



On-Device Deployment of Optuna-Optimized YOLOv8n via Flutter for Real-Time and Explainable Cocoa Pod Disease Detection

I Putu Oka Wisnawa¹ , I Made Dwi Jendra Sulastra² ,
Putu Manik Prihatini¹ , I Made Agus Oka Gunawan¹ ,
Ni Luh Putu Listya Dewi¹, and Ni Wayan Widayarsi Damayanti¹

¹ Information Technology Department, Politeknik Negeri Bali, Bali, Indonesia

² Accounting Department, Politeknik Negeri Bali, Bali, Indonesia
okawisnawa@pnb.ac.id

Abstract. Deep learning-based cocoa pod disease detection models achieve high accuracy on GPU-accelerated infrastructure, yet are rarely validated for on-device deployment in the bandwidth-constrained smallholder plantations where they are most needed. This study deploys a previously published Optuna Bayesian-TPE-optimized YOLOv8n checkpoint, selected for zero-false-negative Macro Recall and lower inference latency over a marginally higher-mAP Hyperband alternative, into a Clean-Architecture Flutter application via TensorFlow Lite, providing the first paired, on-device validation of a model previously evaluated only in its pre-conversion form. On an identical 30-image held-out test set, TFLite conversion and mobile-pipeline integration reduce mAP@0.5 from 0.984 to 0.908 (-7.6 percentage points), with degradation concentrated in the Black Pod Rot class (-18.4 pp) and correlated with close-range, large-bounding-box framing in the affected images. On-device inference on a mid-range Android device averages 4,991.5 ms/frame (≈ 0.20 FPS, $\approx 657\times$ slower than training-time GPU latency), falling well below conventional real-time thresholds, a gap we quantify as the dominant deployment cost rather than obscure. A fully randomized, replicated 8-condition stability protocol found no evidence of memory leaks (one-way ANOVA, $F(2,34)=0.104$, $p=0.90$) and no OS-reported thermal throttling across sessions up to 90 minutes. The application further integrates a fully offline, farmer-facing disease-guidance module surfacing etiology, prevention, and treatment information directly from detection results, adapted from a published phytopathological reference, though its in-application adaptation remains pending independent domain-expert re-validation; this integration is demonstrated architecturally but not empirically validated with end-users. These results characterize concrete, previously unmeasured costs of translating an optimized detection model into field-deployable mobile software.

Keywords: Cocoa Disease Detection, Edge AI, Mobile Deployment, TensorFlow Lite, YOLOv8

1 Introduction

Precision agriculture, data-driven farm management for efficiency and productivity, has pushed deep learning and computer vision to the center of plant-disease surveillance, offering speed and consistency that subjective, labor-intensive visual inspection cannot match (Sajitha et al., 2024; Upadhyay et al., 2025; Bhargava et al., 2024; Gui et al., 2021; Shoaib et al., 2025). The need is acute for cocoa (*Theobroma cacao* L.), a globally significant smallholder commodity where fungal pathogens, chiefly Anthracnose (*Colletotrichum* spp.) and Black Pod Rot (*Phytophthora* spp.), inflict substantial economic losses (Alvarado, Restrepo-Arias, Velásquez, & Maiza, 2025; Alvarado, Restrepo-Arias, Velásquez, Martinez-Vargas, et al., 2025; Berckemas et al., 2025; Dinas Pertanian, Ketahanan Pangan Dan Perikanan, 2025; Garma, 2025); automated identification measurably improves field intervention over unaided scouting (Pagaduan et al., 2026).

Recent work converges on modern object-detection architectures for this task: Faster R-CNN, Mask R-CNN, and lightweight detectors such as SSD MobileNetV2 FPN-Lite have targeted cocoa pod pathology and ripening-stage recognition specifically (Garma, 2025; Restrepo-Arias et al., 2025), alongside broader anomaly-detection-based crop monitoring (Gkoutakos et al., 2024), compact backbones that pursue efficiency without severe accuracy loss (Aji et al., 2025; Bhuiyan et al., 2023; Duhan et al., 2025), and systematic augmentation that improves generalization under field-representative variability (Antwi et al., 2024). Building on this trajectory, our own prior work established a Bayesian Tree-structured Parzen Estimator (TPE)-optimized YOLOv8n framework for simultaneous Anthracnose, Black Pod Rot, and Healthy Cocoa Pod detection on a Bali-sourced dataset, reporting $\text{mAP}@0.5 = 0.984$ and Macro Recall = 1.000 on an independent held-out test set (Wisnawa et al., 2026). A smaller body of work extends beyond bare classification toward integrated disease-management guidance for end-users (Pagaduan et al., 2026), reflecting growing recognition that a diagnostic label alone has limited standalone utility for a non-specialist farmer.

