XR × 影像辨識 × PPO

AI 麻將輔助系統

VR Mahjong Assistance System

F74126092｜譚永祺

Quest / Unity | YOLO Segmentation | Classification | PPO | Quest UI

Final Report

SECTION

Quest 3 基礎介紹

Meta Quest 3 的功能、MR / VR 應用與開發定位

VR / MR

Passthrough

Unity / APK

Meta Quest 3 基礎介紹

一台可獨立運作的 VR / MR 頭戴裝置，能把虛擬內容帶進使用者視野與真實空間

VR / MR Headset

Meta Quest 3 不是單純螢幕，而是能追蹤頭部、手部與空間的互動平台。

獨立運作 ｜ 全彩穿透 ｜ 手把 / 手勢互動

顯示與視野

4K+ Infinite Display，每眼 2064×2208，更新率 72 / 90 / 120Hz，水平視野約 110°。

全彩穿透 MR

可看到真實環境，再把 3D 物件、提示或介面疊到現實空間中。

效能與互動

Snapdragon XR2 Gen 2、8GB RAM；支援 Touch Plus 控制器與手部 / 身體追蹤。

裝置定位

適合 VR 體驗、MR 展示、教學模擬與互動原型；不是桌機 GPU 等級運算平台。

資料來源：Meta 官方 Quest 3 / 3S 比較頁、Meta Quest Help Center

Meta Quest 3 可以做什麼？

重點能力可以分成沉浸式 VR、Mixed Reality、日常使用與開發部署四個方向

1. 沉浸式 VR 體驗

進入完整虛擬場景，用頭部、控制器或手勢互動；常見於遊戲、訓練、教育展示。

2. Mixed Reality 疊加資訊

利用全彩穿透把虛擬物件放到真實桌面、房間或使用者眼前，形成 MR 應用。

3. 大螢幕、影音與 PCVR

可開啟 2D 視窗並調整位置，也能透過 USB-C Link 或 Air Link 連接 Windows PC。

4. Unity / Android XR 開發

Quest App 通常以 Unity 製作並輸出 Android APK，可用 ADB 或 Meta 工具部署到頭戴裝置。

使用限制：Quest 3 是行動裝置，電池、散熱、運算量與權限有限；重型 AI 或高畫質渲染常需要 PC 協同。

資料來源：Meta Quest 設定文件、Meta Quest Help Center（手部追蹤、2D 視窗、Link / Air Link）

SECTION

資料前處理與固定參數

Segmentation 與 Classification 的資料處理流程與訓練設定

Segmentation

Classification

Fixed Settings

Segmentation 前處理整理

744
train images / labels

187
valid images / labels

2 類
hand_tiles / table_tiles

7:2:1
train / valid / test

資料與標註輸入

- data.yaml 指向 train/images 與 valid/images
- YOLO segmentation normalized polygon：class_id + 多邊形點座標

本專案的前處理重點

每次實驗依 imgsz 進行影像尺寸調整與 letterbox

固定影像前處理流程

尺寸與比例

imgsz 依實驗固定，例如 416、640、832、960、1280。YOLO 會在 dataloader 內 resize / letterbox 到指定尺寸。

Letterbox 設定

rect=False，因此訓練使用 square letterbox。原圖比例透過 padding 保留，不是直接拉伸成正方形。

Mask 產生

label 以 normalized polygon 讀取；訓練時使用 mask_ratio=4 與 overlap_mask=True 產生 mask target。

訓練時資料增強設定

沿用 Ultralytics args.yaml 記錄的固定增強超參數；不同 run 未額外手動調整強度

主要啟用

HSV 顏色擾動
hsv_h=0.015
hsv_s=0.7
hsv_v=0.4

平移與縮放
translate=0.1
scale=0.5

水平翻轉與 mosaic
fliplr=0.5
mosaic=1.0

未啟用 / 為 0

幾何變形未啟用
degrees=0.0
shear=0.0
perspective=0.0

上下翻轉未啟用
flipud=0.0

混合式增強未啟用
mixup=0.0
cutmix=0.0
copy_paste=0.0

close_mosaic 依實驗略有不同
多數為 10；P2 640 為 15

auto_augment / erasing
args 記錄為 randaugment / 0.4

Classification 資料前處理整理

35 類
固定標籤順序

10,440
v5 raw images

7 : 2 : 1
train / val / test

8 角度
固定旋轉增強

128×128
模型輸入尺寸

Raw 資料
zip / v4 raw / 新資料
→
合併原圖
不疊加已增強圖
→
分組切分
原圖單位 7 : 2 : 1
→
RGB / EXIF
方向與色彩統一
→
黑底處理
透明區域補黑
→
旋轉增強
0° 到 315°
→
YOLO cls
train/val/<label>

