

淡江大學電機工程學系專題

指導教授：江正雄博士

延遲導向輕量化 YOLO 於邊緣裝置之即時影像感知

Latency-Oriented Lightweight YOLO for
Real-Time Visual Perception on Edge
Devices

學生：潘信瑞
詹盛惟
沈群博

中華民國 115 年 10 月

摘要

隨著物件偵測模型逐漸應用於智慧監控、機器人與嵌入式視覺系統，如何在有限運算資源下兼顧偵測精度與即時推論能力，成為邊緣人工智慧的重要課題。模型參數量與浮點運算量（FLOPs）雖常被用於評估模型效率，但實際推論時間仍受到算子種類、資料存取、推論框架及硬體架構等因素影響，因此理論計算量的降低未必能直接轉換為等比例的部署加速。本研究以 Raspberry Pi 5 的 ARM CPU 為部署平台，以 YOLO26n 為基準模型，探討輕量化架構修改對偵測精度、計算量及實際推論延遲的影響。本研究設計 Cross-stage Feature Reuse（CFR）與 Lightweight Downsampling（LDS）兩種模型變體。CFR 結合 Ghost bottleneck 與分支特徵融合，以改善特徵利用；LDS 則以 GhostConv 接續 Efficient Channel Attention（ECA），取代部分下採樣卷積，以降低計算成本。三個主要模型皆於 PASCAL VOC 資料集進行三個隨機種子、100 epochs 的訓練，並以 mAP50-95 評估精度，完成後分別匯出為 ONNX 與 NCNN 格式，在 Raspberry Pi 5 上量測 FP32 推論時間。實驗結果顯示，CFR 的平均 mAP50-95 為 0.5014，相較 baseline 的 0.4854 提升 1.61 個百分點，但 NCNN 推論延遲由 61.11 ms 增加至 74.74 ms。LDS 的平均 mAP50-95 為 0.4851，與 baseline 接近，而匯出態 FLOPs 減少 10.9%，NCNN 推論延遲降低 4.9% 至 58.14 ms。三個主要模型於 NCNN 流程下的推論時間約為 ONNX 流程的一半，但兩者亦存在匯出計算圖與輸出分支差異。結果顯示，降低 FLOPs 並不足以直接預測實際部署速度，模型設計應同時考量偵測精度、實測延遲、推論流程及目標硬體特性，以建立較符合實際應用需求的輕量化模型。

關鍵詞：物件偵測、YOLO26、特徵複用、輕量下採樣、邊緣運算、推論延遲。

Abstract

As object detection models are increasingly deployed in intelligent surveillance, robotics, and embedded vision systems, achieving a balance between detection accuracy and real-time inference efficiency has become an important issue in edge artificial intelligence. Although parameter count and floating-point operations (FLOPs) are commonly used to evaluate model efficiency, actual inference latency is also affected by operator characteristics, data movement, inference frameworks, and hardware architecture. Therefore, reductions in theoretical computation do not necessarily translate into proportional deployment speedups. This study uses the ARM CPU of a Raspberry Pi 5 as the target platform and YOLO26n as the baseline model to investigate the effects of lightweight architectural modifications on detection accuracy, computational cost, and measured inference latency. Two model variants are developed: Cross-stage Feature Reuse (CFR) and Lightweight Downsampling (LDS). CFR combines Ghost bottlenecks with branch-based feature fusion to improve feature utilization, while LDS replaces selected downsampling convolutions with GhostConv followed by Efficient Channel Attention (ECA) to reduce computational cost. The three main models are trained on the PASCAL VOC dataset for 100 epochs using three random seeds and evaluated using mAP50–95. The trained models are then exported to ONNX and NCNN formats for FP32 inference measurements on the Raspberry Pi 5. Experimental results show that CFR achieves a mean mAP50–95 of 0.5014, improving upon the baseline value of 0.4854 by 1.61 percentage points, while increasing NCNN inference latency from 61.11 ms to 74.74 ms. LDS achieves a mean mAP50–95 of 0.4851, close to the baseline, while reducing exported FLOPs by 10.9% and NCNN inference latency by 4.9% to 58.14 ms. The three main models also require approximately half as much inference time in the NCNN workflow as in the ONNX workflow, although the two pipelines differ in their exported graphs and output branches. These results demonstrate that FLOPs alone are insufficient to predict practical deployment speed, and that model selection for edge devices should consider detection accuracy, measured latency, deployment workflow, and target hardware together.

Keywords: object detection, YOLO26, feature reuse, lightweight downsampling, edge computing, inference latency.

目錄

摘要.....	I
Abstract	II
第一章 緒論.....	1
1.1 背景介紹	1
1.2 實驗動機	1
1.3 研究目標與範圍	1
1.4 研究問題與比較層次	2
1.5 研究成果與文件範圍	2
第二章 相關文獻.....	3
2.1 延遲導向模型設計	3
2.2 YOLO26 與部署分支	3
2.3 GhostConv 與 ECA 通道注意力	3
2.4 NCNN 與既有 Pi 5 量測	4
2.5 輕量化卷積與特徵路徑	4
2.6 通道注意力與模型規模	4
2.7 硬體效率與研究定位	5
2.8 卷積成本與 Ghost 特徵生成.....	5
2.9 通道重加權與特徵保留	6
2.10 模型成本與部署時間的關係	7
第三章 作品介紹.....	8
3.1 系統流程與模型配置	8
3.2 CFR 特徵複用模組.....	9
3.3 LDS 輕量下採樣模組.....	9
3.4 位置消融的設計	9
3.5 訓練與部署實作流程	11
3.6 功能消融與比較原則	11
3.7 CFR 的實際 nano 結構.....	12
3.8 CFR 通道配置與介面核對.....	13
3.9 一般化 CFR 與啟用條件.....	13
3.10 LDS 的算子順序與通道注意力	14

目錄（續）

3.11 Backbone 的逐層尺寸	15
3.12 多尺度融合與偵測輸出	16
3.13 模型狀態與匯出紀錄	18
第四章 成果展示	19
4.1 實驗環境與訓練設定	19
4.2 延遲量測程序與指標	22
4.3 主要模型結果	24
4.4 精度與速度取捨及功能消融	25
4.5 CFR 位置消融	29
4.6 推論流程比較	30
4.7 前處理與後處理成本	31
4.8 FLOPs 與延遲的落差	32
4.9 記憶體行為與可能機制	33
4.10 未成功的嘗試與研究限制	34
4.11 實際偵測案例與選取方式	34
第五章 結論與未來發展	39
5.1 結論	39
5.2 後續驗證與發展方向	39
參考文獻	40

圖目錄

圖 1 模型訓練與邊緣部署評估流程.....	8
圖 2 各模型的 backbone 模組配置.....	8
圖 3 CFR 與 LDS 的功能示意.....	10
圖 4 本次 ONNX 與 NCNN 輸出分支.....	10
圖 5 CFR 在 nano 配置下的實際特徵路徑.....	12
圖 6 LDS 的 GhostConv 與 ECA 運算順序.....	15
圖 7 三個主要模型的訓練精度歷程.....	21
圖 8 主要模型的框回歸與分類損失.....	22
圖 9 主要模型精度與 NCNN 推論延遲.....	25
圖 10 各模型的逐種子精度分布.....	27
圖 11 固定信心門檻下的 precision 與 recall.....	28
圖 12 各模型在兩套部署流程的推論時間.....	30
圖 13 理論計算量與實測延遲的相對變化.....	32
圖 14 影像 008863 的標註與三模型結果.....	35
圖 15 影像 006217 的標註與三模型結果.....	36
圖 16 影像 009854 的標註與三模型結果.....	37

表目錄

表 1 研究問題與對應實驗	2
表 2 相關研究與本專題的對應	5
表 3 下採樣卷積的理論 MAC 比較	6
表 4 本研究的特徵操作與作用	7
表 5 三種效率資料的用途	7
表 6 模型名稱與實作對照	11
表 7 CFR 的 nano 內部尺寸	13
表 8 第 2 與第 4 層的融合態參數比較	13
表 9 CFR 程式版本的結構差異	14
表 10 Backbone 層級與輸出尺寸	16
表 11 多尺度融合的來源與輸出	17
表 12 預設 80 類 YAML 的模型摘要	18
表 13 軟硬體與資料集設定	19
表 14 共同訓練超參數	19
表 15 納入精度統計的 run 對照	20
表 16 主要模型與位置消融的精度及 NCNN 延遲	24
表 17 主要模型與位置消融的資源指標	24
表 18 主要變體與功能消融之精度比較	26
表 19 各模型的逐種子精度	27
表 20 固定信心門檻下的 precision 與 recall	28
表 21 不同 CFR 位置的比較	29
表 22 ONNX 與 NCNN 推論時間	30
表 23 baseline 三階段延遲摘要	31
表 24 相對 baseline 的成本與延遲變化	32
表 25 未成功的探索結果	34
表 26 六張案例的 TP 與 FP 及 FN	38

第一章 緒論

1.1 背景介紹

物件偵測須同時辨識影像中的物件類別與位置，是邊緣視覺系統的重要運算環節。當模型部署於資源有限的 ARM CPU 時，除了偵測精度，單張影像的處理時間也會影響系統更新頻率。模型參數量反映權重規模，浮點運算量（FLOPs）反映指定輸入下的理論計算需求，但執行時間還取決於算子實作、資料存取及推論框架。因此，參數較少或 FLOPs 較低的模型，未必在特定裝置上具有相同幅度的速度優勢。MnasNet 已將行動裝置的實測延遲納入架構搜尋目標[1]，說明硬體量測可以直接參與模型選擇。本研究沿用此觀點，以 YOLO26n 為基礎，在 Raspberry Pi 5 上比較局部架構修改前後的精度與推論時間，分析輕量化設計在實際部署時的效益。

1.2 實驗動機

本研究的動機來自模型成本與部署速度之間的不一致。一般卷積同時進行空間特徵擷取與通道混合，改用較低成本的運算雖可減少乘加次數，卻也可能引入額外的分支、拼接及特徵重加權。這些改動對 GPU 訓練與 ARM CPU 推論的影響未必相同，僅比較模型摘要難以判斷其應用價值。為此，本研究設計兩個方向不同的變體：YOLO-CFR 修改瓶頸內部運算與特徵融合，觀察精度改善所需的成本；YOLO-LDS 調整下採樣模組，檢查降低計算量後是否仍能維持可接受的精度。兩者皆在共同訓練條件下評估，再分別以 ONNX 與 NCNN 流程部署，以區分架構修改及部署流程選擇所帶來的時間變化。

1.3 研究目標與範圍

本研究以 baseline、YOLO-CFR 與 YOLO-LDS 為主要比較對象，其他配置用於功能、位置或設計變體的消融分析。研究目標是建立固定訓練預算下的精度與延遲比較，檢查 FLOPs 變化能否反映實機時間，並分析同一架構在不同部署流程中的表現。實驗範圍限定於 PASCAL VOC、640×640 輸入及 Raspberry Pi 5 CPU 的 FP32 推論。文中 FPS 由純推論延遲換算，使用情境為單張影像的模型運算；攝影機擷取、傳輸、顯示及排程不在該數值的計時範圍內。量化、其他硬體平台及長時間串流驗證列為後續研究方向。

1.4 研究問題與比較層次

模型與部署分別對應兩個評估層次。模型評估在共同資料與訓練預算下比較偵測精度，部署評估則量測模型在指定裝置及推論流程中的時間。表 1 整理兩者與消融實驗的關係。對應用而言，增加精度但延長推論時間的配置，是否優於原模型，取決於可接受的延遲與誤差；相同地，速度提升也需連同漏檢及誤檢的變化判斷。因此，本研究同時報告各模型的精度、計算成本及實測延遲，並保留不同種子的結果，以呈現平均表現之外的訓練變異。

表 1 研究問題與對應實驗

研究問題	比較對象	主要觀察量
架構修改是否改善精度	baseline、CFR、LDS	三種子 mAP50-95
降低計算量是否等比例加速	各模型對 baseline	FLOPs 與實測 ms
部署流程是否改變效益	ONNX 與 NCNN	同模型流程延遲
效果是否隨位置與元件改變	功能及位置消融	精度、成本與配置

架構修改集中於共同偵測器的 backbone，使各模型仍可使用相同的多尺度融合與偵測輸出。此設計便於定位改動，但自訂模組的替換可能同時改變隱藏通道、殘差與算子組合。主要比較因而以完整模型配置為單位；功能與位置消融則提供進一步的結構分析，其可歸因的範圍依實際控制的條件判定。

1.5 研究成果與文件範圍

本研究完成 CFR 與 LDS 自訂模組、模型設定、多種子訓練及 Pi 5 部署，並將精度、成本與推論時間整理為共同的比較結果。第二章說明輕量化卷積、特徵融合、通道注意力及硬體效率的研究背景；第三章依程式與實際架構說明模組配置、張量尺寸和匯出流程；第四章呈現訓練歷程、消融結果、部署量測與偵測案例；第五章歸納研究結果及後續方向。本文區分模組的一般化設計與 nano 模型實際執行的結構，並將量測結果與可能的效能機制分別討論。

第二章 相關文獻

2.1 延遲導向模型設計

MnasNet 以行動裝置的實測延遲評估候選網路，將硬體表現納入精度與成本的取捨[1]。此方法反映了代理成本與實際速度之間的差異：具有相近 FLOPs 的網路，可能因算子類型和執行方式不同而產生不同延遲。本研究採人工設計的局部架構變體，在固定硬體與部署流程下比較其表現。實測延遲在此作為模型選擇的依據，FLOPs 則用於描述計算需求，兩者共同構成效率評估。

2.2 YOLO26 與部署分支

YOLO26 為 Ultralytics 提供的物件偵測模型系列，本研究採用其 nano 配置 YOLO26n 作為基準[2]。模型的端到端偵測設計與傳統密集預測接續非極大值抑制（non-maximum suppression, NMS）的流程不同；YOLOv10 已透過一致的雙重指派研究 NMS-free 偵測[3]，提供相關方法背景。對部署研究而言，訓練時的網路、實際匯出的計算圖及輸出整理程序需一併確認。本次 ONNX 路徑保留端到端輸出，NCNN 匯出則停用該輸出方式，改由密集預測接續 NMS。兩套流程因此同時具有執行框架與輸出分支差異，後續的時間比較以完整部署流程為單位。

2.3 GhostConv 與 ECA 通道注意力

GhostNet 先以一般卷積產生部分基礎特徵，再透過較低成本的運算生成額外特徵，以降低密集卷積的計算需求[4]。本研究將 GhostConv 用於 CFR 的瓶頸運算，以及 LDS 的下採樣階段。ECA 則利用局部跨通道交互作用產生注意力權重，以一維卷積完成通道關係建模[5]。LDS 在 GhostConv 之後加入 ECA，使低成本特徵生成與通道重加權連接於同一模組。兩種元件的功能不同，前者改變特徵生成方式，後者調整輸出通道的相對回應；其組合效果由本研究的精度與消融結果評估。選擇這組元件的理由，在於特徵生成與特徵選擇可分別處理。GhostConv 將部分輸出建立在已生成的基礎特徵之上，減少所有通道都由密集卷積獨立計算的成本，但衍生特徵也受限於基礎特徵所保留的資訊。ECA 接收完整的拼接結果，依輸入內容調整各通道的權重，使兩種來源的特徵共同參與後續處理。這種組合提供了可訓練的通道選擇方式，卻不能保證補回下採樣時已失去的空間細節。其是否適合物件偵測，仍取決於定位資訊的保留程度，以及重加權所增加的操作是否抵銷卷積節省。因而，本研究同時檢查精度與部署時間，並以功能消融觀察組合元件在既定訓練條件下的表現。