Two gaps separate this work from field-deployable, farmer-facing utility. First, high-performing models are typically validated on GPU-accelerated infrastructure with stable connectivity (Bhuiyan et al., 2023; Chung et al., 2026; Janarthan et al., 2022; Rashwan & Elteir, 2022), whereas smallholder cocoa plantations, including the Jembrana Regency site underlying (Wisnawa et al., 2026), mainly operate in bandwidth-constrained regions on mid-range or low-end hardware (Bahaa et al., 2026; Kumi et al., 2022). TensorFlow Lite conversion enables fully offline inference (Duhan et al., 2025; Goklani, 2024; Kumi et al., 2022; Rashwan & Elteir, 2022), yet the literature rarely reports paired pre/post-conversion accuracy on an identical test set, or characterizes on-device latency, memory, and thermal behavior under replicated, randomized conditions robust enough for stability claims. Second, even successful mobile deployments typically stop at classification output: disease-specific etiology, prevention, and treatment guidance, what a farmer needs to act on a diagnosis, is rarely integrated into the workflow, and where attempted (Pagaduan et al., 2026), the provenance and validation status of the advisory content is seldom made explicit.

This study addresses both gaps by deploying the Bayesian-TPE-optimized YOLOv8n checkpoint from (Wisnawa et al., 2026), chosen over a marginally higher-mAP Hyperband alternative for its zero-false-negative Macro Recall and lower latency, into a cross-platform Flutter application via TensorFlow Lite, integrated with an in-app module surfacing etiology, prevention, and treatment guidance immediately upon detection. We use explainability here in this actionable, farmer-facing sense, distinct from model-interpretability techniques (e.g., Grad-CAM), which are outside the present scope. The study examines: (a) how TFLite conversion and mobile-pipeline integration affect detection accuracy relative to the original checkpoint on an identical test set; (b) on-device performance, latency, throughput, memory stability, thermal behavior, under sustained, randomized, replicated execution against real-time thresholds; and (c) whether disease-specific guidance can be integrated into the detection workflow as a first-class feature, with the content-governance considerations this raises. The contribution is threefold: (i) a direct, paired accuracy comparison isolating the cost of TFLite conversion, not previously reported for this framework; (ii) a randomized, replicated benchmarking protocol for on-device latency, memory, and thermal telemetry, validated via one-way ANOVA, addressing the single-session-benchmark limitation common in prior mobile plant-disease studies (Bahaa et al., 2026; Janarthan et al., 2022; Rashwan & Elteir, 2022); and (iii) an in-workflow disease-guidance module adapted from a published phytopathological reference (Soesanto, 2024), with its adaptation flagged pending independent domain-expert review prior to field-authoritative use, a transparency practice we argue should be standard for this application class.

2 Methodology

2.1 Dataset and Preprocessing

The dataset comprises 300 raw RGB images of *Theobroma cacao* pods acquired in-field at Kecamatan Melaya, Jembrana Regency, Bali, Indonesia, spanning three mutually exclusive classes: Anthracnose (*Colletotrichum gloeosporioides*), Black Pod Rot (*Phytophthora palmivora*), and Healthy Cocoa Pod. Bounding boxes were annotated manually on Roboflow, without automated smart-select assistance, to preserve pixel-level fidelity around naturally occluded lesions. The corpus was split 80:10:10 (240/30/30 images) into training/validation/test subsets; validation and test were held at 1× (unaugmented) to prevent data leakage, while training was expanded 3× through combined spatial (flip, static 90° rotation, 0–15% crop, −15° to +15° dynamic rotation, ±5° shear) and spectral (±5° hue, ±15% saturation, ±20% brightness, ±15% exposure, ≤1 px blur, ≤0.1% noise) augmentation, yielding 780 total images (720 effective training images) (Wisnawa et al., 2026). All images were auto-oriented and resized to 640×640 px for the YOLOv8n input specification. This study reuses that dataset and split verbatim; critically, the 30-image test subset is identical across the original training-time evaluation and the on-device evaluation in Section 3, enabling a

direct, paired comparison between pre- and post-deployment accuracy not previously established.

2.2 Hyperparameter Tuning via Optuna

The detection backbone is YOLOv8n (yolov8n.pt, approximately 3.2M parameters), selected for its anchor-free head and favorable accuracy-to-compute ratio on mobile-class hardware. Hyper-parameter optimization was performed with Optuna's Tree-structured Parzen Estimator (TPE) sampler over a ten-dimensional dynamic search space spanning gradient parameters (lr0: $1e^{-4}$ to $1e^{-2}$, log-scale; lr1: 0.01-0.2; momentum: 0.6-0.98; weight_decay: $1e^{-4}$ to $1e^{-2}$, log-scale) and augmentation parameters (mosaic: 0.5-1.0; mixup: 0.0-0.5; degrees: 0-45°; hsv_h: 0.0-0.1; hsv_s: 0.0-0.9; hsv_v: 0.0-0.9), across 100 trials of ≤ 50 epochs each, under a fixed seed (42) with deterministic=True for reproducibility (Wisnawa et al., 2026). Two pruning strategies were previously compared, NopPruner (exhaustive Bayesian TPE) and HyperbandPruner (adaptive early-stopping via Successive Halving), with the Bayesian-TPE configuration converging at trial 21 (validation mAP@0.5:0.95 = 0.9012) and the Hyperband configuration at trial 23 (0.9008) at 44.4% lower cumulative GPU-epoch cost.