核心原則

- 先合併 raw 原始圖，再做 train / val 切分，避免重複增強與資料洩漏。
- 使用檔名前綴分組切分，降低同來源 crop 同時進入 train / val 的風險。
- 資料夾分類順序必須與 35 類 label 檔一致，避免模型輸出 index 對錯牌。

影像處理細節

- alpha / 透明區域先合成到黑色背景，對應 segmentation crop 空洞。
- 旋轉使用 bicubic 且 expand=True；新增邊角一律填黑。
- 補成正方形後用 LANCZOS resize 到 128×128。

一句話總結：classification 的前處理目標是把每張牌 crop 統一成「黑底、正方形、128×128、RGB」的 YOLO classification 輸入。

固定訓練參數與部署一致性

v5 最終長訓練固定設定

項目	設定
model	YOLOv8s-cls
dataset	dataset_cls_blackpad_8rot_v5
image size	128
epochs / batch	100 / 256
optimizer	SGD
lr0 / lrf	0.001 / 0.01
weight decay	0.0005
warmup / patience	0 / 0
init checkpoint	v4 YOLOv8s-cls best
train device	CUDA

部署端必須保持一致

輸入格式
單張牌 crop
RGB input
黑色背景 / 遮罩空洞
正方形 padding
resize 到 128×128

推論權重
best.pt：PyTorch / Ultralytics
best.onnx：部署測試
CPU 端分析使用
ONNX Runtime
CPUExecutionProvider

為什麼這些設定要固定？

- classification 接在 segmentation 後面，輸入分布高度依賴 crop 背景、遮罩空洞與 padding 方式。
- 訓練端與部署端若 resize / padding / RGB 不一致，模型看到的牌面比例與背景特徵會改變。
- 固定 label order 可避免模型輸出的 class index 對到錯誤牌種。

SECTION

Segmentation Model Training

模型選擇、指標、P2 Head 與 Knowledge Distillation

mAP50-95

Latency

P2 / KD

Segmentation 指標說明：為什麼選 mAP50-95 與 Latency？

核心評估邏輯：辨識要準，也要能在 VR/MR 系統中即時回饋。

指標代表什麼？

- Mask mAP50-95 ↑：mask 在多個 IoU 閾值下的整體準確度
- Latency ↓ / FPS ↑：推論速度與即時性
- Params / Image Size：模型規模與輸入解析度，屬於影響因素

為什麼選這兩個當代表？

- mAP50-95 代表「辨識品質」：影響後續 crop 與分類
- Latency 代表「使用體感」：延遲過高會破壞即時輔助
- 兩者剛好呈現 Accuracy–Latency trade-off

表格對應：Accuracy = Mask mAP50-95；Real-time = Quest Local / PC Offload Latency / FPS

Route	Segmentation Setting	Image Size	Params	Mask mAP50-95 ↑	Quest Local Latency ↓	PC Offload Latency ↓
1	YOLOv8n-seg	96×96	3.26M	0.076	91.2 ms / 11.0 FPS	94.2 ms / 10.6 FPS
2	YOLOv8n-seg	320×320	3.26M	0.135	125.0 ms / 8.0 FPS	147.0 ms / 6.8 FPS
3	**YOLOv8n-seg**	**640×640**	**3.26M**	**0.411**	**281.7 ms / 3.5 FPS**	**243.6 ms / 4.1 FPS**
4	YOLOv8s-seg	640×640	11.79M	0.417	521.4 ms / 1.9 FPS	511.7 ms / 2.0 FPS
5	YOLOv8m-seg	640×640	27.24M	0.422	1037.4 ms / 1.0 FPS	964.2 ms / 1.0 FPS

Segmentation 模型比較說明

不同分割模型的類型、規模與推論效率比較

什麼是 segmentation 模型？

Instance Segmentation：找出每個物件，並輸出各自的輪廓mask。
Semantic Segmentation：將每個像素分到某一類，但不區分同類中的個別物件。
Promptable Segmentation：需依提示或互動進行分割，泛化強，但即時部署通常較重。

