test_1.py
- **Edit the line 204, change the config-file-path to your config_infer_primary_yoloV3.txt**

```
201     streammux.set_property('height', 1080)
202     streammux.set_property('batch-size', 1)
203     streammux.set_property('batched-push-timeout', 4000000)
204     pgie.set_property('config-file-path', "/opt/nvidia/deepstream/deepstream
    -5.0/sources/objectDetector_Yolo/config_infer_primary_yoloV3.txt")
```

# Python run deepstream-2

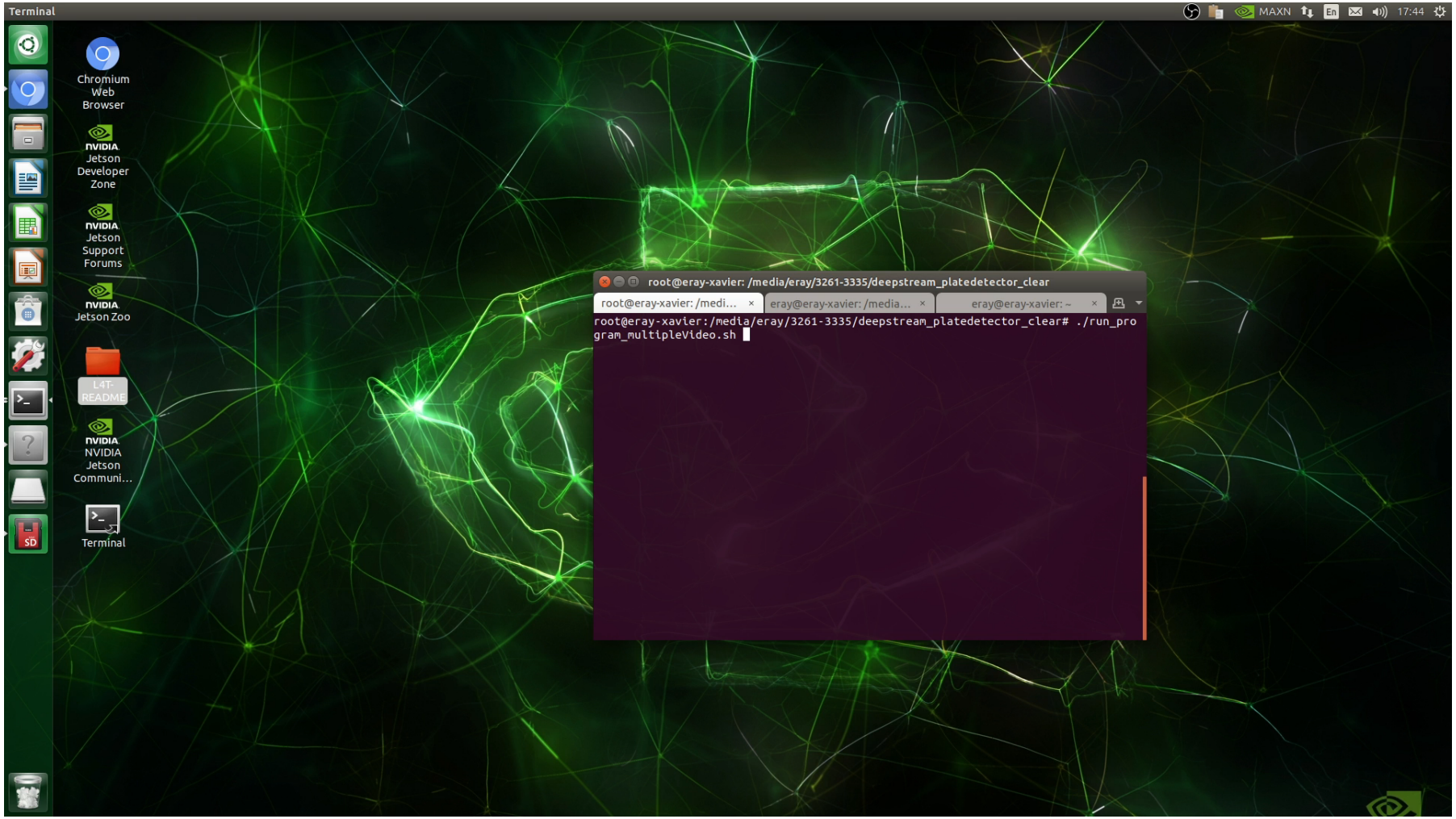
- **Run Code**
- $ sudo python3 deepstream_test_1.py ../../samples/streams/sample_720p.h264
-