2.4 NCNN 與既有 Pi 5 量測

NCNN 是面向行動與嵌入式平台的推論框架，PNNX 為其模型轉換流程中的工具之一[6]。Ultralytics 的 Pi 5 部署文件提供不同格式的測試結果，可作為比較量級的外部參考[7]。本研究在此平台上進一步量測自訂架構，關注同一模型在不同流程的延遲，以及架構修改後的相對變化。由於外部測試的類別數、軟體版本及匯出設定可能不同，官方數值用於背景比較，主要結論仍依本研究共同條件下的量測建立。

2.5 輕量化卷積與特徵路徑

單階段偵測器將物件定位與類別預測整合於同一網路。YOLO 以單一網路完成偵測[8]，SSD 則利用多尺度特徵圖與預設框處理不同大小的目標[9]。FPN 透過由上而下及側向連接形成多尺度語意特徵[10]，PANet 進一步增加由下而上的資訊傳遞路徑[11]。這些設計說明，偵測精度除了與單一層的特徵能力有關，也受到不同解析度之間的資訊融合影響。本研究固定 YOLO26n 的多尺度偵測流程，將修改集中於 backbone，以便比較指定模組的替換效果；YOLO26 的整體設計可參考其技術預印本[12]。輕量化卷積主要透過重新安排空間運算及通道混合來降低成本。MobileNet 採用 depthwise 與 pointwise 卷積[13]，MobileNetV2 引入 inverted residual 與 linear bottleneck[14]，ShuffleNet 則結合 pointwise group convolution 與 channel shuffle[15]。GhostNet 使用低成本操作生成部分特徵[4]，CSPNet 透過階段內的部分特徵路徑安排網路運算[16]。CFR 結合 Ghost bottleneck 與分支融合，相關元件均有既有研究基礎。本研究評估的是這些運算在指定 YOLO26n 配置中的組合，以及該配置於 Pi 5 上的實際精度與延遲。

2.6 通道注意力與模型規模

SE 以通道關係重新校正特徵回應[17]，ECA 則採用局部跨通道交互作用，透過一維卷積產生權重[5]。本研究選用 ECA 接續 GhostConv，希望在較低成本的特徵生成之後保留通道選擇能力。注意力參數量雖少，執行時仍涉及池化與逐通道乘法，因此其精度與時間效益需分別觀察。另一方面，MobileNetV3 結合硬體導向搜尋與網路調整[18]，EfficientNet 以深度、寬度及解析度的協同縮放改善模型配置[19]。這些研究也指出，模組比較應考慮整體模型規模；若兩個配置的計算預算不同，觀察到的精度差異可能同時受到容量變化影響。

2.7 硬體效率與研究定位

ShuffleNet V2 指出，記憶體存取、分組配置、網路碎片化及逐元素運算都可能影響實際速度，FLOPs 因而不足以單獨衡量效率[20]。對本研究的 GhostConv 與 ECA 組合而言，密集卷積的乘加次數下降，同時伴隨分支處理、拼接及池化等操作，這些成本在不同框架中的實作也可能不同。Roofline 模型以計算峰值、可達記憶體頻寬及算術強度描述效能限制[21]，提供分析計算與資料搬移關係的方法。本研究以此解釋可能的效率差異，但 PNNX 的 Memory OPS 屬圖層級摘要，尚不足以建立 Pi 5 的實測 Roofline。

表 2 相關研究與本專題的對應

研究方向	代表文獻	本專題採用的觀點
偵測基礎	YOLO [8]；YOLO26 [2,12]	固定偵測流程，比較 backbone 變體
輕量卷積	MobileNet [13,14]；GhostNet [4]	減少運算，仍需量測部署時間
特徵與注意力	CSPNet [16]；SE [17]；ECA [5]	檢視特徵路徑及通道重加權
硬體效率	MnasNet [1]；ShuffleNet V2 [20]；Roofline [21]	區分計算量、資料處理與實機延遲

表 2 整理本研究採用的設計與分析依據。研究工作著重於 CFR、LDS 的局部架構配置，以及固定訓練條件和 Pi 5 部署流程下的比較。MCUNet 將模型與推論引擎共同設計，以適應微控制器的資源限制[22]；本研究的平台雖不同，仍需同時考慮網路與執行軟體。執行緒設定也是其中一項條件，ONNX Runtime 提供 intra-op 與 inter-op 等管理選項[23]，可能影響 CPU 的平行運算方式。因此，部署結果應連同框架版本、匯出圖及執行設定解讀，才能判斷架構變化在何種環境下有效。

2.8 卷積成本與 Ghost 特徵生成

標準卷積的每個輸出通道都結合輸入通道的空間資訊。設輸入通道為 C_i 、輸出通道為 C_o 、卷積核大小為 $k \times k$ ，輸出特徵圖為 $H_o \times W_o$ ，其乘加次數如公式(1)。本研究以一次乘法累加定義一個 MAC；若將乘法與加法各計為一次浮點運算，卷積 FLOPs 約為 MAC 的兩倍。GhostConv 的實作先由一般卷積產生 $C_o/2$ 個通道，再對這些特徵執行 5×5 depthwise 卷積，最後沿通道維度拼接。對本研究採用的偶數通道配置，其卷積成本可寫為公式(2)，其中第二項描述低成本特徵生成分支，估算未包含 BatchNorm、活化及拼接。

$$MAC_{standard} = H_o W_o C_i C_o k^2 \quad (1)$$

$$MAC_{ghost} = H_o W_o (C_i C_o k^2/2 + 25 C_o/2) \quad (2)$$

表 3 下採樣卷積的理論 MAC 比較

位置	$C_i \rightarrow C_o$	標準卷積	GhostConv
第 3 層	64 $\rightarrow$ 64	235.93 M	123.08 M
第 5 層	128 $\rightarrow$ 128	235.93 M	120.52 M
第 7 層	128 $\rightarrow$ 256	117.96 M	60.26 M

640 輸入；輸出空間依序 80²、40²、20²。僅估算卷積 MAC，非 PNNX 全網路 FLOPs，也未含 ECA。

表 3 列出 LDS 三個替換位置的卷積成本。在相同輸入、輸出形狀下，GhostConv 的 MAC 約為標準卷積的 51%~52%，主要來自密集卷積只生成一半通道。這個比例描述單層運算，整個模型的節省幅度則取決於這些層占原模型成本的比重，以及新增 ECA 的運算。後續偵測層與其他 backbone 層仍需執行，因此全網路 FLOPs 不會下降至相同比例。理論估算可說明替換方向，實際加速仍由 Pi 5 量測判定。由表中數值相加，三層標準卷積約需 589.82 M MAC，替換後的 GhostConv 約為 303.86 M MAC，局部減少約 48.5%。第 3、5 層的標準卷積成本相同，是因為空間位置數減為四分之一的同時，輸入與輸出通道各加倍，使通道乘積增加為四倍。第 7 層的空間位置數再減為四分之一，但輸出通道僅增加一倍，因此其成本約為前一層的一半。這個例子顯示，應同時追蹤解析度與通道數，才能判斷哪個位置值得替換；僅以層號深淺或輸出通道多寡排序，可能忽略實際的計算分布。

2.9 通道重加權與特徵保留

通道注意力依輸入特徵產生權重，調整不同通道的回應幅度。SE 與 ECA 均利用通道統計進行重加權[17,5]；本研究的 ECA 先對各通道執行全域平均池化，再以一維卷積及 sigmoid 產生權重，如公式(3)與公式(4)。其中 X 為輸入特徵， z 為通道統計， a 為注意力權重， Y 為重加權結果。權重沿空間維度廣播後與原特徵相乘，因此輸出仍保有相同的通道數、高度及寬度。由於每個通道最後只得到一個權重，同一通道內所有空間位置會乘上相同係數。ECA 可以調整特徵通道之間的相對強度，卻沒有在此步驟中為每個像素建立獨立權重。因此，本文將它定位為通道選擇元件，而不將其描述為直接修正物件邊界或遮擋區域的空間注意力。定位資訊仍須由卷積特徵及後續多尺度偵測共同提供。

$$z_c = (1/HW) \sum_{h=1}^H \sum_{w=1}^W X_{c,h,w} \quad (3)$$

$$a = \text{sigmoid}(\text{Conv1d}(z)); Y_{c,h,w} = a_c X_{c,h,w} \quad (4)$$

CFR 的特徵保留則透過分支與拼接完成。輸入投影後的一部分特徵直接保留，另一部分經瓶頸處理，再於輸出端融合。這類路徑安排與 CSPNet 的部分特徵處理設計具有相關背景[16]，但與 ECA 改變特徵幅度的作用不同。表 4 比較切分、拼接、相加及重加權的形狀關係。拼接增加後續層的輸入通道，相加要求兩個張量形狀相容，注意力則需額外完成統計及乘法。分析模組成本時，須分別考慮這些運算，而不能僅以參數數量概括其執行負擔。

表 4 本研究的特徵操作與作用

操作	形狀關係	本專題用途
通道切分	$C \rightarrow C/2$ 與 $C/2$	建立保留與處理分支
通道拼接	$C_1, C_2 \rightarrow C_1+C_2$	匯合不同來源特徵
逐元素相加	形狀需相容	殘差保留輸入
通道重加權	輸出形狀不變	ECA 調整各通道回應

2.10 模型成本與部署時間的關係

模型成本可分為權重、運算與執行期特徵三個部分。參數量描述可學習權重的數目，FLOPs 描述指定輸入下的算術運算；中間特徵則隨解析度、通道數及同時保留的分支改變。即使模型權重變少，推論時仍可能需要保留多個特徵張量。GhostConv 將密集卷積拆成基礎與衍生特徵路徑，LDS 再加入 ECA，因而同時改變乘加次數及資料處理方式。ShuffleNet V2 對實際速度的分析[20]與 Roofline 的效能觀點[21]均支持將這些因素分別考慮。

表 5 三種效率資料的用途

資料	可回答的問題	本研究的解讀範圍
參數與 FLOPs	模型規模、理論運算改變	相同統計狀態內比較
實測推論 ms	指定流程在 Pi 5 的速度	受框架、版本與輸出分支影響
記憶體流量與逐層時間	延遲集中於何處及原因	本次尚未取得完整原始量測

表 5 區分靜態摘要、整體時間及細部量測的用途。本研究先在共同訓練條件下確認精度，再比較目標平台的推論時間，最後利用匯出圖及成本摘要討論可能原因。這個流程使效能結論直接連結至量測結果，也保留機制分析所需的證據層次。現有資料能確認特定配置的 FLOPs 與時間變化不成比例；若要判定差異是否由記憶體頻寬或特定算子主導，仍需逐層耗時及實測資料流量。

第三章 作品介紹

3.1 系統流程與模型配置

本研究的流程由模型設定開始，依序完成共同條件訓練、精度評估、模型匯出及 Pi 5 部署。圖 1 呈現資料與模型在各階段的關係，圖 2 則比較 backbone 的修改位置。baseline 採原始 YOLO26n，CFR 與 LDS 為兩個主要設計，其餘配置歸入消融實驗。三個主要模型保留相同的多尺度融合與偵測流程，使精度及時間差異可連回指定模組的變更。

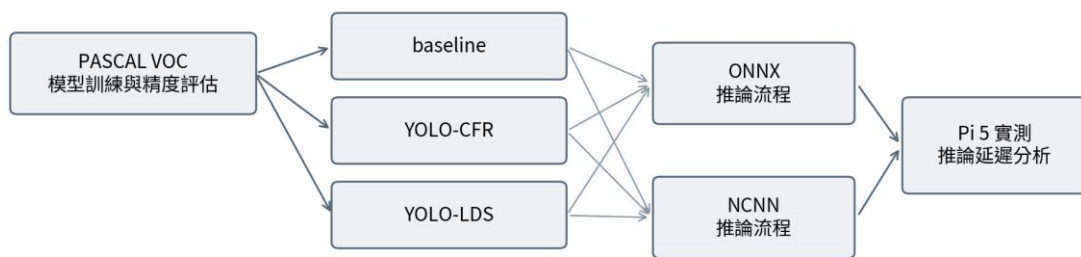
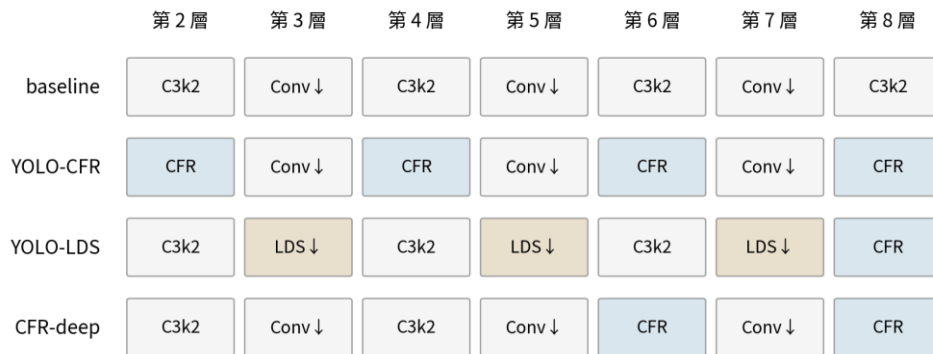


圖 1 模型訓練與邊緣部署評估流程



↓ 表示 stride 2 下採樣；僅列出本研究比較的 backbone 層。

圖 2 各模型的 backbone 模組配置

YOLO-CFR 在第 2、4、6、8 層置入 CFR；YOLO-LDS 在第 3、5、7 層以 LDS 執行下採樣，並於第 8 層保留 CFR。CFR-deep 僅修改第 6、8 層，用於位置消融。層號依本研究 YAML 的排列定義，後續張量表也採相同編號。這種配置讓 CFR 主要影響同尺度的特徵處理，而 LDS 主要影響解析度縮小時的運算。

3.2 CFR 特徵複用模組

CFR 為 Cross-stage Feature Reuse 的縮寫，以 GhostBottleneckReuse 作為內部處理單元，並保留切分分支與單元輸出的拼接。模組先投影輸入通道，再將保留分支和處理分支於輸出端融合。一般化程式亦支援將前一單元的部分輸出傳入後一單元；本次 nano 架構中，每個 CFR 僅有一個內部單元，實際啟用的是分支融合與 Ghost bottleneck。其結構與通道設定於第 3.7 至 3.9 節詳述。CFR 的比較包含瓶頸算子、隱藏通道及殘差設定的共同變化。原始 C3k2 與自訂模組的建構子介面不同，沿用相同 YAML 數列時需確認參數實際對應。這些差異會影響內部卷積成本，因此參數總量接近 baseline 並不代表兩者具有相同運算預算。本研究將 CFR 視為完整配置，並結合精度、FLOPs 與延遲評估其取舍。