模型	分割類型	參數量	測試條件 / 備註
YOLOv8n-seg	Instance Segmentation	3.4M	A100 TensorRT / CPU ONNX，640px，COCO；非常輕量，適合即時應用
YOLOv8s-seg	Instance Segmentation	11.8M	A100 TensorRT / CPU ONNX，精度比 n 高但較重
YOLOv8m-seg	Instance Segmentation	27.3M	A100 TensorRT / CPU ONNX，偏中大型
YOLO11n-seg	Instance Segmentation	2.9M	T4 TensorRT / CPU ONNX，新一代 YOLO nano seg
FastSAM-s	Promptable / Instance	11.8M	Apple M4 Air CPU；比 SAM 類模型輕很多
MobileSAM	Promptable Segmentation	10.1M	GPU 約略值；CPU 上仍偏慢
SAM-B	Promptable Segmentation	93.7M	泛化能力強，但不適合一般 CPU 即時推論
Mask R-CNN R50-FPN	Instance Segmentation	44.4M	Detectron2 benchmark；經典模型，但即時性不如 YOLO
SegFormer-B0	Semantic Segmentation	3.8M	50.5 FPS 換算，ADE20K；輕量語意分割
DeepLabV3+ MobileNetV2	Semantic Segmentation	15.4M	43.1 FPS 換算，ADE20K；經典語意分割 baseline

重點解讀

YOLO 系列最適合即時應用：速度快、參數量小，且可直接做instance segmentation。
SAM / MobileSAM / FastSAM 強調泛化與互動式分割，但在一般CPU 上延遲較高。
Mask R-CNN 屬於經典 instance segmentation 模型，但即時性通常不如YOLO。
SegFormer 與 DeepLabV3+ 偏向 semantic segmentation，較適合像素級場景理解，不一定適合逐張麻將牌分離。

P2 Head：針對小而密集的麻將牌提升 Segmentation

Baseline YOLOv8n-seg

原本只使用三尺度預測

P3/8 small　P4/16 medium　P5/32 large

小物件邊界與位置細節較容易被壓縮

加入 P2/4 →

P2 YOLOv8n-seg

四尺度 Segment(P2, P3, P4, P5)

P2/4 x-small　P3/8 small　P4/16 medium　P5/32 large

P2/4 保留更多邊界、位置與小牌細節

架構與驗證需要同步修正

1. **Head fusion 擴充**
   從 P5 往上融合 P4、P3，再上採樣 concat backbone P2，形成 P2/4 x-small feature。
2. **PAN path 回接四尺度**
   再由 P2 往下採樣回 P3/P4/P5，最後以 Segment([P2,P3,P4,P5]) 輸出。
3. **Validator 改成 proto-stride aware**
   P2 版本的 prototype 實際為 image/2，因此 validation 要使用 proto_stride=2。

代價與限制

P2 對小物件有利，但會明顯增加記憶體與計算量。

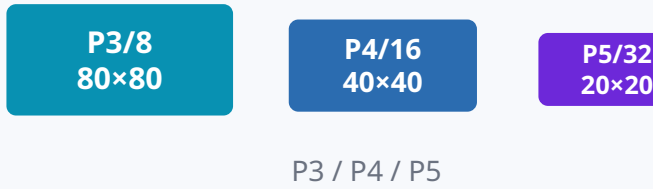
GFLOPs(Giga Floating-Point Operations)：約 11.3 → 23.5

若不修正 proto_stride，predicted mask 與 GT mask 尺寸可能不一致，造成validation 指標失真或錯誤。

P2 Head：讓 YOLO 多看一層高解析度特徵圖

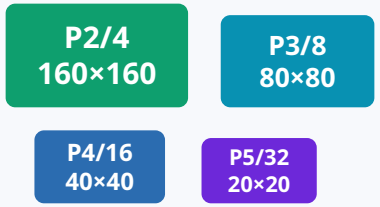
核心概念：三尺度 → 四尺度

Baseline YOLOv8n-seg



P3 / P4 / P5

P2 YOLOv8n-seg



Segment(P2, P3, P4, P5)