This work deploys the Bayesian-TPE (NopPruner) checkpoint specifically, not the marginally higher-mAP Hyperband checkpoint. The choice is deliberate: on the held-out test set, the Bayesian configuration attains Macro Recall = 1.0000 (zero false negatives across both disease classes) versus 0.9985 for Hyperband, and a training-time inference latency of 7.60 ms/image versus 11.64 ms/image, a 34.7% reduction. For a fungal disease-screening application in which a missed detection permits uncontrolled pathogen spread, and for deployment on latency-sensitive edge hardware, these two properties are prioritized over the 0.0073 mAP@0.5 deficit relative to Hyperband, which is not statistically distinguishable on a 30-image test set.

2.3 On-Device Flutter Deployment Pipeline

The selected PyTorch checkpoint (best.pt) was exported to TensorFlow Lite (float32, non-quantized). Rather than hardcoding the tensor layout from generic YOLOv8 documentation, the application performs runtime shape introspection at model-load time via tflite_flutter's interpreter API and dynamically resolves the input layout, a step that proved necessary, since the exported model's input tensor is [1, 3, 640, 640] (channel-first, NCHW), contradicting the NHWC [1, 640, 640, 3] layout commonly documented for YOLO-family TFLite exports. The output tensor is [1, 7, 8400]: the second axis encodes four box-geometry channels (x, y, w, h, center-based, normalized to [0,1]) followed by three raw (sigmoid) per-class scores, YOLOv8's decoupled head omits the separate objectness channel present in YOLOv5/v7, and the third axis (8,400) aggregates anchor-free candidate points across three feature-pyramid strides (P3/stride-8: $80 \times 80 = 6,400$; P4/stride-16: $40 \times 40 = 1,600$; P5/stride-32: $20 \times 20 = 400$).

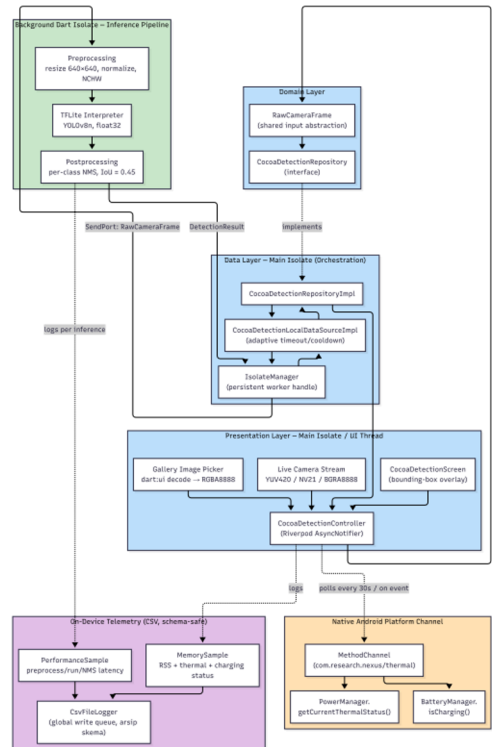


Figure 1. System Architecture of the On-Device Detection Pipeline

The client is implemented in Flutter using a feature-first Clean Architecture (domain/data/presentation layers) with Riverpod for reactive state management. Inference runs inside a persistent background Dart Isolate, decoupling the ≥ 1 s-order inference latency (Section 3) from the UI thread and preventing frame-render jank. Preprocessing (bilinear resize to 640×640 , $[0,1]$ normalization, RGB channel-planar reordering into NCHW) and post-processing (per-class greedy Non-Maximum Suppression, IoU threshold = 0.45, confidence threshold = 0.5) are implemented natively in Dart within the isolate, avoiding cross-isolate serialization overhead for raw pixel buffers. Two independent acquisition paths converge on a common RawCameraFrame abstraction feeding an identical preprocessing/inference/NMS pipeline: (i) a live camera stream (YUV420/NV21/BGRA8888 plane sampling, with a frame-dropping strategy under isolate back-pressure) and (ii) static gallery-image upload (decoded via dart:ui, RGBA8888), included as an accessibility fallback for low-end devices or thermally throttled sessions where sustained live-streaming inference is impractical.