3.3 LDS 輕量下採樣模組

LDS 為 Lightweight Downsampling 的縮寫，以 GhostConv 接續 ECA，取代指定位置的 stride 2 普通卷積。GhostConv 先降低空間解析度並產生特徵，ECA 再利用下採樣後的通道統計調整回應。此設計的目標是在減少密集卷積運算的同時維持偵測所需的特徵表達。YOLO-LDS 於第 8 層保留 CFR，因此本文的 LDS 結果代表整個配置；與其他消融模型比較時，也需同時考慮該層及下採樣模組的差異。下採樣會將較大的特徵圖轉為較小的表示，其輸出將持續供後續 backbone 與多尺度融合使用。因此，這一位置的修改不只影響該層計算，也會改變後續網路收到的資訊。LDS 保留原有輸出通道與空間尺寸，讓比較集中於特徵的產生方式。ECA 安排在下採樣之後，可使池化與逐通道乘法作用於較小的空間範圍；相同通道數下，特徵圖的長寬各減半，這兩類操作所需處理的位置數亦降為四分之一。不過，此處只是在說明算子順序的工作量差異，並未實驗比較注意力放在下採樣前後的優劣。後續結果所驗證的是目前 GhostConv 接續 ECA 的配置，不包含未實作的其他排列。

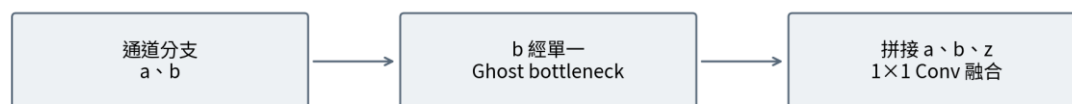
3.4 位置消融的設計

位置消融比較全層 CFR 與僅深層 CFR 的配置。對 640×640 輸入，第 2、4 層特徵圖分別為 160×160 、 80×80 ，第 6、8 層則為 40×40 、 20×20 。最大與最小特徵圖的空間位置數相差 64 倍，因此相似運算放置於不同階段時，可能具有不同的成本。CFR-deep 僅替換第 6、8 層，用以觀察保留深層修改後的精度與延遲。此配置完成一個種子的訓練，屬初步位置分析；通道數與模組內部設定仍會影響成本，不能僅依特徵圖面積估算整體加速。

3.4.1 模組功能與部署輸出示意

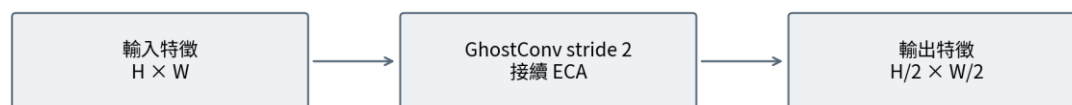
圖 3 整理 CFR 與 LDS 的主要特徵路徑。CFR 的 nano 實例將切分分支與單一 Ghost bottleneck 的輸出融合；LDS 則依序執行 GhostConv 與 ECA。兩者都保留外部張量形狀的相容性，使修改後的 backbone 可銜接原有偵測流程。

CFR 的 nano 實例



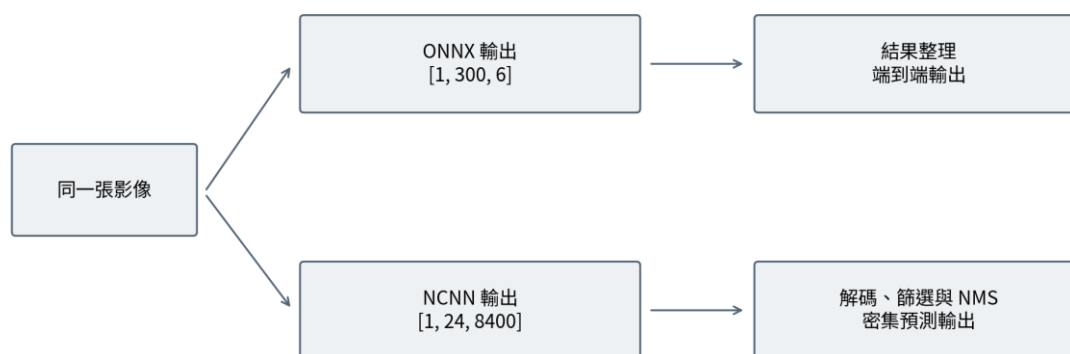
n=1；一般程式支援的跨單元複用路徑未在此實例啟用。

LDS 的實際算子順序



依 block_custom.py、conv_reference.py 與逐層傾印繪製。

圖 3 CFR 與 LDS 的功能示意



本次匯出配置的分支差異；未表示兩條路徑的部署精度已一致。

圖 4 本次 ONNX 與 NCNN 輸出分支

圖 4 呈現部署輸出的差異。VOC 設定包含 20 類，密集預測的 24 個欄位由 4 個框座標及 20 個類別欄位組成；端到端輸出則以每筆 6 個欄位表達篩選後結果。這項差異改變輸出整理與 NMS 的工作量，因此後續量測分別報告前處理、推論及後處理時間。

3.5 訓練與部署實作流程

各模型共用訓練資料與超參數，主要模型及功能消融採 seed 0、1、2，CFR-deep 則使用單一種子。原實驗紀錄將初始化描述為從頭訓練；args.yaml 中的 pretrained 欄位與初始化判讀於第 4.1.1 節說明。固定 100 epochs 的目的，是比較相同訓練預算下各配置的結果。訓練完成後，權重分別匯出為 ONNX 與 NCNN 格式，再於 Pi 5 進行推論量測。自訂模組需能轉換為部署框架支援的運算，匯出後的輸出形狀與分支亦須確認。表 6 列出報告名稱、實驗目錄及對應模組，便於追溯各項結果。

表 6 模型名稱與實作對照

報告名稱	實驗資料夾	內部模組
baseline	baseline-5 baseline_seed1 / seed2	原版 YOLO26n
YOLO-CFR	ghostreuse ghostreuse_seed1 / seed2	C3k2_GhostReuse GhostBottleneckReuse
YOLO-LDS	ghost_reuse8 ghost_reuse8_seed1 / seed2	GhostECA C3k2_GhostReuse
CFR-deep	cfrdeep-2	C3k2_GhostReuse 僅第 6、8 層

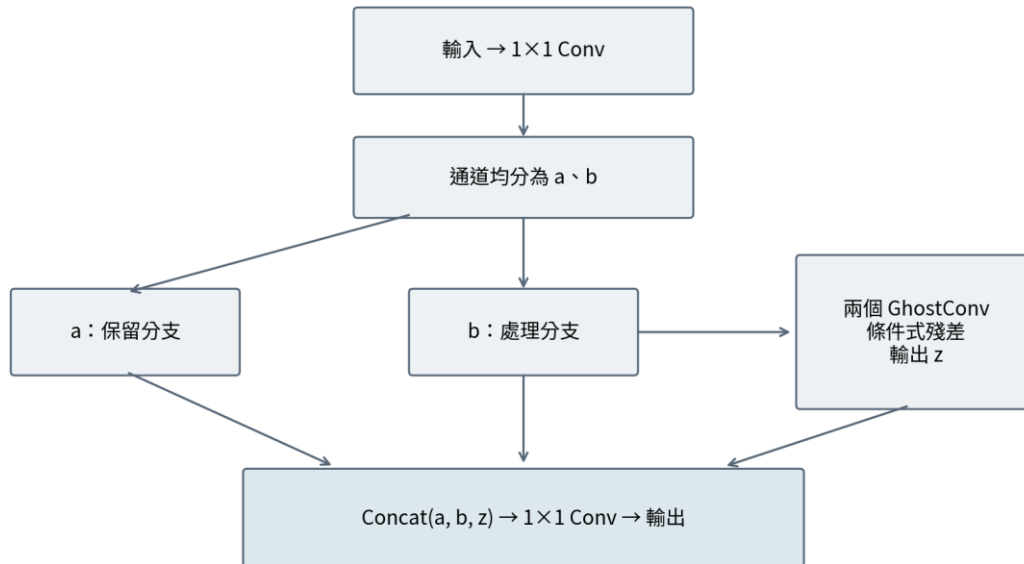
實驗目錄 ghostreuse 對應 CFR，ghost_reuse8 對應 LDS；seed1、seed2 為各模型的重複訓練。CFR-deep 及其他變體均屬消融實驗。

3.6 功能消融與比較原則

功能消融包含 ghost、eca 與 eca_ghost，依實驗設計分別檢查 GhostConv、ECA 及其組合；CFR-deep 為位置消融，V3、V4 等版本則屬結構變體。主要模型仍為 CFR 與 LDS，baseline 為共同基準。功能消融採三個種子，比較固定訓練預算下的精度變化。由於本次程式歸檔未涵蓋每個消融配置的完整 YAML，本文保留其實驗結果，對單一元件的作用則依可確認的架構條件解釋。部署比較固定輸入尺寸、硬體及量測程序，並將訓練態與匯出態的成本分開呈現。主要 FLOPs 與 Memory OPS 比較採用 PNNX 匯出摘要，精度採三種子結果，延遲採重複執行の中位數。這三種資料分別描述計算圖、訓練變異與執行時間，不能以其中一項的誤差代替另一項。

3.7 CFR 的實際 nano 結構

YOLO-CFR 的架構紀錄顯示，第 2、4、6、8 層各自只有一個內部處理單元。YAML 的重複數雖為 2，經 nano 深度倍率 0.50 縮放後，實際數量為 $n=1$ 。圖 5 依此配置繪出特徵路徑，區分輸入投影、分支處理與輸出融合。



nano 傾印：每個 CFR 的 $n=1$ ，沒有第二個單元接收選取特徵。

圖 5 CFR 在 nano 配置下的實際特徵路徑

輸入先經 1×1 卷積投影為 $2c$ 個通道，再切為 a 、 b 兩個分支。分支 b 經包含兩個 GhostConv 的瓶頸得到 z ，輸出端將 a 、 b 、 z 拼接為 $3c$ 個通道，再以 1×1 卷積產生指定輸出。殘差是否啟用由 shortcut 決定。程式另取 z 的前 $c/2$ 個通道作為 main，但 $n=1$ 時沒有後續單元接收，因此該跨單元路徑未參與本次 nano 運算。本文保留 CFR 的實驗名稱，並將觀察到的精度變化解釋為這個完整配置的效果。此融合方式保留了不同處理深度的特徵： a 與 b 來自同一次輸入投影， z 則另經瓶頸轉換。將三者沿通道拼接後，輸出卷積可以分別對各組特徵學習權重；若先將它們逐元素相加，對應通道便會在融合前混合，結構及成本都不同。保留 b 的直接路徑，也表示瓶頸分支不必獨自承擔所有輸入資訊的傳遞。然而， a 與 b 已經過投影卷積，並非原始輸入的無變化複本。這些結構關係可解釋 CFR 提供了哪些資訊路徑，但僅由路徑存在，仍不能判定訓練後模型實際依賴哪一支；該問題需要固定其餘配置的路徑消融才能回答。

3.8 CFR 通道配置與介面核對

CFR 的隱藏通道定義為 $c=\text{int}(C_0e)$ ，輸入投影輸出 $2c$ 個通道，融合卷積則接收 $(2+n)c$ 個通道。表 7 列出本次 nano 配置的實際尺寸，四個 CFR 的隱藏通道依序為 32、64、64、128。尺寸由模型實例確認，除 YAML 中的數值外，也受到寬度縮放及建構子參數對應影響。

表 7 CFR 的 nano 內部尺寸

層	輸入 C	輸出 C	隱藏 c	融合輸入 C	n
2	32	64	32	96	1
4	64	128	64	192	1
6	128	128	64	192	1
8	256	256	128	384	1

原始 C3k2 在 n 後接收 c3k、e、attn 等參數，自訂 C3k2_GhostReuse 則接收 shortcut、g、e，兩者的介面不同。以 YAML 中的 [256, False, 0.25] 為例，0.25 在自訂介面中不再位於 e 的位置。實際架構的第 2、4 層 cv1 輸出分別為 64、128，對應自訂模組 $e=0.5$ ；原版相應配置的隱藏通道則較窄。表 8 顯示，第 2、4 層的融合態參數分別增加 11,880 與 45,776，說明瓶頸替換同時伴隨內部分支寬度變化。這可解釋 CFR 雖使用 GhostConv，整體成本仍可能高於 baseline 的原因。若進一步評估複用路徑本身的效果，則需固定隱藏通道及殘差後再進行比較。隱藏通道的變化也會放大高解析度層的成本。輸入投影的運算隨 $2c$ 增加，輸出融合則隨 $(2+n)c$ 增加，而瓶頸內涉及通道混合的部分還可能同時受到輸入與輸出寬度影響。因此，即使某個內部卷積改為 GhostConv，其節省仍可能被較寬的分支抵銷。檢查 YAML 時，除了外部輸出通道，還須核對實例中的 c、n 與 shortcut；這些數值才能將程式介面、實際結構與成本結果連結起來。

表 8 第 2 與第 4 層的融合態參數比較

層	baseline	CFR	差值
2	6,520	18,400	+11,880
4	25,840	71,616	+45,776

3.9 一般化 CFR 與啟用條件

一般化 CFR 容許 n 大於 1。第一個單元接收分支 b ；自第二個單元起，將前一單元的完整輸出 z 與選取的前半通道 r 拼接，再送入下一個 Ghost bottleneck。

令 $z_0=b$ ，各單元的輸入、通道選取及最終融合如公式(5)至公式(7)。此設計將較早的輸出引入後續運算，而輸出端仍保留 a 、 b 及各單元結果。

$$u_1 = b; \quad u_i = \text{Concat}(z_{i-1}, r_{i-1}), i \geq 2 \quad (5)$$

$$r_i = z_i[:, :c/2, :, :] \quad (6)$$

$$Y = \text{Conv}1 \times 1(\text{Concat}(a, b, z_1, \dots, z_n)) \quad (7)$$

通道選取位於第二個 GhostConv 之後，若啟用 shortcut，則在殘差相加後取得 main。因此，main 表示瓶頸輸出的前半通道，而非第一個 GhostConv 的中間特徵。當 $n=1$ 時，僅執行 $u_1=b$ ； $n \geq 2$ 才會啟用跨單元拼接，並增加後續單元的輸入通道及運算。表 9 列出 CFR、V3、V4 的結構差異，其中 V3 採單一 GhostConv 變體，V4 調整輸出融合範圍。V3、V4 屬消融設計版本，目前未列入具有完整三種子精度與部署數據的主要比較。

表 9 CFR 程式版本的結構差異

版本	處理單元	輸出融合	本報告用途
CFR	兩個 GhostConv	a 、 b 、所有 z	主要實驗配置
V3	單 GhostConv 變體	a 、 b 、所有 z	僅有程式與 YAML
V4	兩個 GhostConv	a 、最後 z	僅有程式與 YAML