P2/4 保留更多邊界、位置與細節，對「小物件＋密集排列」的麻將牌更有利。

為什麼 P2 Head 吃算力？

1. P2 feature map 面積大

以 640×640 輸入為例，P2 是 160×160，共 25,600 個位置，是 P3 的 4 倍。

2. 多做 feature fusion 與 prediction

新增 P2 分支後，需要上採樣、concat、卷積融合，並在四個尺度輸出 segmentation。

3. Mask / prototype 對齊更重

P2 segmentation 會改變 prototype 的有效 stride，validation 需用 proto_stride=2 對齊 mask。

GFLOPs	YOLOv8n-seg	→	YOLOv8n-seg + P2
	11.3	→	23.5

卷積計算量約與 $H \times W \times Cin \times Cout \times k^2$ 成正比；P2 的 H×W 最大，所以特別耗算力。

結論：P2 Head 用更高解析度換取小物件 segmentation 表現；適合麻將牌，但會讓 GFLOPs、VRAM、訓練與推論負擔上升。

Source: p2_head_kd_technical_details_20260611.pdf

Knowledge Distillation：用 Teacher 的 soft targets 輔助 Student

Teacher
大型 Seg Model
(eval / frozen)

→

Soft Targets
logits / DFL / mask / proto

→

Student
YOLOv8n-seg / P2 student

→

Total Loss
Supervised + KD

KD Loss 設計

L_total = L_supervised + weighted distillation terms

蒸餾項	對齊方式	權重
Class logits	BCE with logits	0.25
DFL distribution	KL divergence	0.25
Mask coefficients	MSE	0.05
Mask prototypes	Resize + MSE	0.05

Temperature T=4.0；class logits 與 DFL loss 乘以 T^2 補償梯度尺度。

P2 + KD 的尺度對齊策略

Teacher：P3 / P4 / P5

共同尺度做 KD

↓

Student：P2 / P3 / P4 / P5

新增 P2 branch 只用 GT supervised loss 監督

重點結論：P2 是針對小物件 segmentation 的主要改動；KD 作為輔助監督，可補充 soft target 與分布資訊。

SECTION

Classification Model Evaluation

牌種分類驗證結果、弱項分析與補資料方向

Top-1 Accuracy

Confusion Pairs

Data Strategy

Classification Validation：最穩定的牌

這些牌在 validation 中「真實為該牌」時完全沒有被判成其他牌；但 Precision 可看出是否會吸到其他類別。

97.88%
整體 Top-1 Accuracy

8,712
validation augmented images

11 類
Recall 達到 100%

0
該類錯誤數

注意
Precision 差異

高穩定類別整理

牌	Recall	錯誤數	Precision	重點
1p	100.00%	0	100.00%	完全穩定
1s	100.00%	0	100.00%	完全穩定
4m	100.00%	0	100.00%	完全穩定
5s	100.00%	0	100.00%	完全穩定
green	100.00%	0	99.70%	幾乎穩定
2s	100.00%	0	99.63%	幾乎穩定
7s	100.00%	0	99.52%	幾乎穩定
6s	100.00%	0	98.92%	少量被吸入
5m	100.00%	0	98.25%	少量被吸入
flower	100.00%	0	96.64%	易吸其他牌
south	100.00%	0	94.57%	易吸其他牌

Recall = 真實為該牌時，有多少比例被正確判回該牌。

Precision = 被模型判成該牌的結果中，有多少真的屬於該牌。

關鍵解讀：flower 與 south 的 Recall 都是 100%，代表它們自己不會被判錯；但 Precision 較低，表示其他牌會被誤判成 flower / south。

這些穩定牌可作為目前模型的可靠基準；下一輪應把資料量集中補在弱項與混淆配對。

Classification Validation：弱項與補資料方向

弱點主要集中在相似圖樣、跨花色混淆，以及被flower / south 等類別吸收的錯分。

優先補強：2m / 2p / 6m / 4s / 9s / 8m

固定混淆：9s↔9p、4s→8p、6m→9m

資料策略：補小 crop、低解析、破碎邊界與旋轉樣本

最不準的牌

牌	Recall	錯誤數	主要誤判
2m	**90.09%**	**23**	1m:9, 8m:8, flower:4
2p	**90.83%**	**22**	4p:8, west:5, 4s:3
6m	**91.83%**	**17**	9m:12
4s	**93.75%**	**16**	8p:16
9s	**93.75%**	**17**	9p:16
8m	**94.83%**	12	6m:9, 3m:3
6p	95.31%	6	flower:5
3s	95.98%	9	3p:8
west	97.12%	9	south:6, white:3

主要固定混淆配對

錯分方向	次數
9s → 9p	**16 次**
4s → 8p	**16 次**
6m → 9m	**12 次**
8m → 6m	**9 次**
2m → 1m	**9 次**
2m → 8m	**8 次**
3s → 3p	**8 次**
2p → 4p	**8 次**