To operationalize explainability in the farmer-facing sense described above, the application integrates an in-workflow disease-information module accessible directly from the detection result screen. On user-initiated request following any classification (live-scan capture or gallery upload), the application resolves a DiseaseInformation record keyed by the predicted class label through a dedicated repository/data-source

pair, structurally isolated from the detection pipeline. Content, etiological summary, causative agent, and enumerated prevention and treatment steps per class, is bundled as a static JSON asset loaded from the application's asset bundle rather than fetched over a network connection, preserving the fully offline operating model established for the detection pipeline. Advisory content was adapted from a published phytopathological reference on cocoa disease management (Soesanto, 2024); the bundled content nonetheless carries an explicit, user-visible disclaimer stating that its in-application adaptation, wording, and presentation have not independently undergone dedicated expert review for this deployment context, and is therefore not positioned as an authoritative, field-ready recommendation at this stage.

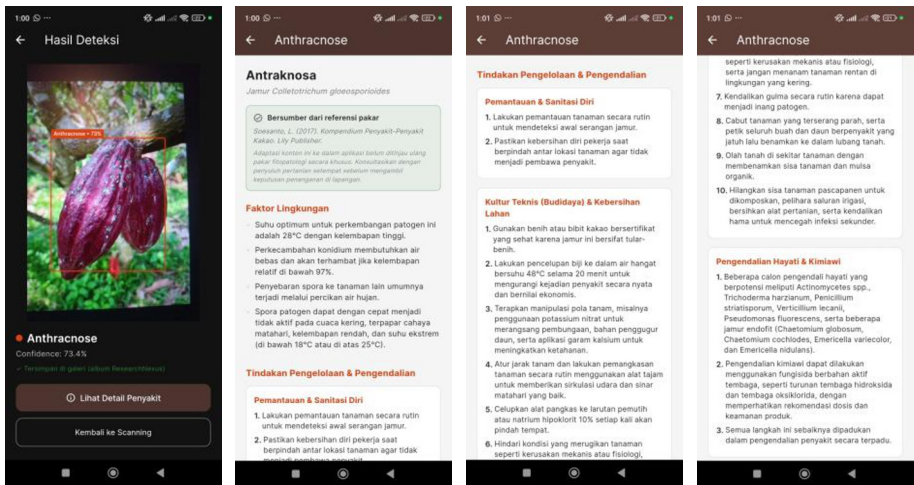


Figure 2. Flutter Application UI Screenshots, Live Scanning View with Bounding-Box Overlay and Confidence Label

Fixed-duration timeouts and inter-frame cooldowns proved empirically insufficient given the high observed variance in on-device inference latency (Section 3). The deployed pipeline instead computes an adaptive per-inference timeout ($2.5\times$ the rolling mean of the five most recent successful inferences, floor 15,000 ms) and an adaptive inter-frame cooldown ($0.3\times$ the same rolling statistic, floor 1,000 ms), rather than relying on static, device-agnostic thresholds. To characterize deployment behavior without external profiling tooling, the application logs, per inference: stage-wise latency (preprocessing/inference/NMS, millisecond resolution) to structured CSV; periodic (30 s) process-level Resident Set Size (RSS) via `dart:io`'s `ProcessInfo`; and OS-reported thermal status via a native Android platform channel to `PowerManager.getCurrentThermalStatus()` (API ≥ 29), yielding a categorical throttling state (NONE/LIGHT/.../SHUTDOWN) rather than a raw temperature reading, which Android does not expose through a standardized public API. Evaluation hardware: POCO M4 Pro (model 2201117PG), MediaTek Helio G96 octa-core SoC (up to 2.05 GHz), 6 GB RAM.

3 Result and Discussion

3.1 Result

The deployed model was evaluated, on-device and post-TFLite-conversion, on the identical 30-image held-out test set used for training-time validation (Section 2.1), via a custom Python evaluation harness that independently re-implements the application’s preprocessing, layout detection, and NMS logic to guarantee metric fidelity to the deployed artifact rather than the pre-conversion checkpoint.

Table 1. Post-Deployment Detection Accuracy

Class	Precision	Recall	F1	AP@0.5	95% Wilson CI (Recall)
Anthracnose	0.800	1.000	0.889	0.958	0.676-1.000
Black Pod Rot	0.875	0.778	0.824	0.778	0.453-0.937
Healthy Cocoa Pod	0.867	1.000	0.929	0.989	0.772-1.000
Macro Average	0.847	0.926	0.881	0.908	0.815-1.000*

* This interval is not a standard Wilson score (as that does not apply to combined averages), it is calculated via bootstrap resampling (10,000 iterations, with each class resampled according to its respective N).