3.10 LDS 的算子順序與通道注意力

LDS 對應 GhostECA 類別，其前向運算為 $\text{eca}(\text{ghost}(x))$ ，即先以 stride 2 的 GhostConv 下採樣，再執行 ECA。圖 6 呈現兩個元件的順序。GhostConv 的一般卷積先生成部分通道， 5×5 depthwise 卷積再生成其餘特徵；拼接後的完整輸出送入 ECA。因此，通道統計與重加權作用於已縮小的特徵圖，外部輸出尺寸仍與原下採樣層相容。

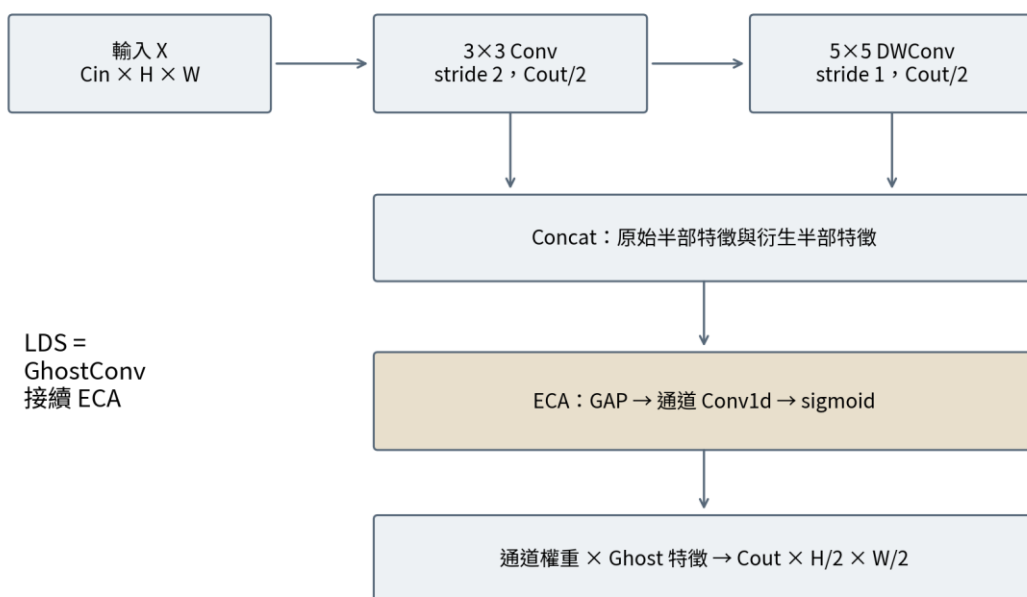


圖 6 LDS 的 GhostConv 與 ECA 運算順序

ECA 的卷積核大小依輸出通道決定。本實作設 $\gamma=2$ 、 $b=1$ ，計算 $t=\text{int}(\text{abs}(\log_2(C)/2+0.5))$ 後，若 t 為偶數則加 1，以得到奇數卷積核。三個 LDS 的輸出通道為 64、128、256，對應核大小 3、5、5。GhostConv 的 5×5 depthwise 卷積沿空間維度生成特徵，ECA 的一維卷積則沿通道序列估計權重。兩種運算作用於不同維度，組合後保留同樣的 $C \times H \times W$ ，故能直接接回第 3、5、7 層之後的共同網路。核大小 3 與 5 表示一維卷積在通道序列上使用的局部範圍，並非影像中的 3×3 或 5×5 鄰域。通道序號本身也不等同於預先指定的物件類別，因此不能把較大的通道核直接解釋為辨識更多類別。就張量操作而言，ECA 先把 $H \times W$ 的空間資訊彙整為每通道一個統計量，再生成 C 個權重，最後回到原特徵圖執行廣播乘法。權重生成所需的參數雖少，池化與乘法仍會處理整張特徵圖，故注意力的部署成本不能只由一維卷積的參數量判斷。

3.11 Backbone 的逐層尺寸

表 10 整理 batch=1、640×640 輸入下的 backbone 張量形狀，省略 batch 維度，以 $C \times H \times W$ 表示通道數、高度及寬度。主要模型與 CFR-deep 在這些層的外部尺寸相同，差異位於指定模組內部。逐層尺寸可用於確認修改是否與後續網路相容，也可協助理解不同位置的運算成本。

表 10 Backbone 層級與輸出尺寸

層	輸出 C×H×W	baseline	CFR	LDS
0	16×320×320	Conv	Conv	Conv
1	32×160×160	Conv	Conv	Conv
2	64×160×160	C3k2	CFR	C3k2
3	64×80×80	Conv	Conv	LDS
4	128×80×80	C3k2	CFR	C3k2
5	128×40×40	Conv	Conv	LDS
6	128×40×40	C3k2	CFR	C3k2
7	256×20×20	Conv	Conv	LDS
8	256×20×20	C3k2	CFR	CFR
9	256×20×20	SPPF	SPPF	SPPF
10	256×20×20	C2PSA	C2PSA	C2PSA

第 2 層具有 160×160 個空間位置，第 8 層則為 20×20，但通道數由 64 增至 256。空間尺寸與通道規模共同決定成本，尤其普通卷積涉及輸入、輸出通道的乘積，與 depthwise 卷積不同。LDS 的替換點位於 stride 2 下採樣，CFR 的替換點則維持同尺度運算，兩者因而改變不同階段的特徵處理。第 9、10 層的 SPPF 與 C2PSA 在主要配置中保留，處理完成後再送往共同的多尺度融合部分。以 FP32、batch=1 且只計單一輸出張量為例，第 2 層的 64×160×160 共含 1,638,400 個元素，約占 6.25 MiB；第 8 層的 256×20×20 則含 102,400 個元素，約占 0.39 MiB。兩者相差 16 倍，而不是僅比較空間位置所得的 64 倍。這項計算只描述張量本身的資料大小，尚未包括同時保留的分支、卷積工作區及框架配置的緩衝區。若某層需要拼接多個特徵，其執行期需求還會受到張量存續時間影響，不能直接以最大單一輸出當作整個模型的峰值記憶體。

3.12 多尺度融合與偵測輸出

第 11 層之後以兩次上採樣與特徵拼接融合不同解析度的資訊，再透過 stride 2 卷積建立由高解析度往低解析度的路徑。表 11 列出各層來源與輸出尺寸。這部分沿用共同架構，其資訊傳遞方式可連結至 FPN 與 PANet 的多尺度融合設計 [10,11]。固定此部分有助於將主要模型的比較集中於 backbone 修改。

表 11 多尺度融合的來源與輸出

層	運算	來源層	輸出 C×H×W
11	上採樣	10	256×40×40
12	Concat	11、6	384×40×40
13	C3k2	12	128×40×40
14	上採樣	13	128×80×80
15	Concat	14、4	256×80×80
16	C3k2	15	64×80×80
17	Conv stride 2	16	64×40×40
18	Concat	17、13	192×40×40
19	C3k2	18	128×40×40
20	Conv stride 2	19	128×20×20
21	Concat	20、10	384×20×20
22	C3k2	21	256×20×20
23	Detect	16、19、22	依輸出分支而異

三個偵測尺度分別為 80×80 、 40×40 、 20×20 ，合計 8,400 個空間位置。VOC 的密集預測含 4 個框欄位及 20 個類別欄位，形成 [1,24,8400] 的輸出；端到端分支則輸出至多 300 筆結果，每筆 6 個欄位，仍需依信心值整理。端到端輸出的最後維度不隨類別數直接增加，因此 [1,300,6] 本身不足以識別模型為 20 類或 80 類。類別設定需由資料設定與模型物件確認，這也是比較參數量及匯出圖時必須保留的條件。這三個尺度中， 80×80 的分支提供 6,400 個位置，占全部位置約 76.2%，顯示高解析度分支在密集輸出中具有較大的資料規模。以 batch=1 的輸出元素數計算，密集分支含 201,600 個數值，端到端輸出最多含 1,800 個，前者為後者的 112 倍。這個比例是張量元素數的比較，不能直接換算為整體推論的速度比，因為兩者在網路內部所執行的偵測分支及篩選程序不同。它的用途在於說明，匯出後除了核對輸入形狀，還需要核對輸出欄位和整理方式，才能對應後處理的實際工作。

3.13 模型狀態與匯出紀錄

模型摘要受到類別數、BatchNorm 融合及輸出分支影響。表 12 列出預設 nc=80 的 YOLO 模型在融合前後的參數與計算量，用於比較相同架構狀態。第四章的主要資源結果則沿用 VOC 實驗紀錄，兩者分開標示，以避免將不同類別數或融合狀態的數值直接混合。

表 12 預設 80 類 YOLO 的模型摘要

模型	融合前參數	融合後參數	GFLOPs 前／後
baseline	2,572,280	2,408,932	6.1 / 5.4
YOLO-CFR	2,593,224	2,430,044	7.3 / 6.5
YOLO-LDS	2,308,197	2,145,105	5.5 / 4.8
CFR-deep	2,535,352	2,372,388	6.1 / 5.3

推論融合可將 BatchNorm 的固定統計及線性轉換併入卷積，框架也可能整理偵測分支，故融合前後的數值需連同計算圖解讀。NCNN 匯出程式由權重載入 YOLO，呼叫 `export(format="ncnn")`，並擷取 PNNX 的 FLOPs 與 Memory OPS 摘要；此步驟不重新訓練模型。本研究部署採 FP32，前後處理範圍與計時程序於第 4.2 節說明。部署結果沿用中位數摘要，逐次 Pi 紀錄與跨格式精度仍待補齊。

第四章 成果展示

4.1 實驗環境與訓練設定

表 13 軟硬體與資料集設定

項目	設定
訓練硬體	NVIDIA RTX 3080 Ti 12 GB
訓練環境	WSL2 Ubuntu；PyTorch 2.5.1；CUDA 12.1
部署硬體	Raspberry Pi 5；四核心 Cortex-A76；16 GB RAM
模型框架	Ultralytics 8.4.93
部署轉換工具	NCNN 1.0.20260526；PNNX 20260526
資料集	PASCAL VOC；20 類
資料分割	trainval 16,551 張；test2007 4,952 張
部署精度與輸入	FP32；640 像素輸入

本研究使用 PASCAL VOC 的 20 類物件偵測設定[24,25]，訓練集共 16,551 張影像，test2007 共 4,952 張。表 13 列出訓練與部署環境，表 14 列出共同超參數。精度統一採 mAP50-95，亦即在 IoU 0.50 至 0.95、間隔 0.05 的十個門檻下計算平均精度，再對類別取平均。這項指標同時反映分類與框定位品質，與僅使用 IoU 0.5 的 mAP50 分開解讀。

表 14 共同訓練超參數

項目	數值	項目	數值
epochs	100	imgsz	640
batch	16	optimizer	AdamW
lr0	0.001	warmup_epochs	5
close_mosaic	15	初始化	見 4.1.1
主要模型種子	0 / 1 / 2	CFR-deep 種子數	1

各模型以 AdamW 訓練 100 epochs。AdamW 將權重衰減與梯度更新分離[26]，本研究對所有配置採相同優化器與學習率設定，以降低訓練配方差異。主要比較衡量固定訓練預算下的結果，並未將各模型調整至其個別最佳超參數。訓練於 RTX 3080 Ti 進行，Pi 5 僅負責部署推論；兩者的硬體與數值設定分別紀錄。共同訓練配方使架構比較具有一致的資料曝光次數與更新預算，但相同 epoch 數不代表相同 GPU 時間。較大的計算圖可能需要更久才能完成一次更新，因此本研

究的精度表回答的是「相同訓練輪數下得到何種結果」，並非「相同訓練時間下哪個模型最好」。這項區分也適用於部署端：訓練使用的批次與 GPU 運算不直接代表 Pi 上單張影像的 CPU 執行效率，模型是否適合邊緣推論仍須由獨立量測確認。

4.1.1 訓練紀錄的選取與核對

訓練資料包含 29 份 results.csv，本文依模型與種子選取 19 份完整紀錄進行主要統計，如表 15 所示。baseline、CFR、LDS 及三組功能消融各有三次訓練，CFR-deep 為一次訓練。中途停止、不同保存名稱及未納入主要比較的紀錄不計入獨立樣本，避免以檔案數代替實驗重複數。

表 15 納入精度統計的 run 對照

模型	seed 0	seed 1、2 前綴
baseline	baseline-5	baseline_seed
YOLO-CFR	ghostreuse	ghostreuse_seed
YOLO-LDS	ghost_reuse8	ghost_reuse8_seed
ghost	ghost-5	ghost_seed
eca	eca-5	eca_seed
eca_ghost	eca_ghost-5	eca_ghost_seed
CFR-deep	cfrdeep-2	僅 seed 0

納入的 19 份 CSV 均有 100 個 epoch，mAP50-95 最大值皆出現在最後一個 epoch。原實驗另記載 best.pt 與 last.pt 的評估摘要相同，故本組結果未呈現最佳 epoch 高於最後 epoch 的情況。主要設定由 YAML 建立模型，args.yaml 中同時保留 pretrained=true；該旗標不足以單獨確認外部權重是否載入，初始化仍以原訓練描述及載入紀錄為準。其餘共同設定包含 batch=16、imgsz=640、lr0=0.001、lrf=0.01、weight_decay=0.0005、warmup_epochs=5、workers=0、cache=false、amp=true 與 deterministic=true。AMP 為 GPU 訓練設定，Pi 部署仍使用 FP32。WSL 可見系統記憶體約 8.7 GiB，與 GPU 的 12,288 MiB 顯示記憶體分別記錄。