下一輪補資料重點

優先補弱項牌

- 2m、2p、6m
- 4s、9s、8m

針對混淆成對補圖

- 9s/9p、4s/8p
- 6m/9m、2m/1m/8m

結論：最新模型整體已穩定，但要再提升實機表現，應優先補「小 crop、低解析、邊界破碎、旋轉後仍清楚」的弱項樣本，並檢 查 flower / south 是否有吸收其他類別的趨勢。

SECTION

PPO Agent Evaluation

參考 16mj 建立決策環境, 並比較 PPO 與 MaskablePPO

16mj

MaskablePPO

25,000 Games

參考 16mj：從可玩遊戲到可訓練 Mahjong RL 環境

參考 simonchih/16mj 的台灣 16 張麻將遊戲流程，延伸成 PPO / MaskablePPO 評估平台

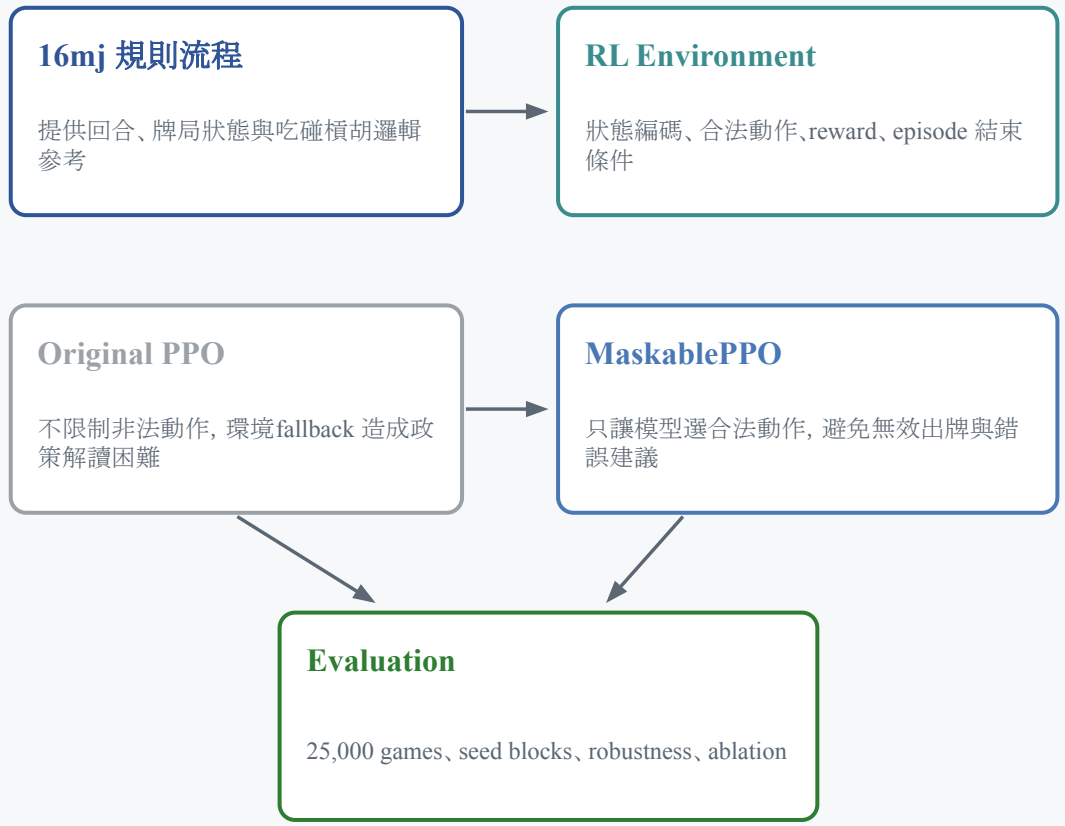
Reference Game

16mj 提供的基礎

Python + Pygame 的 16 張麻將遊戲專案
具備台灣麻將核心流程：摸牌、出牌、吃、碰、槓、胡
可作為規則與遊戲狀態處理的參考基礎
本研究重點不是重做遊戲，而是把流程改造成 AI 可學習的環境