Table 2. On-Device Performance Summary

Metric	Value
Mean Latency (ms)	4,991.5
Median Latency (ms)	4,666.5
σ Latency (ms)	1,097.8
Estimated FPS	0.20
RSS Range (MB)	76.7 - 519.4
Thermal Status Distribution	NONE: 100% (n = 2,489)
CPU Usage	Not measured

Overall mAP@0.5 = 0.908; mAP@0.5:0.95 = 0.806. Given the small per-class N (8/9/13 for Anthracnose/Black Pod Rot/Healthy, respectively), 95% Wilson-score confidence intervals are reported alongside point estimates for Recall; the apparent Recall = 1.000 for Anthracnose and Healthy Cocoa Pod carries a lower CI bound of 0.676 and 0.772, respectively, and should not be read as a zero-miss guarantee at population scale. Exported model size: 12.3 MB. Across cumulative live-scan telemetry (4,264 samples, 10 independent application sessions, 590.3 minutes $\approx$ 9.8 cumulative hours, excluding two pathological stall events detailed in Section 4), per-frame inference latency (preprocessing + TFLite invocation + NMS) exhibited mean = 4,991.5 ms, median = 4,666.5 ms, σ = 1,097.8 ms (CV $\approx$ 22.0%), ranging from 2,819 to 20,971 ms, a sustained throughput of approximately 0.20 FPS. Resident memory (RSS) ranged 76.7–519.4 MB across sessions; CPU utilization was not instrumented, so RSS served as the primary resource-pressure proxy (Section 4). No OS-reported thermal throttling (thermalStatus > NONE) occurred in any session, including a continuous 89.6-minute session (n = 769 inference samples).

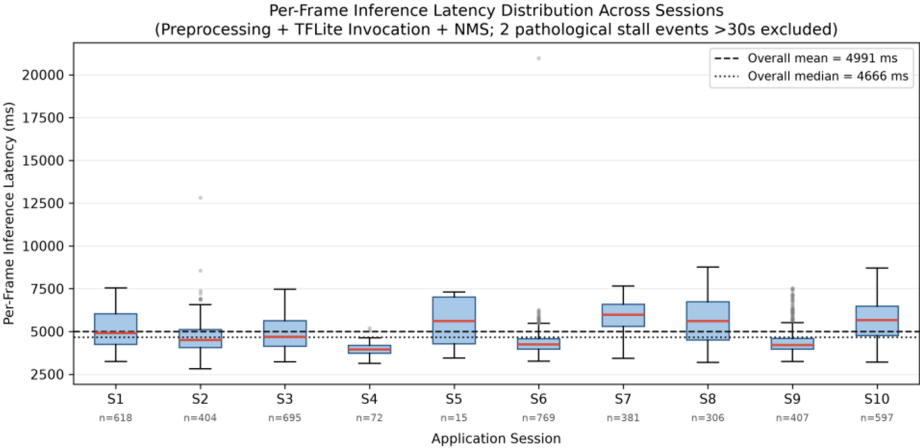


Figure 3. Per-Frame Inference Latency Distribution Across 10 Independent Application Sessions

Table 3. Paired Comparison of Detection Accuracy and Inference Latency Between the Original Bayesian-TPE-Optimized Checkpoint (Wisnawa et al., 2026) and the Same Checkpoint After TFLite Conversion and On-Device Flutter Integration

Metric	Training (PyTorch, GPU)	Deployed (TFLite, on-device)	Δ (Deployed - Training)
AP@0.5 Anthracnose	0.995	0.958	-0.037
AP@0.5 Black Pod Rot	0.962	0.778	-0.184
AP@0.5 Healthy Cocoa Pod	0.995	0.989	-0.006
mAP@0.5 (overall)	0.984	0.908	-0.076
mAP@0.5:0.95 (overall)	0.8932	0.806	-0.0872
Precision (macro)	0.931	0.847	-0.084
Recall (macro)	1.000	0.926	-0.074
Inference Latency (ms/image)	7.60	4,991.5	+4,983.9 ($\approx 657\times$)

3.2 Discussion

The Bayesian-TPE configuration improves training-time mAP@0.5 to 0.984 against sub-90% baselines reported for non-HPO-tuned YOLO variants on comparable disease-detection tasks (Wisnawa et al., 2026). However, the present on-device evaluation reveals that this gain is only partially retained after TFLite conversion and integration into the mobile preprocessing pipeline: mAP@0.5 falls from 0.984 (original PyTorch checkpoint, identical test set) to 0.908 post-deployment, a 7.6-percentage-point retention loss not previously measured in the optimization-focused source study. The loss is markedly class-imbalanced, Anthracnose AP@0.5 falls by only 3.7 pp (0.995 to 0.958) and Healthy Cocoa Pod by 0.6 pp (0.995 to 0.989), while Black Pod Rot falls by 18.4 pp (0.962 to 0.778). Manual inspection of the five test images responsible for the residual error mass, two Black Pod Rot to Anthracnose misclassifications and three duplicate (non-suppressed) detections, shows all five ground-truth bounding boxes cover an unusually large fraction of the frame (mean