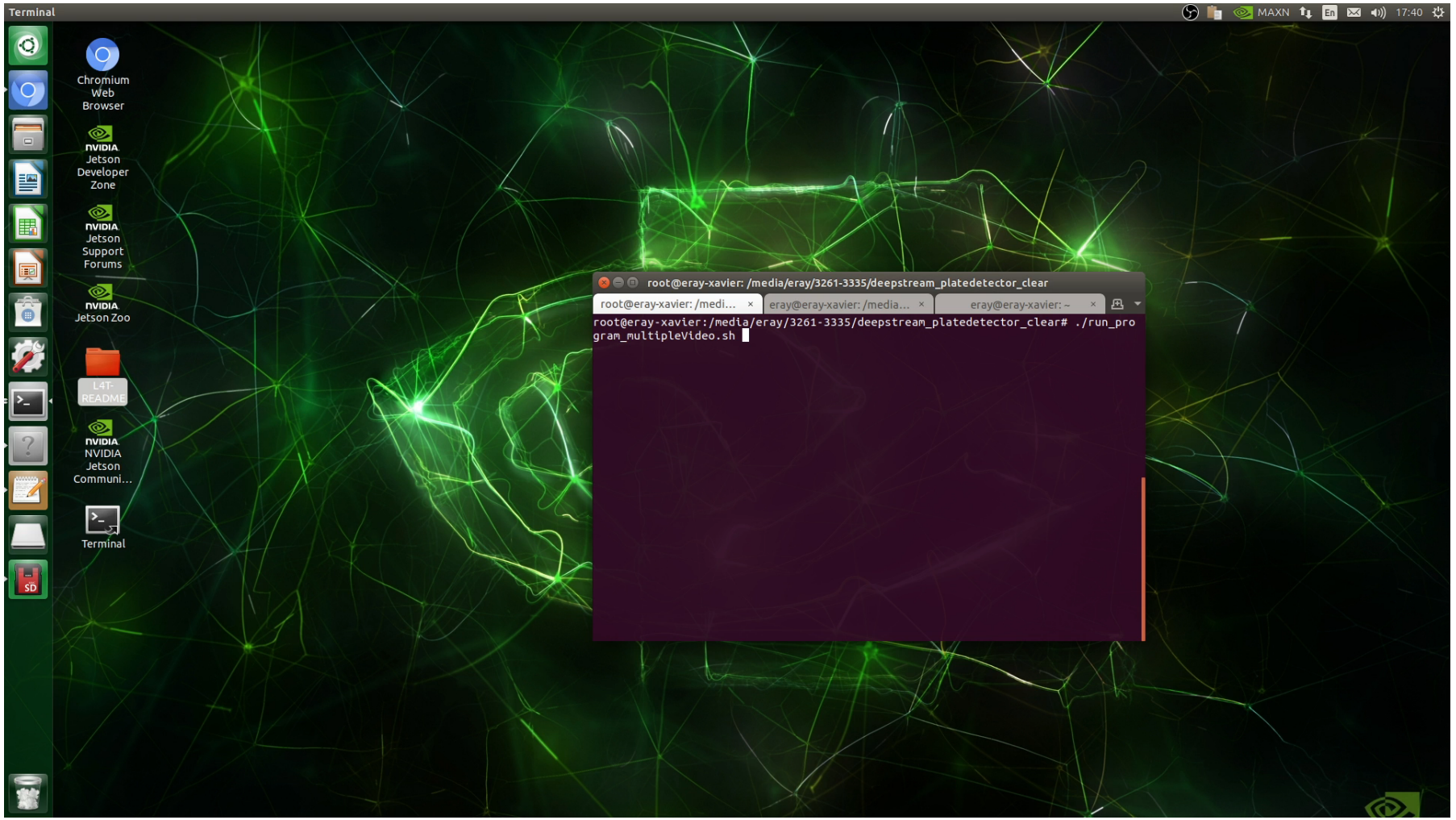

```
eray@eray-xavier: /opt/nvidia/deepstream/deepstream/sources/objectDetector...  ×  eray@eray-xavier: /opt/nvidia/
41 def osd_sink_pad_buffer_probe(pad,info,u_data):
42     frame_number=0
43     #Intiallizing object counter with 0.
44     obj_counter = {
45         PGIE_CLASS_ID_VEHICLE:0,
46         PGIE_CLASS_ID_PERSON:0,
47         PGIE_CLASS_ID_BICYCLE:0,
48         PGIE_CLASS_ID_ROADSIGN:0
49     }
50     num_rects=0
51
52     gst_buffer = info.get_buffer()
53     if not gst_buffer:
54         print("Unable to get GstBuffer ")
55         return
56
57     # Retrieve batch metadata from the gst_buffer
58     # Note that pyds.gst_buffer_get_nvds_batch_meta() expects the
59     # C address of gst_buffer as input, which is obtained with hash(gst_buffer)
60     batch_meta = pyds.gst_buffer_get_nvds_batch_meta(hash(gst_buffer))
61     l_frame = batch_meta.frame_meta_list
62     while l_frame is not None:
63         try:
64             # Note that l_frame.data needs a cast to pyds.NvDsFrameMeta
65             # The casting is done by pyds.glist_get_nvds_frame_meta()
66             # The casting also keeps ownership of the underlying memory
67             # in the C code, so the Python garbage collector will leave
68             # it alone.
69             #frame_meta = pyds.glist_get_nvds_frame_meta(l_frame.data)
70             frame_meta = pyds.NvDsFrameMeta.cast(l_frame.data)
71         except StopIteration:
72             break
73