轉換重點：把「玩家可按的按鈕」轉成 Agent 可選的 action，並加入 state、reward、legal action mask。

Research Pipeline



Source: simonchih/16mj GitHub；本專案 PPO 評估摘要

主實驗: MaskablePPO 提升勝率

25,000 games per model；matched random seeds；Original PPO vs MaskablePPO 80M

Win rate

38.33% → 42.12%

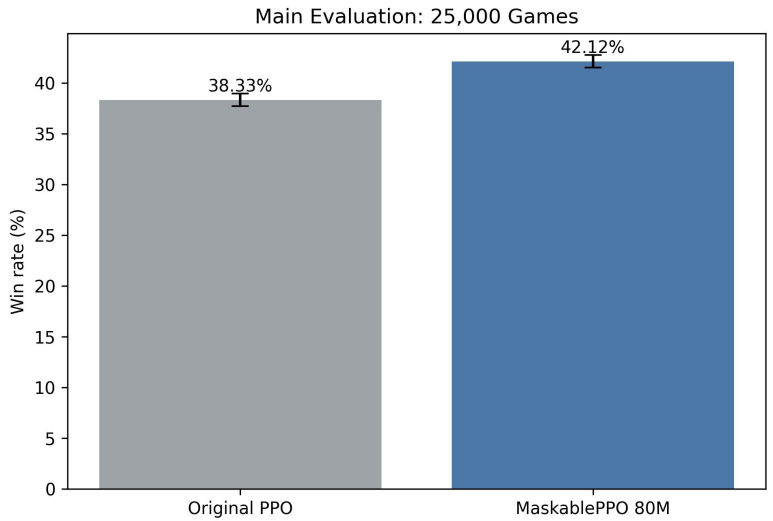
+3.792 percentage points；bootstrap 95% CI: +2.976% 到 +4.620%

