

Article

eBirdNet-Nano: An Operator-Aware Lightweight Detector and Edge AI Terminal for Endangered Bird Real-Time Monitoring

Xiaoyuan Huang ^{1,2,*} , Lu Shen ¹  and Su-Kit Tang ^{1,*} ¹ Faculty of Applied Sciences, Macao Polytechnic University, Macao SAR 999078, China; lu.shen@mpu.edu.mo² School of Electrical Engineering and Intelligentization, Dongguan University of Technology, Dongguan 523808, China

* Correspondence: xiaoyuan.huang@mpu.edu.mo or huangxy@dgut.edu.cn (X.H.); sktang@mpu.edu.mo (S.-K.T.)

Abstract

Real-time monitoring of endangered birds on edge AI hardware is constrained by a structural mismatch between modern lightweight detectors and mainstream NPU deployment toolchains. Recent attention-based detectors rely heavily on dynamic-shape operators that fall back to the host CPU on embedded NPUs, negating the advantages of lightweight architectures. To address this, we propose eBirdNet-Nano, a 1.05 M-parameter detector derived from YOLOv12n through a three-level NPU-friendly redesign: a static NPUConv block at the operator level, an NPU-C3k2 module together with an NPU-SE-Block at the module level, and a balanced 64-channel detection head at the head level. The resulting model achieves a 59% parameter reduction over YOLOv12n at only 5.8 GFLOPs while attaining an mAP@0.5 of 0.929 on a curated 24-species endangered-bird dataset collected in Macao. We further evaluate the model across four heterogeneous edge platforms—the Rockchip RK3588 (ARM + NPU), Kendryte K230 (RISC-V + KPU), Raspberry Pi 4B (pure ARM), and LicheePi 4A (pure RISC-V)—to characterize its behavior under distinct execution models. On the RK3588 NPU under INT8 quantization, eBirdNet-Nano delivers 13.83 ms inference latency and 26.76 ms end-to-end latency at 37.4 FPS, attaining the best parameter–latency balance and the highest parameter-normalized throughput (35.62 FPS/M) among six nano-scale YOLO variants, with an overall $3.53\times$ end-to-end speedup over the YOLOv12n FP16 baseline that decomposes into a $2.97\times$ architectural factor and a $1.19\times$ quantization factor. Integrated into the EbirdEye field terminal, the same model sustains 23.5 ms thread-level end-to-end latency during live operation while supporting approximately 13.5 h of battery-powered runtime per charge. The proposed design offers a practical pathway toward deployable, low-power AI terminals for endangered-species conservation in resource-constrained field environments.



Academic Editor: Aryya Gangopadhyay

Received: 24 May 2026

Revised: 14 June 2026

Accepted: 21 June 2026

Published: 1 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.**Keywords:** endangered bird detection; edge AI; NPU-aware lightweight network; operator-friendly design; field deployment

1. Introduction

The wetlands of the Pearl River Estuary support wintering populations of several International Union for Conservation of Nature (IUCN)-listed endangered birds [1], of which the *Platalea minor* (Black-faced Spoonbill) is the most emblematic: roughly six thousand individuals remain worldwide [2,3], and a meaningful fraction of them visit the Macao Cotai Ecological Reserve between December and February each year [4–6]. Continuous,

automated monitoring of these populations would allow conservation managers to detect arrivals earlier, log behavior over the full diel cycle rather than only during expert-led surveys, and accumulate the long-baseline records on which habitat-management decisions ultimately depend. The technical question is how to deliver this capability at a cost and form factor that scale to dozens of unattended observation nodes deployed across a fragile reserve, sustained by battery and solar power rather than by mains and bandwidth. Existing field-deployed bird-detection systems either offload inference to a cloud server reached over a 4G uplink [7] or run on phone-class central processing unit/graphics processing unit (CPU/GPU) hardware [8], and although recent wetland datasets such as DTH-Birds have begun to address the data side of the problem [9], no openly released visual system to date targets the endangered avifauna of the Pearl River Estuary on consumer-grade neural processing unit (NPU) hardware.