4.1.2 精度隨訓練進度的變化

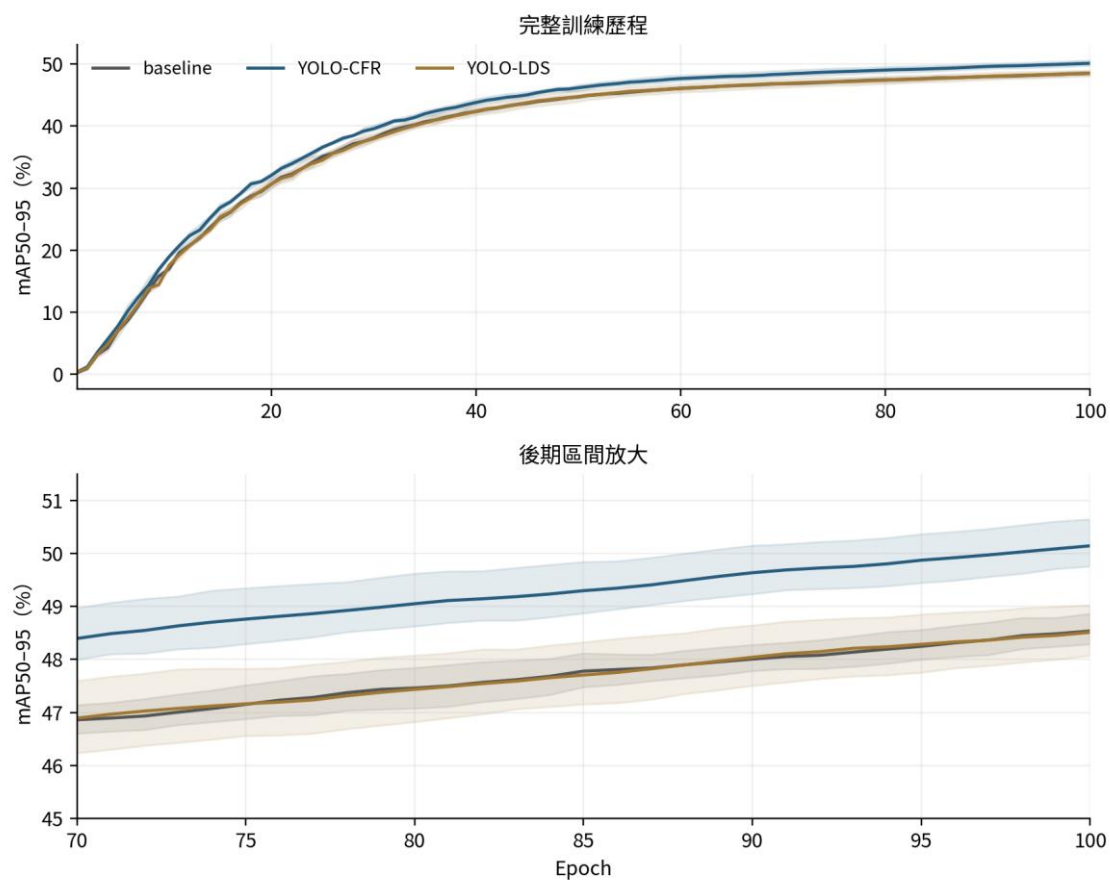


圖 7 三個主要模型的訓練精度歷程

實線為三種子平均；陰影為該 epoch 三次結果的最小至最大範圍，並非信賴區間。資料來自各 run 的 results.csv。

圖 7 呈現三個主要模型的 mAP50-95 訓練歷程，實線為三種子平均，陰影為各 epoch 的最小至最大範圍。第 50 epoch 時，baseline、CFR、LDS 分別約為 0.4472、0.4620、0.4477；第 100 epoch 則為 0.4854、0.5014、0.4851。CFR 的平均優勢在中後期已出現，並非僅由最後一筆紀錄形成。LDS 與 baseline 的曲線範圍多有重疊，與最終平均相近的結果一致。三組最高紀錄均位於第 100 epoch，因此本研究比較的是固定預算內的學習結果，尚不能據此判斷延長訓練後的性能上限及排序。

4.1.3 損失曲線與共同訓練條件

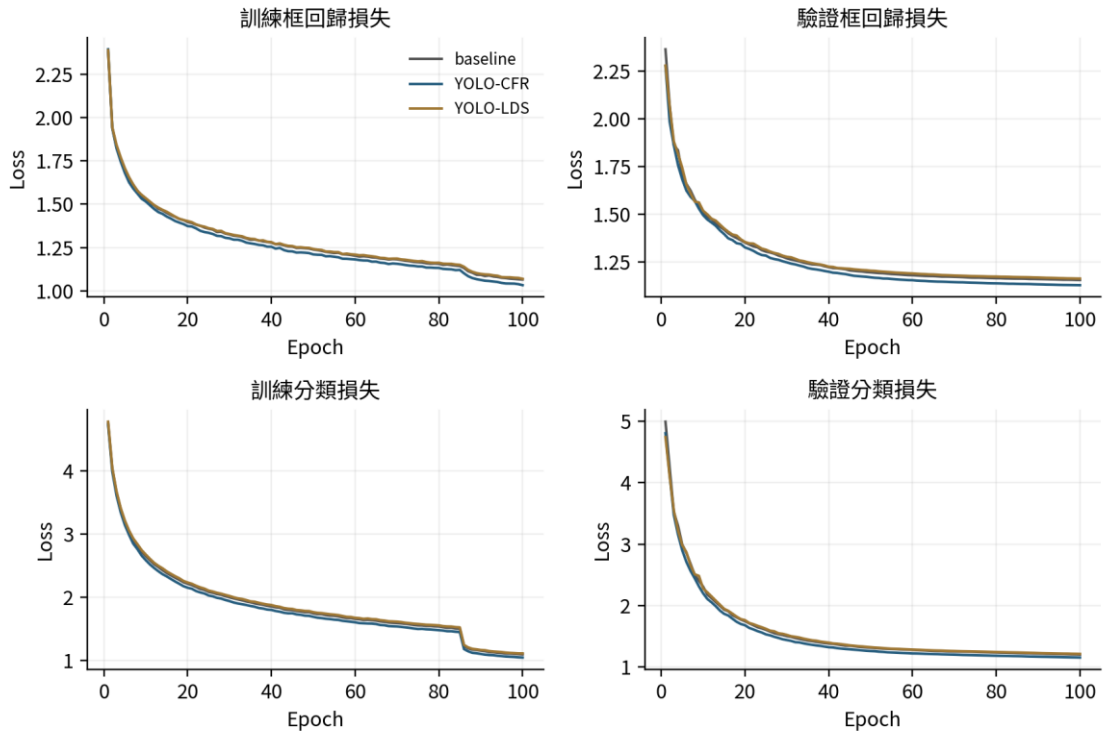


圖 8 主要模型的框回歸與分類損失

各曲線為相同模型三個種子的逐 epoch 平均。訓練與驗證損失各自解讀；並未對不同損失分量重新加權。

圖 8 整理框回歸與分類損失，曲線均為同一模型三個種子的平均。各組損失隨訓練下降，與 mAP 上升的方向一致，CFR 的後期損失也較低。損失曲線提供優化歷程的補充資訊，最終比較仍以偵測指標為準，因為訓練目標與 mAP 的計算方式不同。共同資料增強包含 $hsv_h=0.015$ 、 $hsv_s=0.7$ 、 $hsv_v=0.4$ 、 $translate=0.1$ 、 $scale=0.5$ 、 $fliplr=0.5$ 及 $mosaic=1.0$ ， $mixup$ 、 $cutmix$ 、 $copy_paste$ 均為 0； $close_mosaic=15$ 表示最後 15 個 epoch 關閉 Mosaic。這些條件固定於各模型間，本文沒有另行比較增強策略的效果。訓練曲線描述模型學習，部署延遲則由 Pi CPU 量測，兩者採不同的實驗指標。

4.2 延遲量測程序與指標

Pi 5 以 SSH 純終端環境執行量測，關閉圖形介面及遠端桌面串流，CPU governor 設為 performance。每個模型先暖機 15 次，再連續取樣 60 次；既有紀錄中的 `vcgencmd get_throttled` 為 0x0，溫度約 59–62°C。時間分為前處理、推論及後處理，本文採中位數描述典型延遲。各模型於同一量測 session 依序執行，p90 與跨 session 變異未納入現有摘要，因此數毫秒等級的差距仍需透過重複量測確認其穩定性。

4.2.1 延遲與吞吐的計算

$$T_{total} = T_{pre} + T_{infer} + T_{post} \quad (8)$$

$$FPS_{infer} = 1000 / T_{infer} \quad (9)$$

公式(8)表示同一張影像的三階段總時間，公式(9)則由純推論時間換算理論吞吐，時間單位均為 ms。若先對各階段分別取中位數，再將三個摘要相加，所得數值未必等於逐張總時間的中位數。本文因此將該數值稱為階段摘要合計，並與純推論 FPS 分列；它尚未包含影像擷取、解碼、傳輸及顯示，不代表完整攝影機系統的吞吐。

4.2.2 FLOPs 變化與延遲變化

$$\Delta F = (F_{model} / F_{baseline} - 1) \times 100\% \quad (10)$$

$$\Delta T = (T_{model} / T_{baseline} - 1) \times 100\% \quad (11)$$

$$R = \Delta T / \Delta F \quad (12)$$

公式(10)與公式(11)分別表示相對 baseline 的 FLOPs 及延遲變化，公式(12)以兩者比值描述時間變化相對於計算量變化的大小。本文將 R 稱為變化兌現率，表列時再乘以 100%。這是本組實驗的描述性指標，並非硬體利用率； ΔF 接近零時比值容易被小分母放大，因而不適合用於比較。若部署流程的計算圖不同，也需明確標示 FLOPs 所屬的匯出圖。

4.2.3 精度評估與重現資料

物件偵測評估先以類別與位置判定預測是否正確。IoU 為預測框與標註框的交集面積除以聯集面積，precision 為預測中正確項目的比例，recall 為標註物件被找出的比例；AP 彙整 precision-recall 關係，mAP 再對類別取平均。本研究採 mAP50-95，並以固定信心門檻下的 precision、recall 補充操作點表現[24,25]。三個訓練種子用於觀察初始化及隨機程序的變異，同一權重的 60 次推論則反映執行時間波動。精度以平均與樣本標準差呈現，延遲以中位數呈現，統計單位分別為一次訓練與一次推論。

4.2.4 資料分割與權重選擇

VOC.yaml 使用 trainval 16,551 張影像訓練，並將 test2007 的 4,952 張同時列為 val 與 test[24]。本研究選取的 19 份訓練紀錄在第 100 epoch 達到最高 mAP50-95，與原先 best、last 評估摘要相同的紀錄一致。不過，最後 epoch 與最佳 epoch

一致，只能處理權重選擇的問題；若 test2007 曾參與架構、超參數或訓練長度選擇，其結果仍可能受到模型選擇影響。因此，本文將資料分割方式明確列出，後續研究應另設驗證分割，再以保留的測試集進行最終評估。

4.2.5 訓練與部署紀錄

實驗紀錄以模型設定、訓練 run、權重與部署檔相互對照。模型端保留程式、YAML、seed、超參數及 CSV；部署端則需對應匯出版本、張量形狀、執行緒設定與計時紀錄。本文依架構尺寸、訓練曲線及逐種子精度核對主要結果，第 4.11 節另以同批 VOC 影像比較三模型的改善與失敗情形。

4.3 主要模型結果

表 16 主要模型與位置消融的精度及 NCNN 延遲

模型	mAP50-95	延遲 ms	推論 FPS
baseline	0.4854 ± 0.0029	61.11	16.36
YOLO-CFR	0.5014 ± 0.0045	74.74	13.38
YOLO-LDS	0.4851 ± 0.0048	58.14	17.20
CFR-deep	0.4848	63.47	15.76

± 為三種子精度的樣本標準差（分母 n-1）；CFR-deep 僅一個種子。延遲欄未提供跨 session 誤差範圍，不能把精度標準差當作延遲誤差。

表 17 主要模型與位置消融的資源指標

模型	參數量	FLOPs G	Memory OPS M	FLOPs / Mem OPS
baseline	2,502,104	5.452	99.5	54.8
YOLO-CFR	2,523,216	6.612	118.2	55.9
YOLO-LDS	2,238,277	4.857	102.8	47.2
CFR-deep	2,535,352*	5.405	98.5	54.9

FLOPs 與 Memory OPS 採 PNNX 摘要，後者非實測 DRAM 流量。參數沿用原紀錄；* CFR-deep 對應 80 類 YAML 融合前數值，未與其餘列統一口徑。

表 16 比較主要模型，並附 CFR-deep 位置消融作為參考。CFR 的平均 mAP50-95 比 baseline 高 1.61 個百分點，匯出態 FLOPs 增加 21.3%，NCNN 延遲增加 22.3%。其提升伴隨較高執行成本，適合解讀為精度與速度之間的取捨。LDS 的平均精度為 0.4851，較 baseline 低約 0.03 個百分點，FLOPs 則減少

10.9%，推論時間減少 2.97 ms。這組結果呈現較接近的平均精度與較短的量測延遲，精度差異的統計解讀於第 4.4 節說明。

表 17 列出資源摘要。原參數紀錄中，CFR 增加約 0.84%，LDS 減少約 10.5%；由於參數量與類別數及模型狀態有關，這兩項比例仍需與相同狀態權重對照。CFR-deep 的 FLOPs 與 Memory OPS 略低於 baseline，延遲卻高 2.36 ms，說明在本組配置中，靜態成本的排序並未完全反映實機速度。此差異可作為後續算子分析的起點，現有摘要尚無法分離各層對總時間的貢獻。在模型選擇上，這些結果可轉為明確的延遲限制。例如，若只考慮本次 NCNN 的純推論中位數，並設定每張 60 ms 的假設預算，LDS 的 58.14 ms 低於此值，baseline 的 61.11 ms 與 CFR 的 74.74 ms 則高於此值；若預算放寬至 75 ms，三者都可列入比較，再依精度需求選擇。這只是用量測摘要示範選擇方式，不能視為任何單次推論都能滿足期限的保證。實際系統還需納入前後處理及尾端延遲，平均精度較高或典型推論較快，都不會自動形成適合所有情境的唯一配置。

4.4 精度與速度取捨及功能消融

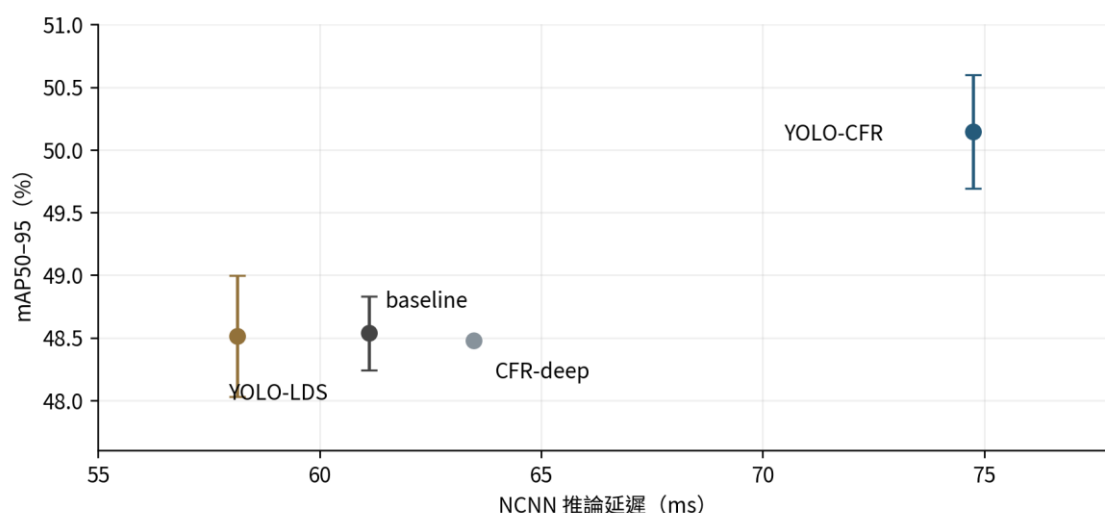


圖 9 主要模型精度與 NCNN 推論延遲

誤差棒為三種子精度的 ± 1 樣本標準差；CFR-deep 僅單種子。延遲未提供跨 session 誤差。

表 18 主要變體與功能消融之精度比較

變體	ΔmAP 百分點	原始 p	Holm 校正 p
YOLO-CFR	+1.61	0.0102	0.0511
YOLO-LDS	-0.03	0.9427	1.0000
ghost	-0.14	0.7584	1.0000
eca	-0.67	0.0668	0.2003
eca_ghost	-0.91	0.0444	0.1778

