## 🚀 第三部分：AI 部署流程概念（30 分鐘）

### 🎯 教學目標

了解模型從訓練到部署的完整流程

掌握 ONNX 與 TensorRT 的角色與優勢

瞭解 CPU / GPU / TensorRT 執行效能比較

準備下一階段：模型實際部署演練

## 🔹 一、模型部署的整體流程架構

AI 部署從模型訓練到邊緣設備推論，主要分為以下幾個階段：

模型訓練（雲端／PC）

↓

模型導出（PyTorch / TensorFlow）

↓

模型轉換（→ ONNX）

↓

優化部署（→ TensorRT）

↓

Jetson 執行推論（C++ / Python）

📌 **ONNX（Open Neural Network Exchange）**

一種中立格式，可以將模型在不同平台間轉換與共享。

PyTorch / TensorFlow / Keras → ONNX → Jetson 運行

📌 **TensorRT（NVIDIA 深度學習推論引擎）**

專為 NVIDIA GPU 最佳化的高效推論工具

提供 layer fusion、weight quantization（FP32 → FP16 / INT8）等加速技術

可以直接載入 ONNX 模型產生 .engine 檔，在 Jetson 上運行

## 🔹 二、範例流程說明（以 PyTorch 模型為例）

### 🛠 第 1 步：PyTorch 模型導出

import torch

import torchvision.models as models

model = models.resnet18(pretrained=True)

dummy\_input = torch.randn(1, 3, 224, 224)

torch.onnx.export(model, dummy\_input, "resnet18.onnx")

### 🛠 第 2 步：轉換為 TensorRT 引擎（透過 trtexec 或 Python API）

/usr/src/tensorrt/bin/trtexec --onnx=resnet18.onnx --saveEngine=resnet18.engine

或使用 Python：

import tensorrt as trt

# 使用 ONNX parser 轉換成 TensorRT Engine

### 🛠 第 3 步：在 Jetson 上推論

使用 Python + OpenCV 接收影像並進行推論：

# 推論前處理 → TensorRT 執行 → 後處理

## 🔹 三、部署方式比較：效能、功耗與彈性

| **模型執行方式** | **優點** | **缺點** | **使用場景** |
| --- | --- | --- | --- |
| CPU | 安裝簡單、免 GPU | 效能慢 | 雲端伺服器驗證 |
| GPU (CUDA) | 中等效能 | 使用資源較多 | 即時推論 |
| TensorRT | 極速、低延遲、節能 | 轉換較繁瑣 | Jetson 邊緣裝置 |

🧪 **效能測試建議：**  
使用 time.perf\_counter() 或 trtexec 測試推論時間

trtexec --onnx=resnet18.onnx --fp16

## 🔹 四、最佳實踐與注意事項

| **項目** | **建議** |
| --- | --- |
| 模型格式 | 優先使用 ONNX 格式 |
| 輸入大小 | 與原始模型訓練設定一致 |
| 模型精度 | 使用 FP16 / INT8 可提升效能但須驗證準確率 |
| 資源限制 | 注意 GPU memory，避免超出 Jetson 記憶體限制 |
| 調用方式 | 可使用 C++ 或 Python，但部署建議使用 Python 搭配 OpenCV |

## 📌 小結

AI 模型部署的核心在於「**轉換與加速**」：PyTorch → ONNX → TensorRT

TensorRT 可將模型極速化、低功耗化，非常適合 Jetson 執行

教師可將此流程應用於學生專題：如手勢辨識、人臉分類、智慧物件追蹤等