Average reward

5.13 → 7.95

勝率提高的同時，平均 reward 也明顯上升

Win Rate Comparison

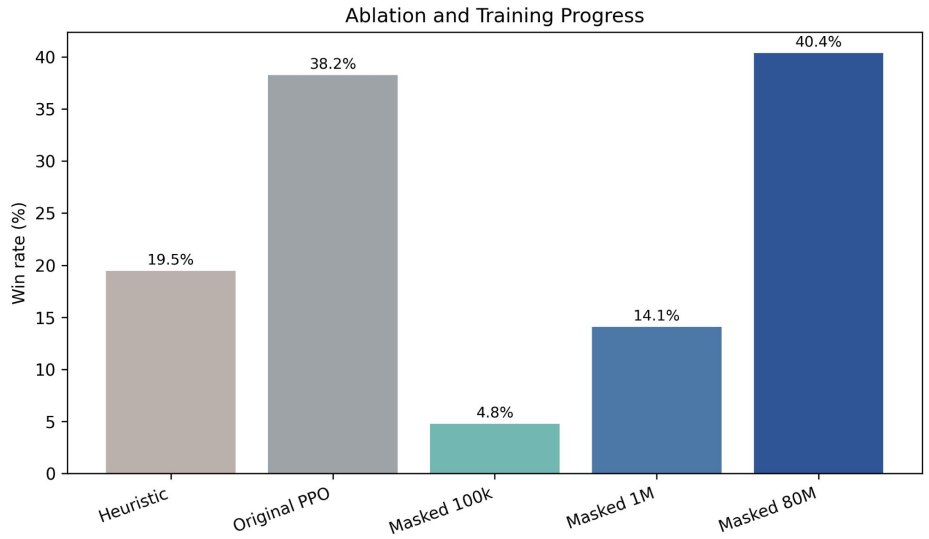


Source: main_25k_summary.csv；thesis_results_summary.md

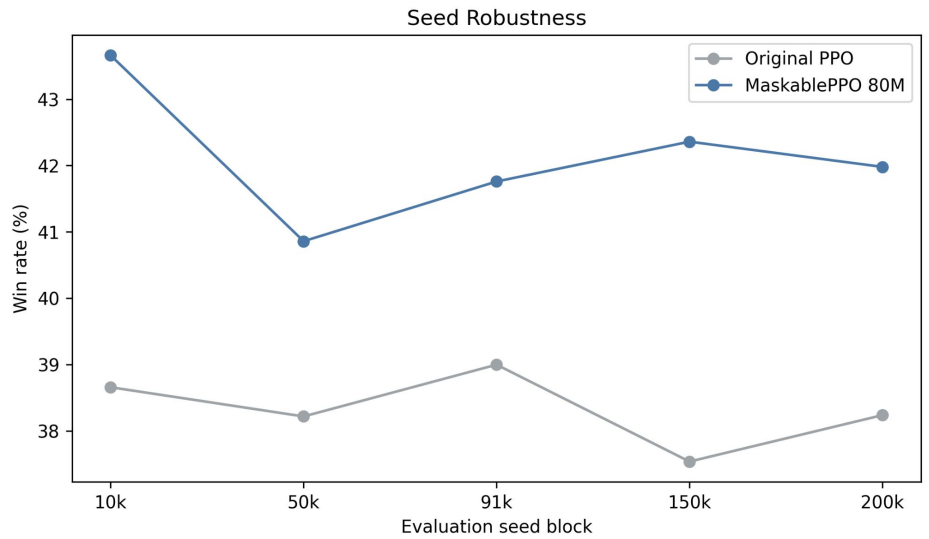
消融與穩健性：Masking 要搭配足夠訓練才有效

Ablation 說明 action mask 的作用；seed robustness 確認結果不是單一隨機種子偶然

Ablation: training progress



Seed robustness



結論重點

Action masking 可把非法動作率降到0%，但 100k / 1M 訓練量仍不足；80M 後才超越 Original PPO，且五個 seed block 全部勝出。

Source: condition_summary.csv；thesis_results_summary.md

SECTION

延遲測試補充

Quest Local 與 PC Offload 的分段延遲與部署判斷

Quest Local

PC Offload

Pipeline Latency

ADB 連線較快，但不是本次延遲測試的主要 Bottleneck

1. App-layer 串流：USB 與 Wi-Fi ADB 行為幾乎相同

Transport	收 frame	期間	FPS	平均 frame
USB adb reverse	69	68.49 s	0.993	62.20 KB
Wi-Fi ADB + reverse	69	68.58 s	0.992	62.28 KB
差異解讀	相同	+0.09 s	≈ 0	≈ 0.08 KB

兩種連線都能接到 127.0.0.1:5000 並維持約 1 FPS；所以目前 FPS 差異不主要由 USB / Wi-Fi ADB 造成。

2. 真正量到的瓶頸：等待與推論遠大於傳輸寫入

ADB/TCP uplink
0.10–0.19 ms
Quest 寫入 payload

PC Server total
23–849 ms
decode + inference

Wait / response
93.8–955.7 ms
含 PC 推論等待

判斷依據：uplink 僅佔 PC offload E2E 的 < 0.12%；相較之下，segmentation 單段可達 6.7–825.0 ms。

3. 以 640×640 v8n PC offload 為例：ADB 幾乎不是肉眼可見的長條

R3 / set1_v8n_seg_img640：PC offload E2E = 243.6 ms（Avg）

最大主要段落

結論
即使 USB ADB reverse 比 Wi-Fi ADB 更適合當穩定低延遲 baseline，優先優化方向仍應放在 segmentation 推論、PC backend、response loop 與整體 pipeline 排程，而不是先換連線方式。

Takeaway：ADB 連線方式會影響穩定性與展示便利性，但在本次數據中，端到端延遲的顯著瓶頸是「模型推論 + 等待 PC response」。

source: latency_test_process_2026-06-09.md, usb_reverse / wifi_adb_reverse logs

Quest / PC Offload 延遲測試報告

測試目的

- 比較 Quest local 與 PC offload 的端到端延遲。
- 每條 route 都包含 segmentation、classification_yolos 與 PPO。
- 5 個 segmentation route × 2 種部署方式，共 10 組測試。

實驗矩陣

Route	Seg	Input	Params	JPEG
R1 / idx 0	v8n 96	96×96	3.26M	1,086 B
R2 / idx 1	v8n 320	320×320	3.26M	7,898 B
R3 / idx 2	v8n 640	640×640	3.26M	30,837 B
R4 / idx 3	v8s 640	640×640	11.79M	30,837 B
R5 / idx 4	v8m 640	640×640	27.24M	30,837 B

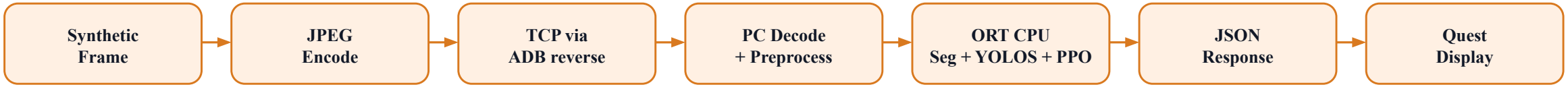
Pipeline A：Quest Local

全部模型在 Quest 3 端以 Unity Sentis GPUCompute 執行；模型輸出需 Readback 後才算可用結果。

固定條件

- classification_yolos 固定
- PPO model 固定
- 每 route 20 次正式量測
- warmup 3 次不納入統計

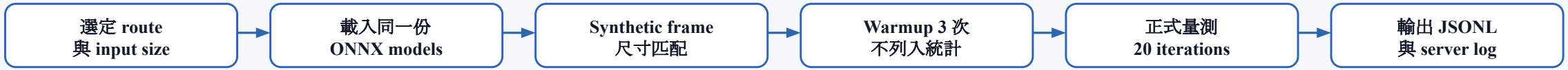
Pipeline B：PC Offload（USB / ADB reverse）