雙尾 Welch t 檢定；每組 $n=3$ 。由原始 CSV 五位小數數據重算；Holm 將五項對 baseline 的比較視為同一檢定家族[27]。

圖 9 呈現精度與 NCNN 延遲的關係，CFR 位於較高精度、較長延遲的位置，LDS 則以接近 baseline 的平均精度換得較短時間。表 18 同時列出主要變體與功能消融相對 baseline 的雙尾 Welch 檢定。CFR 的原始 p 值為 0.0102，將五項比較納入 Holm 校正後為 0.0511；eca_ghost 的 p 值則由 0.0444 調整為 0.1778[27]。在此多重比較設定下，CFR 未達 0.05 門檻，本文因此以平均差、種子分布及成本變化描述結果。LDS 的 $p=0.9427$ 表示未檢出精度差異，並不等同於已證明精度等效[28]。兩個主要模型各有不同取捨，應依應用的精度要求及延遲限制選擇。功能消融也顯示，加入元件後的精度不一定提升。ghost、eca 與 eca_ghost 的平均 mAP_{50-95} 分別為 0.4840、0.4786 與 0.4763，均未超過 baseline 的 0.4854。在本組訓練條件下，單獨的元件名稱不足以預測組合結果，模組位置、通道寬度及前後層的配合仍需一併考慮。尤其 eca_ghost 的平均值低於 ghost，不能據此推定所有 GhostConv 後接注意力的設計都會退步，因為其配置尚未與 YOLO-LDS 逐項對齊。這組消融較能支持的判斷是：主要模型的結果來自具體的完整架構，不能由個別元件的既有研究成效直接推導，也不能將數個消融的平均差相加，當作組合模型的預期增益。

4.4.1 各種子結果與訓練變異

表 19 各模型的逐種子精度

模型	seed 0	seed 1	seed 2	平均	樣本 std
baseline	0.48859	0.48468	0.48283	0.4854	0.0029
YOLO-CFR	0.50039	0.49752	0.50643	0.5014	0.0045
YOLO-LDS	0.49025	0.48445	0.48064	0.4851	0.0048
ghost	0.48304	0.49083	0.47808	0.4840	0.0064
eca	0.47728	0.47596	0.48266	0.4786	0.0035
eca_ghost	0.47153	0.47964	0.47772	0.4763	0.0042

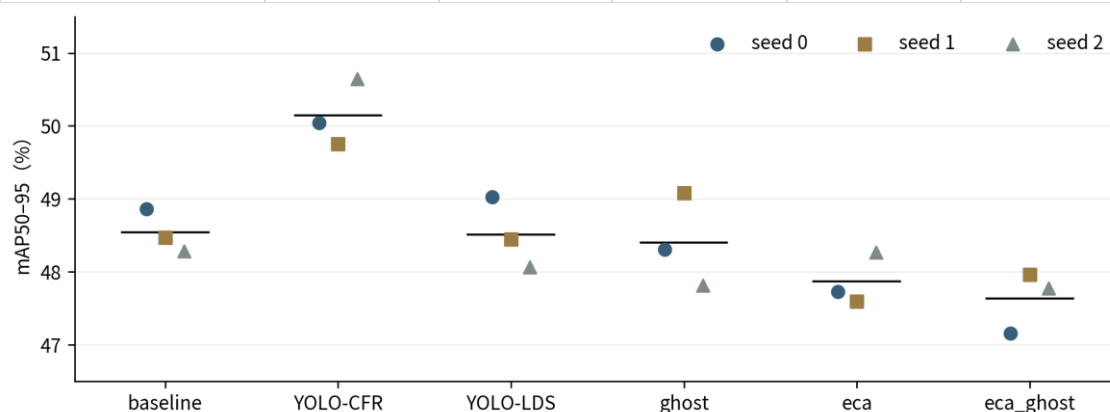


圖 10 各模型的逐種子精度分布

每點代表一次訓練，黑色短線為三種子平均；圖中不以線段連接不同模型的相同 seed。

表 19 與圖 10 列出各次訓練結果。CFR 的三次 mAP50-95 均高於 baseline 的三次結果，顯示平均改善並非由單一高分種子主導。ghost 的結果介於 0.47808 至 0.49083，樣本標準差約 0.0064，為表中六組最高者；相較之下，其平均差距較小，需連同訓練變異解讀。進一步比較分布範圍，CFR 的最低值 0.49752 仍高於 baseline 的最高值 0.48859，差距約為 0.893 個百分點；LDS 的範圍則涵蓋 baseline 的三次結果。這有助於區分「每次觀察的方向是否一致」與「平均數是否接近」兩個問題，但三次訓練仍不足以完整描述可能的結果分布。尤其資料增強、批次順序及權重初始化都可能隨隨機程序改變，單次最佳結果不宜取代三種子摘要。本文同時保留逐種子數值與平均值，讓配置間的差距能與配置內的波動一起閱讀。Welch 檢定將各模型訓練結果視為獨立樣本。不同架構即使採相同 seed，也未必消耗相同隨機序列，因此本文不單憑種子編號相同採用配對檢定。各組僅三次訓練，LDS 減 baseline 的平均差之 Welch 95% 信賴區間約為 -1.01 至 +0.96 個百分點。此區間描述所觀察的訓練變異，未包含測試影像抽樣誤差。若要進一步

主張精度等效，需先定義具有應用意義的等效界限，再配置足夠的重複實驗 [28]。

4.4.2 固定信心門檻的偵測表現

固定門檻的補充評估使用 test2007 中 500 張影像，共含 1,895 個標註目標，配對條件為同類別且 $\text{IoU} \geq 0.5$ 。表 20 與圖 11 比較五個 confidence 門檻下的 precision (P) 與 recall (R)。此分析描述指定門檻的實際操作點，與完整測試集跨門檻彙整的 mAP50-95 具有不同用途。

表 20 固定信心門檻下的 precision 與 recall

conf	baseline P / R	YOLO-CFR P / R	YOLO-LDS P / R
0.10	0.503 / 0.756	0.500 / 0.773	0.496 / 0.749
0.20	0.663 / 0.681	0.664 / 0.692	0.653 / 0.674
0.25	0.722 / 0.651	0.718 / 0.659	0.711 / 0.648
0.35	0.801 / 0.584	0.796 / 0.597	0.809 / 0.593
0.50	0.889 / 0.493	0.880 / 0.513	0.889 / 0.499

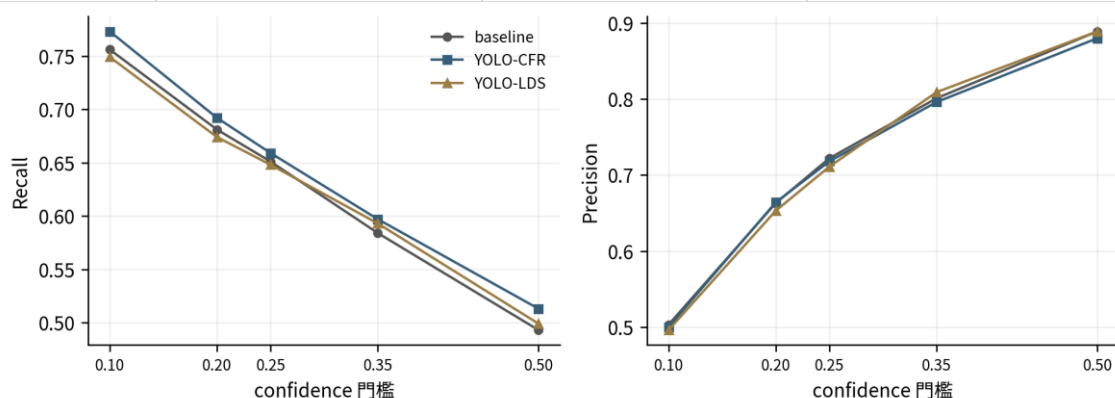


圖 11 固定信心門檻下的 precision 與 recall

CFR 在五個門檻下的 recall 均高於 baseline，差距依序為 1.7、1.1、0.8、1.3、2.0 個百分點，precision 的絕對差均小於 0.01。提高門檻時，各模型大致呈現 precision 上升、recall 下降，反映誤檢與漏檢之間的取捨。在本子集中，CFR 的優勢主要表現在較高召回；不同模型的信心分數校準可能不同，若要比較相同 precision 下的 recall，仍需更密集的門檻掃描。此表為單組權重的子集結果，未提供跨種子誤差，案例清單與配對程序也需與權重版本一同保存。以門檻 0.25 為例，baseline 與 CFR 的 recall 分別為 0.651 與 0.659，而 precision 分別為 0.722 與 0.718，呈現較多目標被找出、正確預測比例略降的組合。若將門檻提高至 0.50，三個模型的 precision 均接近 0.88~0.89，但 recall 約為 0.49~0.51。這表示單純提

高信心門檻，雖可降低部分誤檢，也可能排除相當數量的正確候選。對漏檢成本較高的任務，門檻選擇不能只追求 precision；對需要人工複核的流程，則須同時考慮誤檢帶來的處理負擔。門檻應在獨立驗證資料上依用途設定，再用測試資料評估，避免為了改善測試數字反覆調整。

4.5 CFR 位置消融

表 21 不同 CFR 位置的比較

變體	CFR 層號	訓練態 FLOPs G	匯出態 FLOPs G	mAP50-95
baseline	無	6.1	5.452	0.4854
CFR-all	2、4、6、8	7.3	6.612	0.5014
CFR-deep	6、8	6.1	5.405	0.4848

CFR-all 即 YOLO-CFR；原紀錄的訓練態與匯出態 FLOPs 分列；訓練態值與 80 類 YAML 摘要一致，類別口徑待核對。前兩列精度為三種子平均，CFR-deep 為單種子。

表 21 比較完整 CFR 與深層配置。CFR-deep 的 mAP50-95 為 0.4848，接近 baseline，未保留完整 CFR 的平均精度增益；其匯出態 FLOPs 略低於 baseline，推論時間卻較長。僅保留第 6、8 層修改，在本次實驗中未取得預期的精度與速度組合。這項結果說明配置位置值得考慮，但完整配置與深層配置之間同時包含淺層模組及跨層交互作用的差異，尚不能將增益完全歸於淺層。YOLO-LDS 於第 8 層保留 CFR，其平均精度高於 ghost 約 0.11 個百分點。由於兩者其餘模組及位置尚未確認為完全相同，該差值僅作完整配置的比較，不視為 CFR@8 的獨立貢獻。後續位置分析可加入僅修改第 2、4 層的 CFR-shallow，並以相同種子數比較完整、淺層與深層配置。

4.5.1 成本與精度的解讀

CFR-all 的精度與計算量同時上升，代表觀察到的效果包含運算容量改變。若要評估其在相同成本下是否更有效，需建立相近 FLOPs 或相近實測延遲的基準，例如調整原模型的深度或寬度，再與 CFR 比較。進一步的路徑消融則應固定隱藏通道、殘差及單元數，僅切換目標連接。這些比較可將配置效益與一般容量增加區分，並補足目前單種子 CFR-deep 的證據範圍。

4.6 推論流程比較

表 22 ONNX 與 NCNN 推論時間

模型	ONNX ms	NCNN ms	時間比值 ONNX / NCNN
baseline	122.40	61.11	2.00×
YOLO-CFR	150.27	74.74	2.01×
YOLO-LDS	119.22	58.14	2.05×

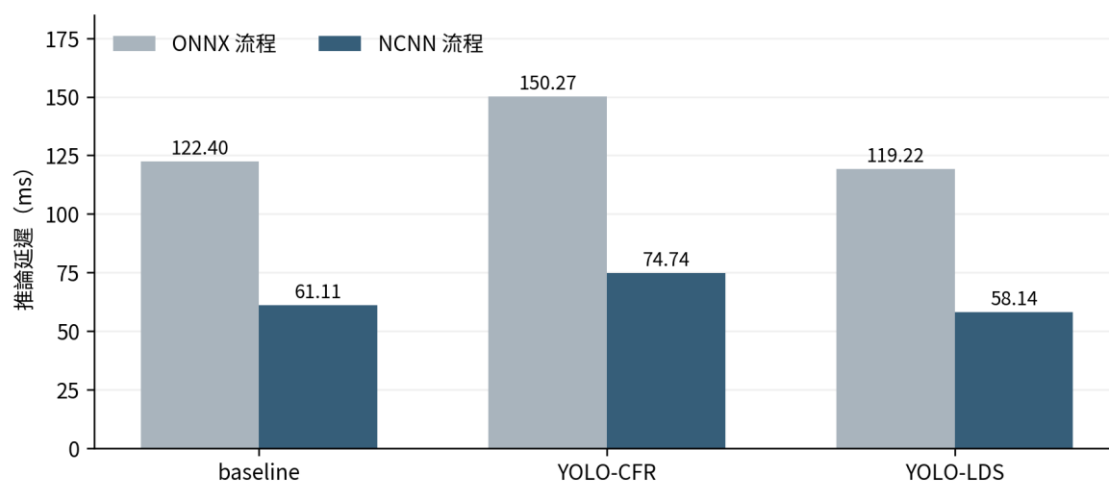


圖 12 各模型在兩套部署流程的推論時間

表 22 與圖 12 顯示，三個主要模型在 NCNN 流程中的推論時間約為 ONNX 的一半，時間比值介於 2.00 至 2.05。baseline 從 122.40 ms 降至 61.11 ms，減少 61.29 ms；相較之下，在 NCNN 流程內將 baseline 改為 LDS，減少 2.97 ms。就本組量測而言，部署流程選擇對總時間的影響大於本次局部架構輕量化。此比較同時涵蓋框架、匯出圖與輸出分支，因而不能將全部差異歸於某一算子的實作。Ultralytics 的 Pi 5 文件列出 YOLO26n ONNX 125.99 ms、NCNN 67.03 ms[7]，其量級與排序和本研究相近；外部測試的類別數及軟體條件不同，僅用於背景參照。要進一步分離框架因素，需固定輸出分支與後處理程序，再分析各層時間。兩套流程內的模型排序均為 LDS 最快、baseline 次之、CFR 最慢，說明本次相對方向並未隨流程切換而反轉；然而，LDS 相對 baseline 節省的時間分別為 3.18 ms 與 2.97 ms，其占各自基準時間的比例不同。因此，評估部署改善時需同時保留絕對毫秒數和相對百分比。前者表示每張影像實際節省多少時間，後者表示對當前流程成本的影響，兩者分別服務於時間預算與模型比較。

4.7 前處理與後處理成本

表 23 baseline 三階段延遲摘要

部署流程	前處理 ms	推論 ms	後處理 ms	摘要合計 ms
ONNX	4.77	122.40	0.33	127.50
NCNN	2.83	61.11	0.86	64.80