Edge artificial intelligence (AI) hardware has matured to the point where this objective is no longer obviously implausible. A modern heterogeneous system-on-chip such as the Rockchip RK3588 packages an octa-core ARM CPU together with a 6-trillion-operations-per-second (TOPS) NPU within a sub-10 W envelope, sufficient in principle for real-time visual detection on a small battery-powered terminal. Detection-model design, however, has not co-evolved with this class of accelerator. The You Only Look Once (YOLO) family, which has dominated real-time detection on the desktop and on the GPU since Redmon et al. [10] and through the v3–v11 lineage [11–14], has moved over its most recent releases towards attention-based modules: YOLOv12 [15] introduces an Area-Attention block (A2C2f) and a pointwise self-attention block (C2PSA) that lift Common Objects in Context (COCO) accuracy [16] but rely on dynamic-shape tensor operations. On consumer NPU toolchains—RKNN-Toolkit2 [17], nncase [18], NCNN [19]—those operations are not natively supported and are silently offloaded to the host CPU [20], where they execute several times more slowly than they would on the NPU and break the static schedule that low-precision quantization depends on. Reproducible benchmarks on RK3588-class hardware confirm that operator type consistently outweighs parameter count as a predictor of on-device latency [21], with effective NPU utilization falling below 55% under such fallbacks—turning a nominal $2\text{--}3\times$ 8-bit integer (INT8) speedup into a marginal gain or a regression [20]. Lightweight backbones such as MobileNet [22], ShuffleNetV2 [23] and GhostNet [24] mitigate the floating-point operations (FLOPs) side of this problem but offer no guarantee on the operator side, because their evaluations are conducted against GPU or general-purpose CPU latency rather than against an NPU operator-support matrix [25].

A natural response is to delegate the gap to quantization or to the compiler: with a well-chosen calibration set and a tidy export script, post-training quantization [26] or quantization-aware training [27] should, on this view, recover most of the latency. We would suggest, however, that this view is incomplete in a structural rather than a numerical sense. INT8 compresses arithmetic cost but does not relocate operators back onto the accelerator. A model whose dynamic-shape operators sit on the CPU at 16-bit floating-point (FP16) precision still has them sitting on the CPU at INT8, and the per-frame CPU–NPU dispatch overhead persists regardless of numerical precision—a pattern recently characterized in detail for on-device large language model (LLM) workloads [28] and consistent with the latency profiles reported across YOLO variants on RK3588 [21]. The natural place to intervene is therefore earlier in the pipeline—at the operator graph itself, before quantization is applied. Edge-YOLO variants for RK3588 [29] have begun to pursue this direction by substituting incompatible activations, but operator-level adaptation has not yet been treated as a coordinated, multi-level methodology, nor has its outcome been validated end-to-end on a deployed terminal.

A further observation, less often made in the model-design literature, is that “real-time” as a board-level benchmark and “real-time” as a field-deployment property are not the same statement. A detector that hits 30 frames per second (FPS) on a development board running a controlled benchmark loop may fail to do so once the same system-on-chip (SoC) is wrapped into a complete observation terminal—with a universal video class (UVC) capture pipeline, a touch display, a multi-thread runtime and a battery-bound power envelope—because peripheral overhead, thread contention and capture-side cadence start to dominate the budget. Closing this gap requires not only a lightweight detector but also a deployable terminal whose end-to-end behavior and energy profile have been measured in the field rather than inferred from the silicon datasheet.

We develop these claims concretely for the task of detecting twenty-four protected and endangered bird species in the Macao region. Starting from a YOLOv12n base [15], we apply a three-level NPU-friendly transformation: at the operator level, every SiLU-activated convolution is rewritten as a static-shape NPUConv block (convolution–batch normalization (BN)–ReLU6); at the module level, the dynamic Area-Attention (A2C2f) and pointwise self-attention (C2PSA) blocks are replaced by a Cross-Stage Dense Convolution module (NPU-C3k2) and an NPU squeeze-and-excitation block (NPU-SE-Block); at the head level, the three detection branches are rebalanced to a uniform 64-channel output. The resulting network, eBirdNet-Nano, is designed so that its full computational graph remains resident on the NPU under the operator-support matrix of mainstream consumer toolchains, while preserving sufficient representational capacity for fine-grained species discrimination on a small in-domain dataset. We then integrate eBirdNet-Nano into EbirdEye, a battery- and solar-powered observation terminal built around the LubanCat-RK3588s board with a UVC camera and a touchscreen, and characterize its end-to-end runtime and energy budget under live capture in the Macao Cotai Ecological Reserve.

Against this background, the contributions of this paper are threefold.