74         frame_number=frame_meta.frame_num
75         num_rects = frame_meta.num_obj_meta
76         l_obj=frame_meta.obj_meta_list
77         while l_obj is not None:
78             try:
79                 # Casting l_obj.data to pyds.NvDsObjectMeta
80                 #obj_meta=pyds.glist_get_nvds_object_meta(l_obj.data)
81                 obj_meta=pyds.NvDsObjectMeta.cast(l_obj.data)
82             except StopIteration:
83                 break
84             if obj_meta.class_id < 4:
85                 obj_counter[obj_meta.class_id] += 1
86             obj_meta.rect_params.border_color.set(0.0, 0.0, 1.0, 0.0)
87             try:
88                 l_obj=l_obj.next
89             except StopIteration:
90                 break
91
92         # Acquiring a display meta object. The memory ownership remains in
93         # the C code so downstream plugins can still access it. Otherwise
94         # the garbage collector will claim it when this probe function exits.
95         display_meta=pyds.nvds_acquire_display_meta_from_pool(batch_meta)
96         display_meta.num_labels = 1
:set nu
```

# Xavier未使用DeepStream

# Xavier使用DeepStream(不含追蹤)



# Xavier使用DeepStream(含追蹤)




奕瑞科技

# Xavier使用DeepStream差異

| 未使用DeepStream | 使用 DeepStream (不含追蹤) | 使用 DeepStream (含追蹤) |
| --- | --- | --- |
| 16-20 FPS | 54 FPS | 60 FPS |

# Xavier使用TensorRT & DeepStream差異

| 未使用 DeepStream | 使用TensorRT | 使用 DeepStream (不含追蹤) | 使用 DeepStream (含追蹤) |
|---|---|---|---|
| 16-20 FPS | 36-42 FPS | 54 FPS | 60 FPS |

# Q&A

THANK YOU

奕瑞科技

奕瑞科技