表 23 列出 baseline 的前處理、推論及後處理摘要。ONNX 的推論占階段摘要合計約 96.0%，NCNN 約 94.3%，顯示本次量測的主要時間仍集中於網路運算。由純推論時間換算，兩者分別為 8.17、16.36 FPS；若以三階段摘要合計換算，則為 7.84、15.43 FPS。這些數值尚未計入攝影機及顯示流程，但已說明純推論速度與包含前後處理的估計吞吐不同。三階段時間也可用來估計最佳化的作用範圍。以 NCNN 的摘要合計 64.80 ms 為例，前後處理合計為 3.69 ms。若在推論時間保持 61.11 ms 的假設下，將這兩部分完全省略，摘要合計最多下降約 5.7%；這是一個依現有數值計算的理想上限，並非可實現的系統量測。相對而言，卷積與其他網路算子的時間仍占主要部分，後續效能分析應先確認推論階段的耗時分布。這個判斷也有適用範圍：目前後處理使用的輸入條件固定，若真實影像產生更多候選框，或加入影像解碼及顯示流程，時間比例便可能改變。因此，模型層級的最佳化與完整影像系統的最佳化，需要依各自的計時範圍安排優先順序。

4.7.1 輸出分支與 NMS

ONNX 輸出為 [1,300,6]，NCNN 則為 [1,24,8400]，後者需要整理密集預測並執行 NMS。兩者後處理摘要相差 0.53 ms，此差值也包含解碼、格式轉換與輸出筆數的影響，不能直接當作 NMS 的獨立時間。本次後處理量測使用合成影像，候選框分布可能與真實場景不同；當重疊物件或高分候選框增加時，仍需重新評估後處理成本。因此，現有結果描述本次輸入與流程的典型時間，尚不足以判定 NMS-free 設計在所有場景下的時間效益。

4.7.2 匯出精度與使用情境

精度表與部署時間表分別來自模型評估與 Pi 量測，尚未逐一列出 PyTorch、ONNX、NCNN 在完整測試集的精度。因此，時間改善不附帶跨格式精度完全相同的假設。後續一致性驗證需固定前處理、confidence、IoU、max_det 及輸出分支，再比較部署 mAP。應用端也需先定義更新頻率與延遲期限；本研究約 13–17 FPS 的純推論結果，尚不構成 30 FPS 攝影機系統的完整驗證。

4.8 FLOPs 與延遲的落差

表 24 相對 baseline 的成本與延遲變化

模型	Δ FLOPs	Δ NCNN 延遲	變化兌現率
YOLO-CFR	+21.3%	+22.3%	約 105%
YOLO-LDS	-10.9%	-4.9%	約 45%
CFR-deep	-0.9%	+3.9%	不採用

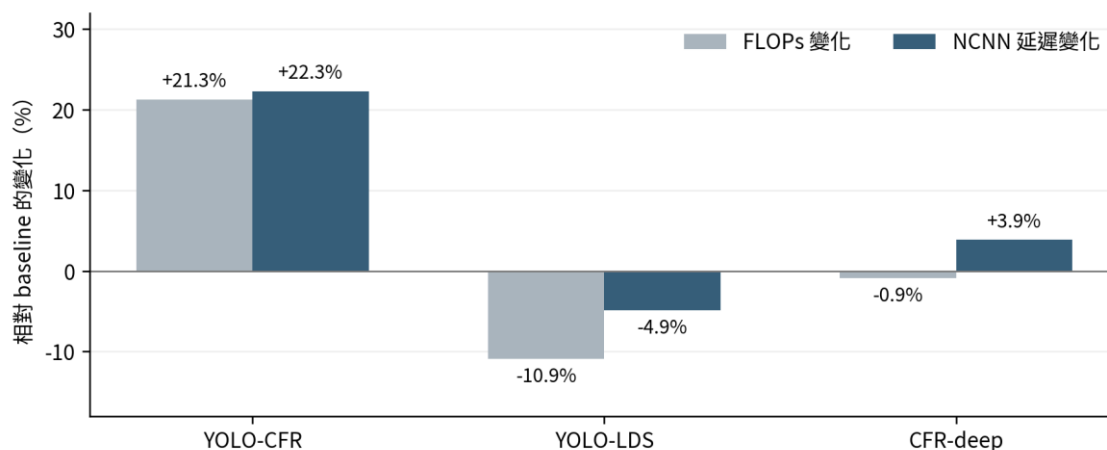


圖 13 理論計算量與實測延遲的相對變化

表 24 與圖 13 比較相對 baseline 的計算量與時間變化。CFR 的 FLOPs 增加 21.3%，NCNN 延遲增加 22.3%，兩者變化接近；LDS 的 FLOPs 減少 10.9%，延遲則減少 4.9%，時間降幅明顯較小。CFR-deep 的 FLOPs 僅減少約 0.9%，延遲反而增加約 3.9%，其比值受小分母影響，因而不列變化兌現率。這些結果說明，在相同部署平台上，局部架構的 FLOPs 變化仍不足以直接決定延遲排序。以 NCNN 匯出態 FLOPs 為共同參照時，CFR 與 LDS 在 ONNX 流程中的描述性比值約為 107%、24%，在 NCNN 中則約為 105%、45%。LDS 的時間改善在兩套流程不同，表示相同架構修改會與執行環境產生交互影響。由於尚未獨立統計 ONNX 計算圖，這組比值僅用於整理現象，不作跨框架算子效率的直接估計。若假設架構修改後，每一單位 FLOPs 的執行成本都保持不變，將 baseline 的 61.11 ms 乘以 LDS 的 FLOPs 比值 4.857/5.452，可得到約 54.44 ms。實際摘要為 58.14 ms，比這個單純比例估計高約 3.70 ms。這項差距說明以 FLOPs 等比例預測延遲，會高估本配置的加速幅度；但它不是任何特定算子的實測耗時，不能把 3.70 ms 全部歸為 ECA、拼接或資料搬移。修改同時改變多個算子的工作量與執行方式，且仍有大量共同網路未被替換，只有細部量測才能分辨各部分對總時間的貢獻。

4.9 記憶體行為與可能機制

LDS 的 PNNX FLOPs 由 5.452G 降至 4.857G，Memory OPS 卻由 99.5M 增至 102.8M，增加約 3.3%；兩者比值由 54.8 降至 47.2。此結果顯示，減少算術運算的同時，圖層級資料處理摘要並未下降，可作為分析延遲降幅較小的線索。依 Roofline 的定義[21]，與硬體頻寬比較的算術強度需以實際傳輸位元組及特定記憶體層級為基礎。現有 Memory OPS 未包含完整的快取重用、融合與執行資料布局資訊，因此本文將其用於靜態比較，不直接認定模型已受 DRAM 頻寬限制。

4.9.1 算子類型的影響

GhostConv 以較低成本分支取代部分密集通道混合，同時增加特徵生成與拼接的步驟；ECA 則需讀取特徵圖以計算通道統計，再執行逐通道乘法。這些操作的延遲取決於工作量、核心實作與圖最佳化。concat、split、reshape 是否造成實體複製，也需依執行方式判定，不能一律視為完整資料搬移。depthwise 卷積雖減少跨通道計算，其權重與輸入仍具有可重用性，實際成本需連同尺寸、核大小及快取行為分析。相同 FLOPs 也可能分布在不同大小的運算上。單一較大的卷積與數個較小的卷積，即使乘加總數接近，仍可能具有不同的工作分配、緩衝區使用及平行執行效率。對 LDS 而言，一般卷積、depthwise 卷積、拼接與注意力依序銜接，某一部分運算減少後，其他操作占總時間的比例便可能上升。分析時應先量測各部分的占比，再決定要減少算術運算、合併操作或改善資料布局。現有整體延遲只能顯示最後的淨效果，無法據此選定其中一項作為已確認的主要瓶頸。

4.9.2 匯出紀錄的證據範圍

LDS 匯出紀錄出現三次 `insert_reshape_global_pooling_forward`，與三個 LDS 模組的數量相符，提供追查池化相關圖轉換的線索。訊息本身不包含各次轉換的執行時間，也不足以判定是否造成頻寬瓶頸。後續可先取得各層或各算子的耗時，找出總時間集中的位置，再結合快取未命中、記憶體流量及執行緒設定分析。這樣可將目前由靜態摘要提出的假設，轉為可直接驗證的效能機制。

4.10 未成功的嘗試與研究限制

表 25 未成功的探索結果

嘗試	結果	可支持的判讀
KalmanPSA	mAP 0.4575 -2.79 個百分點	本次配置低於 baseline
KalmanPSA-bidir	mAP 0.4569 -2.85 個百分點	本次雙向配置未改善
RepConv 重參數化	未完成整合	屬實作整合限制，非效能結果

表 25 整理未取得預期效果的探索。KalmanPSA 與雙向版本的 mAP 均低於 baseline，表示本次狀態更新配置未帶來改善；現有結果尚未分離狀態定義、遞推軸及觀測模型的影響。RepConv 則未完成整合，因此沒有對應的性能結果。RepVGG 透過結構重參數化，將訓練時的多分支轉為推論時的簡潔結構[29]。在 RepConv 實作中， $k=3$ 、 $p=1$ 的限制並不代表 stride 只能為 1，stride 會傳入卷積分支，identity BN 分支才另有 $s=1$ 條件[30]。後續整合需回查模型解析與分支相容性，而非將未完成整合解釋為方法無效。

4.10.1 實驗外推的限制

本研究的實驗範圍限定於 VOC、100 epochs、Pi 5 CPU 與 FP32，尚未涵蓋 COCO[31]、其他訓練預算、量化或不同硬體。主要模型與功能消融各有三個種子，CFR-deep 僅一個種子，因而對變異及等效性的估計仍有限。VOC 的 val 與 test 使用相同影像，亦需考慮架構選擇可能帶來的偏差。CFR 的實際 nano 配置未啟用跨單元傳遞，且隱藏通道與原版不同，故目前結論限於完整配置的效果。部署端每模型取樣 60 次，現有資料以中位數摘要為主，尚缺 p90、跨 session 變異、完整執行緒設定與逐次 Pi 紀錄。參數量亦需以相同類別數、融合狀態及權重重重新核對；跨格式精度、真實影像後處理和長時間串流未完成完整評估。這些限制不改變已列出的實驗數值，但界定了其適用條件：本文能比較既定訓練與部署流程中的結果，尚不能將速度差異普遍外推至其他環境，也不能僅憑靜態摘要確認記憶體瓶頸。

4.11 實際偵測案例與選取方式

偵測案例從 500 張掃描影像中以 spread 模式選取，信心門檻為 0.25，配對 IoU 門檻為 0.5。圖 14 至圖 16 比較同一畫面的標註、baseline、CFR 與 LDS 輸出，藍框為標註，綠框為 TP，紅框為 FP，黃色虛線框為 FN。六張案例的統計整理於

表 26。案例以呈現模型差異為目的，並非隨機抽樣，因此用於質性說明，不據此估計整體模型排名。



圖 14 影像 008863 的標註與三模型結果

圖 14 的牛群共有 7 個標註。baseline 得到 TP 1、FP 8、FN 6，並可見多個 sheep 類別預測；CFR 得到 TP 6、FP 3、FN 1，LDS 為 TP 6、FP 0、FN 1。兩個修改模型在此畫面均找出更多正確目標，其中 LDS 同時減少誤檢。這個結果具體呈現固定門檻下分類與漏檢的改善，但單一影像仍混合類別、框位置及信心分數的影響，不能由此判定模型已學得某一特定特徵機制。以該張影像的計數換算，baseline 的 precision 為 1/9、recall 為 1/7；CFR 分別為 6/9、6/7；LDS 則為 6/6、6/7。CFR 與 LDS 找到的正確目標數相同，但輸出預測的正確比例不同，說明只比較 TP 或 recall 會漏掉誤檢差異。這些分數僅用於解釋該案例，並非整體測試集的 precision 或 recall。另由影像中的類別標籤可以提出分類混淆的觀察，但若將錯誤歸因於某種紋理、尺度或遮擋，仍需逐框配對與更多同類條件影像，不能由單張成功結果推定特徵機制。

4.11.1 CFR 退步的案例

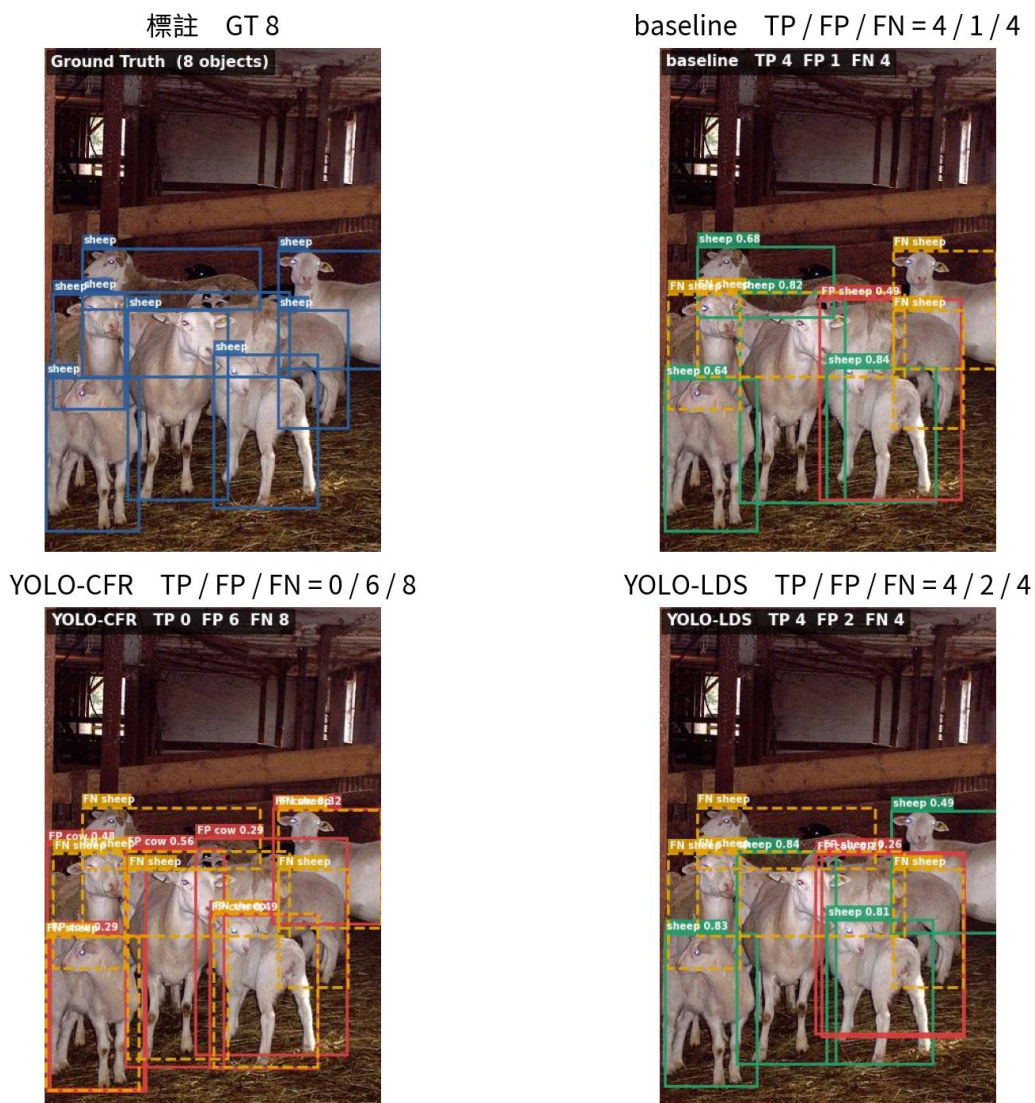


圖 15 影像 006217 的標註與三模型結果

圖 15 共有 8 個標註，baseline 為 TP 4、FP 1、FN 4，CFR 為 TP 0、FP 6、FN 8，LDS 為 TP 4、FP 2、FN 4。雖然 CFR 的整體平均 mAP 較高，在此畫面卻未取得正確配對，說明平均改善與個別失敗可以同時存在。TP、FP、FN 由類別與位置門檻共同決定，預測框即使靠近目標，若類別不符或 IoU 未達門檻，仍可能形成 FP 與 FN。因此，案例中的零 TP 不表示模型完全沒有產生回應，分析時需區分類別錯誤、定位偏差及信心門檻的影響。本文沿用既有案例圖片與統計。由於未保存於本次資料中的逐框數值、權重種子及案例生成程式尚待補齊，案例未重新計算配對。後續可在固定權重和一對一配對規則下輸出框座標、類別與信心值，以進一步區分各種錯誤來源。若進一步分析此類失敗，可先固定位置配對條件，檢查預測類別是否正確，再觀察正確類別的候選框是否因信心門檻被移除，以及同一目標是否出現多個重複框。這些檢查分別對應分類、分數與配對程序，