29.3% of image area vs. an unquantified but visibly smaller norm for correctly classified instances), consistent with (i) reduced contextual/surface information available to the classification head under close-range framing and (ii) a fixed 0.45 IoU-NMS threshold becoming less discriminative between two legitimate candidate boxes on a single large object. This finding directly corroborates the source study's own confusion-matrix analysis, which independently identified Black Pod Rot as the dominant source of classification ambiguity due to its diffuse, irregular necrotic lesion boundary (Wisnawa et al., 2026). The dominant cost of on-device deployment in this study is latency, not accuracy. While mAP@0.5 retention loss is a bounded single-digit-percentage effect, per-frame inference latency increases by two to three orders of magnitude relative to the GPU-based training-time measurement (7.60 ms/image to 4,991.5 ms/image on-device), reflecting the absence of hardware-accelerated delegate execution (NNAPI/GPU) in the present CPU-only deployment configuration. At 12.3 MB, the exported TFLite artifact is compact enough for practical one-time distribution even under constrained mobile bandwidth, and, combined with the fully on-device inference pipeline described in Section 2.3, requires no sustained network connectivity during field operation, a property directly relevant to smallholder cocoa plantations in the remote, connectivity-limited agroecosystems targeted by the source optimization study (Wisnawa et al., 2026). The resulting throughput (~ 0.20 FPS on a POCO M4 Pro) falls substantially below conventional real-time thresholds (≥ 10 -15 FPS). We report this not as a deployment failure but as a quantified empirical characterization of the accuracy-latency Pareto frontier for anchor-free nano-scale YOLO models under unaccelerated mobile CPU execution, a gap not previously measured for this model family in the cocoa-disease-detection literature, and one that motivates the adaptive-scheduling design of Section 2.3 (a fixed-timeout pipeline would misclassify the observed latency distribution's upper tail as isolate failure rather than expected variance). Supplementary stability instrumentation (Section 2.3) found no evidence of memory leaks across a fully randomized, replicated 8-condition trial matrix (2 modes $\times$ 2 input sources $\times$ 3 capture intervals, one-way ANOVA on RSS growth slope vs. interval: $F(2,34) = 0.104$, $p = 0.90$) and no OS-level thermal escalation across sessions up to 90 minutes, indicating that, latency notwithstanding, the deployed pipeline is suitable for sustained field sessions without progressive resource degradation.

Unlike the preceding two findings, this result is architectural and descriptive rather than empirically measured, and we report it as such rather than implying a validation standard it does not meet. The disease-guidance module described in Section 2.3 demonstrates that etiology-, prevention-, and treatment-oriented content can be integrated as a first-class, fully offline, in-workflow feature triggered directly from a detection result, establishing technical feasibility. The remaining content-governance question surfaces a single, concrete finding: bundling advisory content as a static asset decouples its release cycle from model updates, which is operationally convenient but also means the content's accuracy is fixed at build time with no runtime correction path; we address this specific risk through an explicit, user-visible disclaimer rather than presenting the guidance as authoritative, and we treat this disclosure practice itself as the governance contribution, not merely a caveat appended to it. What this result does not include is any evidence of farmer comprehension, trust calibration, or behavioral

effect, no usability study, comprehension testing, or field engagement measurement was conducted. We therefore frame the module as a validated technical capability whose advisory content, while sourced from a published phytopathological reference, has an in-application adaptation that remains independently unvalidated, and treat empirical assessment of both the adaptation and end-user comprehension as necessary, distinct next steps (Section 4) rather than a claim this study makes.

4 Conclusion

This study deploys an Optuna Bayesian-TPE-optimized YOLOv8n checkpoint, selected over a marginally higher-mAP Hyperband alternative on the basis of zero-false-negative Macro Recall and lower inference latency, into a Clean-Architecture Flutter application via TFLite, and provides the first on-device validation of a model previously evaluated only in its pre-conversion, GPU-executed form. Post-deployment evaluation on the original study's held-out 30-image test set shows $\text{mAP}@0.5 = 0.908$ (down from 0.984 pre-conversion), with degradation concentrated in the Black Pod Rot class (-18.4 pp), correlated with large-bounding-box, close-range framing in the affected test images. On-device inference throughput (~0.20 FPS on a POCO M4 Pro) falls well short of conventional real-time thresholds under CPU-only execution, a gap we quantify rather than obscure, and which we identify as the dominant accuracy-latency trade-off of this deployment, not detection-quality loss, which remains bounded to single-digit percentage points for two of three classes. Extensive controlled stability testing found no measurable memory leak (ANOVA $p = 0.90$ across capture intervals) and no OS-reported thermal throttling across sessions up to 90 minutes, supporting the pipeline's suitability for sustained field use once inference latency is addressed. Future work directions include: (i) hardware-delegate acceleration (NNAPI/GPU) to close the measured real-time gap; (ii) Grad-CAM/EigenCAM-based model-interpretability overlays, distinct from the farmer-facing disease-guidance module already implemented in Section 2.3, for internal model-trust calibration, not implemented in the present study; (iii) expansion of the 30-image test set with a pre-registered statistical power target, particularly additional early-incubation-stage Black Pod Rot samples, as independently recommended by the source optimization study; and (iv) multi-device generalization testing beyond the single evaluation unit used here.