source: latency_test_process_2026-06-09.md

延遲測試流程與方法合理性

共通測試流程



具體量測點

Quest / Unity 端

- 使用 System.Diagnostics.Stopwatch 高解析度 timer。
- captureEncodeMs：Texture2D → JPEG byte array。
- Sentis stage：worker.Schedule + PeekOutput + ReadbackAndClone。
- Readback 後才代表模型輸出已可被 CPU/後續流程使用。

PC Server 端

- 使用 Python time.perf_counter() 計算 stage duration。
- TCP 採 4-byte length-prefix，避免 stream 邊界不明。
- ONNX Runtime 指定 CPUExecutionProvider。
- 分開記錄 decode / segmentation / YOLOS / PPO，另由 Quest 端記錄 E2E。

為什麼可以用這種方法？

- 同一份 ONNX route 與相同 YOLOS / PPO，主要變因只剩部署位置。
- Synthetic frame 固定尺寸與內容，可避免真實 camera frame 抖動干擾，專注測推論與傳輸成本。
- warmup 排除 cold start；20 次 iteration 反映穩態延遲；兩端皆使用高解析度monotonic timer。

source: latency_test_process_2026-06-09.md

Pipeline A：Quest Local 分段延遲

量測方式

- 每個 Sentis 模型 stage 量測「Schedule + output readback」，因此不是只量呼叫排程時間，而是量到可被後續流程使用的輸出時間。
- totalAvgMs = capture/encode + segmentation + YOLOS classification + PPO + display；下表剩餘項為 total - seg - YOLOS - PPO。

分段延遲結果（Avg ms）

Route	Seg model	Input	Total	Seg	YOLOS	PPO	Encode+Display (估)	FPS
R1	v8n 96	96×96	91.2	49.9	40.0	1.0	0.3	10.96
R2	v8n 320	320×320	125.0	84.4	37.5	0.8	2.3	8.00
R3	v8n 640	640×640	281.7	233.2	38.3	0.8	9.4	3.55
R4	v8s 640	640×640	521.4	482.8	30.2	0.8	7.7	1.92
R5	v8m 640	640×640	1037.4	1001.6	27.9	0.6	7.3	0.96

解讀

- Quest local 的主要瓶頸幾乎都在 segmentation：96px 為 49.9 ms，640px v8n 升到 233.2 ms，v8m 640 更達 1001.6 ms。
- YOLOS classification 約 28–40 ms，PPO 約 0.6–1.0 ms；因此優化重點應先放在 segmentation input size / model size / output readback。

source: latency_test_process_2026-06-09.md

總結：部署方式的分界點

端到端延遲比較（Avg ms）

Route	Seg	Input	Quest E2E	PC E2E	較快	差距
R1	v8n 96	96×96	91.2	94.2	Quest local	3.0
R2	v8n 320	320×320	125.0	147.0	Quest local	21.9
R3	v8n 640	640×640	281.7	243.6	PC offload	38.1
R4	v8s 640	640×640	521.4	511.7	PC offload	9.6
R5	v8m 640	640×640	1037.4	964.2	PC offload	73.2

讀法：PC server 本身通常更快，但小圖時　offload 固定成本會抵銷優勢。

- R1/R2：Quest local 較快，因為模型計算量小，傳輸與等待成本相對明顯。
- R3 起：640px segmentation 成本變大，PC offload 開始能追回並超過Quest local。

部署判斷地圖

小圖 / 輕量模型：優先　Quest local，少掉 offload 固定成本。

640px / 較大模型：PC offload 開始有利，但仍受傳輸等待影響。

結論

- 96 / 320 小圖：PC server 雖快，但 offload 固定成本抵銷優勢。
- 640 v8n 起：segmentation 計算量提高，PC offload 的 compute 優勢開始超過傳輸成本。
- v8m 640：兩者都接近 1 FPS，代表大模型不適合直接做即時　VR 互動。

source: latency_test_process_2026-06-09.md