能避免把所有 FN 都解釋為網路沒有辨識到物件。對目前案例而言，圖中可見多個預測框，而 CFR 的 TP 仍為零，正好說明「產生回應」與「通過偵測評估」是不同條件。後續修正若要針對原因，必須先找出未通過的是哪一項條件，再判斷應調整訓練、定位能力或部署門檻。

4.11.2 多目標場景與案例總覽

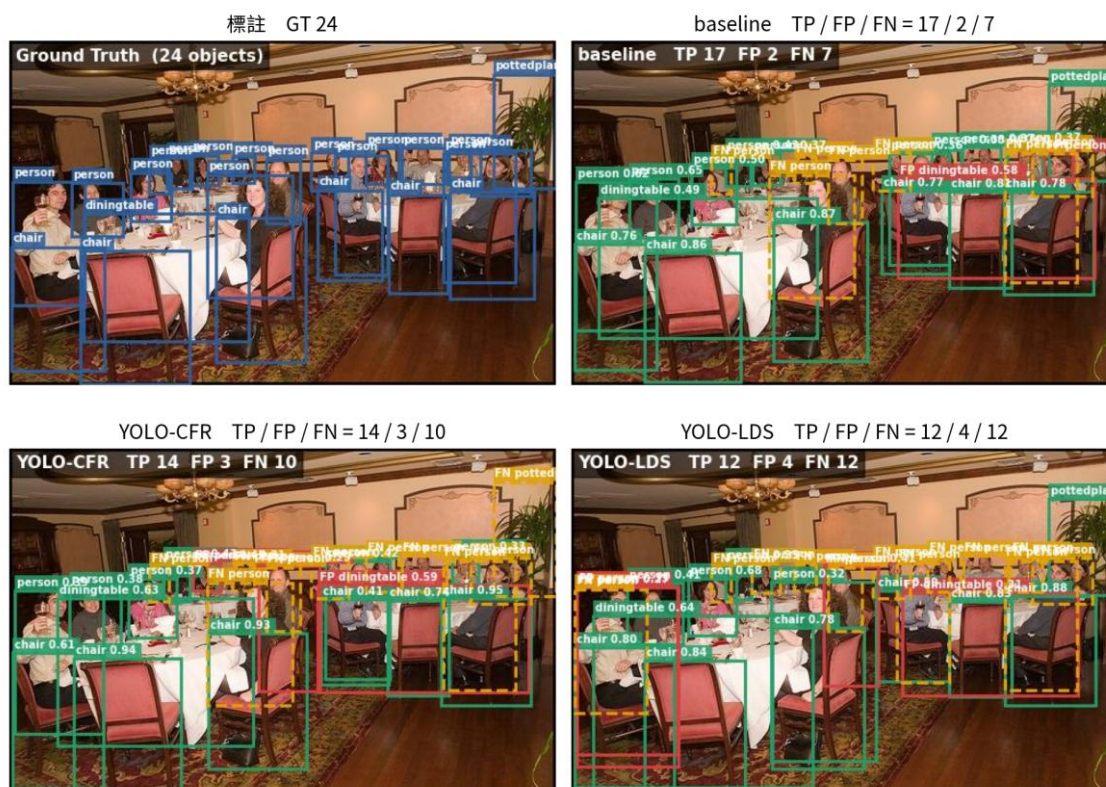


圖 16 影像 009854 的標註與三模型結果

圖 16 包含 24 個標註，baseline、CFR、LDS 的 TP 分別為 17、14、12，FN 分別為 7、10、12，兩個修改模型在此多目標畫面均未優於 baseline。表 26 另外列出三張案例：008863 以外的結果也呈現改善與退步並存，例如 004819 中 CFR 的 TP 由 8 增至 10，但 002818 中 LDS 的 TP 由 5 降至 1。這些差異說明局部架構修改不會一致改善所有場景。由於六張影像採差異選取，其統計不彙總為測試集指標；小物件或特定類別的能力，仍需透過相應子集與逐類別評估確認。

其餘案例進一步呈現誤檢與漏檢未必同步改善。影像 007644 中，CFR 的 FP 從 baseline 的 2 降為 0，但 TP 也由 6 降為 4，FN 從 9 增至 11；若只看誤檢數，便會忽略少找出兩個目標的代價。影像 004819 中，CFR 將 TP 提高至 10，同時增加 1 個 FP；LDS 雖也將 TP 提高至 9，FP 卻增加至 5。兩者在該畫面的取捨並不相同。這些例子說明，部署時應先釐清漏檢與誤檢的可接受程度，再決定信心門檻及模型，而不能僅挑選視覺上框線較多或誤檢較少的結果。案例圖可揭露這類具體差異，但每張影像的物件數、遮擋及類別組成不同，仍需回到共同測試集才能衡量整體效益。

表 26 六張案例的 TP 與 FP 及 FN

影像	GT	baseline	CFR	LDS
008863	7	1 / 8 / 6	6 / 3 / 1	6 / 0 / 1
006217	8	4 / 1 / 4	0 / 6 / 8	4 / 2 / 4
009854	24	17 / 2 / 7	14 / 3 / 10	12 / 4 / 12
002818	6	5 / 4 / 1	4 / 3 / 2	1 / 7 / 5
007644	15	6 / 2 / 9	4 / 0 / 11	2 / 4 / 13
004819	11	8 / 0 / 3	10 / 1 / 1	9 / 5 / 2

模型欄依序為 TP / FP / FN。六張採差異選取，不彙總為整體模型排名。

第五章 結論與未來發展

5.1 結論

本研究以 YOLO26n 為基準，完成 CFR、LDS 兩種架構配置，以及共同條件下的多種子訓練、模型匯出與 Raspberry Pi 5 部署量測。CFR 的平均 mAP50-95 為 0.5014，較 baseline 增加 1.61 個百分點，NCNN 推論延遲增加至 74.74 ms；LDS 的平均 mAP50-95 為 0.4851，與 baseline 接近，延遲降至 58.14 ms。兩者分別呈現較高精度與較低延遲的方向，其選擇需依應用條件取捨。工程實作亦保留自訂模組、逐層架構、訓練曲線與偵測案例，使精度及部署結果能對應至實際配置。LDS 的匯出態 FLOPs 減少 10.9%，NCNN 延遲僅減少 4.9%；CFR-deep 也出現 FLOPs 略低但延遲較高的結果。三個主要模型在 NCNN 流程中的推論時間約為 ONNX 的一半，顯示部署流程對效能具有明顯影響。這些結果支持以目標硬體的實測時間評估架構，而非只依參數量或計算量排序。CFR 的增益屬於本次 nano 完整配置，跨單元傳遞與記憶體瓶頸等機制仍需獨立驗證；兩套部署流程的輸出分支差異，也界定了框架比較的解讀範圍。

5.2 後續驗證與發展方向

後續工作應先以 checkpoint 核對模型版本、類別數及內部配置，將訓練、匯出與部署紀錄連結至同一權重，再確認跨格式精度與時間穩定性。在相同測試集上固定前處理、confidence、IoU、max_det 及輸出分支，比較 PyTorch、ONNX 與 NCNN 的結果，並以真實影像量測推論及 NMS。時間量測應涵蓋多個 session，交錯模型順序，保存 CPU 時脈、溫度及執行緒設定，報告中位數、p90 和變異範圍，以確認 LDS 約 3 ms 優勢的穩定程度。架構研究可加入 CFR-shallow、多種子位置消融及相同成本基準，並固定隱藏通道、殘差與單元數，進一步分析特定連接的作用。部署最佳化則可由逐層耗時出發，評估解析度、執行緒數與量化。Jacob 等人的整數運算推論及量化訓練方法[32]可作為 INT8 延伸基礎，但須同時確認精度與目標平台的實際速度。LDS 目前為 17.20 純推論 FPS，距離 30 FPS 仍需約 1.74 倍加速，完整系統還須計入影像擷取、前後處理及顯示。

參考文獻

- [1] Mingxing Tan et al., “MnasNet: Platform-Aware Neural Architecture Search for Mobile,” CVPR, 2019. <https://arxiv.org/abs/1807.11626>
- [2] Ultralytics, “Ultralytics YOLO26,” official model documentation. <https://docs.ultralytics.com/models/yolo26/>
- [3] Ao Wang et al., “YOLOv10: Real-Time End-to-End Object Detection,” NeurIPS, 2024. <https://arxiv.org/abs/2405.14458>
- [4] Kai Han et al., “GhostNet: More Features from Cheap Operations,” CVPR, 2020. <https://arxiv.org/abs/1911.11907>
- [5] Qilong Wang et al., “ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks,” CVPR, 2020. <https://arxiv.org/abs/1910.03151>
- [6] Tencent, “ncnn,” official source repository and deployment documentation. <https://github.com/Tencent/ncnn>
- [7] Ultralytics, “YOLO26 on Raspberry Pi Setup & Benchmarks,” official deployment guide. <https://docs.ultralytics.com/guides/raspberry-pi/>
- [8] Joseph Redmon et al., “You Only Look Once: Unified, Real-Time Object Detection,” CVPR, 2016. <https://arxiv.org/abs/1506.02640>
- [9] Wei Liu et al., “SSD: Single Shot MultiBox Detector,” ECCV, 2016. <https://arxiv.org/abs/1512.02325>
- [10] Tsung-Yi Lin et al., “Feature Pyramid Networks for Object Detection,” CVPR, 2017. <https://arxiv.org/abs/1612.03144>
- [11] Shu Liu et al., “Path Aggregation Network for Instance Segmentation,” CVPR, 2018. <https://arxiv.org/abs/1803.01534>
- [12] Glenn Jocher et al., “Ultralytics YOLO26: Unified Real-Time End-to-End Vision Models,” arXiv:2606.03748, 2026 (預印本). <https://arxiv.org/abs/2606.03748>
- [13] Andrew G. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” arXiv:1704.04861, 2017 (預印本). <https://arxiv.org/abs/1704.04861>
- [14] Mark Sandler et al., “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” CVPR, 2018. <https://arxiv.org/abs/1801.04381>
- [15] Xiangyu Zhang et al., “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” CVPR, 2018. <https://arxiv.org/abs/1707.01083>
- [16] Chien-Yao Wang et al., “CSPNet: A New Backbone that can Enhance Learning Capability of CNN,” CVPR Workshops, 2020. <https://arxiv.org/abs/1911.11929>
- [17] Jie Hu, Li Shen, and Gang Sun, “Squeeze-and-Excitation Networks,” CVPR, 2018. https://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html
- [18] Andrew Howard et al., “Searching for MobileNetV3,” ICCV, 2019. <https://arxiv.org/abs/1905.02244>
- [19] Mingxing Tan and Quoc V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML, 2019. <https://arxiv.org/abs/1905.11946>
- [20] Ningning Ma et al., “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design,” ECCV, 2018. <https://arxiv.org/abs/1807.11164>
- [21] Samuel Williams, Andrew Waterman, and David Patterson, “Roofline: An Insightful Visual Performance Model for Multicore Architectures,” Communications of the ACM, vol. 52, no. 4, pp. 65–76, 2009. <https://doi.org/10.1145/1498765.1498785>

- [22] Ji Lin et al., “MCUNet: Tiny Deep Learning on IoT Devices,” NeurIPS, 2020. <https://arxiv.org/abs/2007.10319>
- [23] Microsoft, “ONNX Runtime: Thread management,” official performance tuning documentation. <https://onnxruntime.ai/docs/performance/tune-performance/threading.html>
- [24] Ultralytics, “PASCAL VOC Detection Dataset,” dataset configuration documentation. <https://docs.ultralytics.com/datasets/detect/voc/>
- [25] Mark Everingham et al., “The PASCAL Visual Object Classes (VOC) Challenge,” International Journal of Computer Vision, vol. 88, pp. 303–338, 2010. <https://doi.org/10.1007/s11263-009-0275-4>
- [26] Ilya Loshchilov and Frank Hutter, “Decoupled Weight Decay Regularization,” ICLR, 2019. <https://arxiv.org/abs/1711.05101>
- [27] Sture Holm, “A Simple Sequentially Rejective Multiple Test Procedure,” Scandinavian Journal of Statistics, vol. 6, no. 2, pp. 65–70, 1979. <https://www.jstor.org/stable/4615733>
- [28] Daniël Lakens, “Equivalence Tests: A Practical Primer for t Tests, Correlations, and Meta-Analyses,” Social Psychological and Personality Science, vol. 8, no. 4, pp. 355–362, 2017. <https://doi.org/10.1177/1948550617697177>
- [29] Xiaohan Ding et al., “RepVGG: Making VGG-style ConvNets Great Again,” CVPR, 2021. <https://arxiv.org/abs/2101.03697>
- [30] Ultralytics, “nn.modules.conv API Reference,” RepConv implementation. <https://docs.ultralytics.com/reference/nn/modules/conv/>
- [31] Tsung-Yi Lin et al., “Microsoft COCO: Common Objects in Context,” ECCV, 2014. <https://arxiv.org/abs/1405.0312>
- [32] Benoit Jacob et al., “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” CVPR, 2018. <https://arxiv.org/abs/1712.05877>