References

- Aji, N. B., Yudiantoro, T. R., Safitri, Z., Kuntardjo, S. B., Mardiyono, M., Prayitno, P., & Santoso, K. (2025). Application of MobileNetV2-Based Deep Learning in Detecting Diseases in Chili Plants. *Journal of Informatics Information System Software Engineering and Applications (INISTA)*, 7(2), 203–212. <https://doi.org/10.20895/inista.v7i2.1825>
- Alvarado, J., Restrepo-Arias, J. F., Velásquez, D., & Maiza, M. (2025). Disease Detection on Cocoa Crops Based on Computer-Vision Techniques: A Systematic Literature Review. *Agriculture*, 15(10), 1032. <https://doi.org/10.3390/agriculture15101032>
- Alvarado, J., Restrepo-Arias, J. F., Velásquez, D., Martinez-Vargas, J. D., & Maiza, M. (2025). Precision Agriculture Applied to Cocoa: ML Solutions for Classification

- of Moniliophthora Roreri. In C. Kahraman, S. Cebi, B. Oztaysi, S. Cevik Onar, C. Tolga, I. Ucal Sari, & I. Otay (Eds.), *Intelligent and Fuzzy Systems* (pp. 79–87). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-98565-2_10
- Antwi, K., Bennin, K. E., Pobi Asiedu, D. K., & Tekinerdogan, B. (2024). On the Application of Image Augmentation for Plant Disease Detection: A Systematic Literature Review. *Smart Agricultural Technology*, 9, 100590. <https://doi.org/10.1016/j.atech.2024.100590>
- Bahaa, M., Hesham, A., Ashraf, F., & Abdel-Hamid, L. (2026). A Smart AIoT-Based Mobile Application for Plant Disease Detection and Environment Management in Small-Scale Farms Using MobileViT. *AgriEngineering*, 8(1), 11. <https://doi.org/10.3390/agriengineering8010011>
- Berckemas, Y., Ariati, P. E. P., & Sujana, I. P. (2025). Eksplorasi Kejadian Penyakit Busuk Buah (Phytophthora Palmivora) pada Tanaman Kakao di Desa Cau Kecamatan Marga Kabupaten Tabanan. *Agrofarm: Jurnal Agroteknologi*, 4(1), 30–36. <https://doi.org/10.36733/agrofarm.v4i1.11787>
- Bhargava, A., Shukla, A., Goswami, O. P., Alsharif, M. H., Uthansakul, P., & Uthansakul, M. (2024). Plant Leaf Disease Detection, Classification, and Diagnosis Using Computer Vision and Artificial Intelligence: A Review. *IEEE Access*, 12, 37443–37469. <https://doi.org/10.1109/ACCESS.2024.3373001>
- Bhuiyan, Md. A. B., Abdullah, H. M., Arman, S. E., Saminur Rahman, S., & Al Mahmud, K. (2023). BananaSqueezeNet: A Very Fast, Lightweight Convolutional Neural Network for the Diagnosis of Three Prominent Banana Leaf Diseases. *Smart Agricultural Technology*, 4, 100214. <https://doi.org/10.1016/j.atech.2023.100214>
- Chung, H., Kim, J.-H., Ahn, J., Chung, Y., Kim, E., & Heo, W. (2026). Resource-Constrained Edge AI Solution for Real-Time Pest and Disease Detection in Chili Pepper Fields. *Agriculture*, 16(2), 223. <https://doi.org/10.3390/agriculture16020223>
- Dinas Pertanian, Ketahanan Pangan Dan Perikanan. (2025, September 16). *Mengenai Hama dan Penyakit Pada Tanaman Kakao Penyebab Rendahnya Produksi Kakao Busungbiu*. Dinas Pertanian, Ketahanan Pangan Dan Perikanan. https://distankan.bulelengkab.go.id/informasi/detail/berita/23_mengenai-hama-dan-penyakit-pada-tanaman-kakao-penyebab-rendahnya-produksi-kakao-busungbiu
- Duhan, S., Gulia, P., Gill, N. S., & Narwal, E. (2025). RTR_Lite_MobileNetV2: A Lightweight and Efficient Model for Plant Disease Detection and Classification. *Current Plant Biology*, 42, 100459. <https://doi.org/10.1016/j.cpb.2025.100459>
- Garma, G. M. (2025). Efficient Detection of Cacao Pod Diseases using SSD MobileNetV2 FPN-Lite. *World Journal of Advanced Research and Reviews*, 25(2), 1099–1105. <https://doi.org/10.30574/wjarr.2025.25.2.0128>
- Gkountakos, K., Ioannidis, K., Demestichas, K., Vrochidis, S., & Kompatsiaris, I. (2024). A Comprehensive Review of Deep Learning-Based Anomaly Detection Methods for Precision Agriculture. *IEEE Access*, 12, 197715–197733. <https://doi.org/10.1109/ACCESS.2024.3522248>
- Goklani, D. (2024). *Real-Time Plant Disease Detection Using Mobile Device with TensorFlow Lite and Flutter* (SSRN Scholarly Paper No. 4921827). Social Science Research Network. <https://doi.org/10.2139/ssrn.4921827>
- Gui, P., Dang, W., Zhu, F., & Zhao, Q. (2021). Towards Automatic Field Plant Disease Recognition. *Comput. Electron. Agric.*, 191(C). <https://doi.org/10.1016/j.compag.2021.106523>

- Janarthan, S., Thuseethan, S., Rajasegarar, S., & Yearwood, J. (2022). P2OP—Plant Pathology on Palms: A Deep Learning-based Mobile Solution for In-Field Plant Disease Detection. *Computers and Electronics in Agriculture*, 202, 107371. <https://doi.org/10.1016/j.compag.2022.107371>
- Kumi, S., Kelly, D., Woodstuff, J., Lomotey, R. K., Orji, R., & Deters, R. (2022). Cocoa Companion: Deep Learning-Based Smartphone Application for Cocoa Disease Detection. *Procedia Computer Science, 17th International Conference on Future Networks and Communications / 19th International Conference on Mobile Systems and Pervasive Computing / 12th International Conference on Sustainable Energy Information Technology (FNC/MobiSPC/SEIT 2022), August 9-11, 2022, Niagara Falls, Ontario, Canada*, 203, 87–94. <https://doi.org/10.1016/j.procs.2022.07.013>
- Pagaduan, Z., Occidental, J., Duro, N., Badilles, D., & Palconit, E. (2026). *Development of a Cacao Disease Identification and Management App Using Deep Learning* (arXiv:2602.00216). arXiv. <https://doi.org/10.48550/arXiv.2602.00216>
- Rashwan, S. A., & Elteir, M. K. (2022). Plant Leaf Disease Detection using Deep Learning on Mobile Devices. *International Journal of Computational Vision and Robotics*, 12(2), 156–176. <https://doi.org/10.1504/IJCVR.2022.121151>
- Restrepo-Arias, J. F., Montoya-Castaño, M. J., Moreno-De La Espriella, M. F., & Branch-Bedoya, J. W. (2025). An Application of Deep Learning Models for the Detection of Cocoa Pods at Different Ripening Stages: An Approach with Faster R-CNN and Mask R-CNN. *Computation*, 13(7), 159. <https://doi.org/10.3390/computation13070159>
- Sajitha, P., Andrushia, A. D., Anand, N., & Naser, M. Z. (2024). A Review on Machine Learning and Deep Learning Image-based Plant Disease Classification for Industrial Farming Systems. *Journal of Industrial Information Integration*, 38, 100572. <https://doi.org/10.1016/j.jii.2024.100572>
- Shoaib, M., Sadeghi-Niaraki, A., Ali, F., Hussain, I., & Khalid, S. (2025). Leveraging Deep Learning for Plant Disease and Pest Detection: A Comprehensive Review and Future Directions. *Frontiers in Plant Science*, 16, 1538163. <https://doi.org/10.3389/fpls.2025.1538163>
- Soesanto, L. (2024). *Kompedium Penyakit-Penyakit Kakao*. Penerbit Andi.
- Upadhyay, A., Chandel, N. S., Singh, K. P., Chakraborty, S. K., Nandede, B. M., Kumar, M., Subeesh, A., Upendar, K., Salem, A., & Elbeltagi, A. (2025). Deep Learning and Computer Vision in Plant Disease Detection: A Comprehensive Review of Techniques, Models, and Trends in Precision Agriculture. *Artificial Intelligence Review*, 58(3), 92. <https://doi.org/10.1007/s10462-024-11100-x>
- Wisnawa, I. P. O., Prihatini, P. M., Saptarini, N. G. A. P. H., Sulastra, I. M. D. J., & Dewi, N. L. P. L. (2026). Penerapan Bayesian-Optimized Deep Learning Framework Berbasis YOLOv8 dan Optuna untuk Deteksi Multikelas Penyakit Kakao. *METHOMIKA: Jurnal Manajemen Informatika & Komputerisasi Akuntansi*, 10(1), 326–343. <https://doi.org/10.46880/jmika.Vol10No1.pp326-343>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (<http://creativecommons.org/licenses/by-nc/4.0/>), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