1. An NPU-aware detector and three-level transformation methodology for endangered-bird monitoring. We present eBirdNet-Nano, a compact detector for twenty-four protected species in the Macao region, together with the curated Macao-ebird-det dataset [30]. It instantiates a three-level NPU-aware transformation pipeline acting at the operator (NPUConv), module (NPU-C3k2, NPU-SE-Block) and detection-head (balanced 64-channel) levels, each removing a distinct class of CPU-fallback trigger from the computational graph. To our knowledge, this is the first openly released visual-detection system that explicitly targets the Macao endangered avifauna on consumer-grade NPU hardware.
2. End-to-end deployment of the detector on an RK3588 NPU and across heterogeneous edge platforms. We take eBirdNet-Nano through the full deployment pipeline of Section 3.2—32-bit floating-point (FP32) Open Neural Network Exchange (ONNX) export, graph simplification, Kullback–Leibler (KL)-divergence INT8 calibration, RKNN compilation—and report its on-device behavior on a LubanCat-4 RK3588 board at 320×320 , profiling NPU inference latency, end-to-end pipeline timing, board-level power and per-frame energy under both FP16 and INT8. To put these numbers in context we benchmark five contemporary nano-scale YOLO variants through the same toolchain on the same hardware, and we re-run the model on two additional architectures—a Raspberry Pi 4B (ARM CPU) and a LicheePi 4A (RISC-V CPU) through ONNX Runtime—so that the detector is characterized not only on its primary target but also on the alternative execution models a field deployment would realistically have to choose between.
3. Field-validated system-level deployment on the EbirdEye terminal. We report runtime and power measurements collected in situ on EbirdEye, an RK3588s-based observation

terminal that integrates eBirdNet-Nano with three optical paths (short range to long range), a touch display and a battery-and-solar power supply. The measurements characterize the full pipeline at the thread level—capture, inference and display—and connect laboratory throughput to deployable field runtime through a system-level power and endurance budget, moving the real-time claim from a benchmark statement to a property of an integrated terminal.

The remainder of the paper is organized as follows. Section 2 reviews edge detection work most relevant to NPU deployment. Section 3 describes the three-level transformation methodology and the resulting eBirdNet-Nano architecture. Section 4 specifies the experimental setup. Section 5 reports the experimental analysis, covering detection accuracy, the staged ablation across in-domain and out-of-domain datasets, cross-model NPU latency, the FP16/INT8 quantization decomposition, board-level power and energy, cross-architecture portability on RISC-V, and the system-level field deployment of the EbirdEye terminal in the Macao Cotai Ecological Reserve. Section 6 concludes the paper.

2. Related Work

2.1. Detection-Based Bird Monitoring on Edge Devices

Field-deployed bird monitoring has shifted from offline camera-trap pipelines toward systems capable of inference at the point of capture. Chalmers et al. [7] deployed a YOLOv10-based detector across eleven 4G-connected camera traps in Wales for endangered Curlew monitoring (F1-score (F1) = 95.05%), with inference offloaded to a cloud server. Félix-Jiménez et al. [8] combined YOLOv8s with MobileNetV3 for five native species in Zatecas and reported explicit power measurements—still uncommon in this literature. Fang et al. [9] assembled the DTH-Birds dataset at Dongting Lake and proposed Birds-YOLO, a YOLOv11 variant with exponential moving average (EMA) attention targeting 47 wetland species. Earlier work covers context-aware attention for small unmanned aerial vehicle (UAV) targets [31], weakly supervised detection in dense UAV scenes [32], and two-stage YOLOv5+EfficientNetB3 pipelines for joint detection and species classification [33]. Xing et al. [34] specifically addressed the challenge of monitoring small, low-contrast bird targets in dense wetland environments by proposing an optimized YOLO-based detector, enhancing local feature extraction for distant targets. Rossi Labianca et al. [35] designed a dedicated on-edge AI architecture for real-time bird identification, targeting hardware-software co-design on low-power edge nodes like NVIDIA Jetson boards to enable continuous ecosystem protection.

In the acoustic domain, BirdNET [36] established convolutional neural network (CNN)-based spectrogram analysis as the standard for large-scale passive monitoring, with later work applying EfficientNet to multi-species classification on Mel spectrograms [37].

The pattern revealed by Table 1 is consistent across the ten works surveyed. Detection accuracy is almost uniformly the primary optimization target, with the choice of backbone (YOLOv5–v11 variants) driven by GPU benchmarks rather than by the operator set of any specific deployment runtime. Edge deployment, where reported, falls into two distinct modes: cloud-assisted inference connected by a 4G uplink [7], or on-device CPU/GPU execution on a mobile phone [8]—but in neither case is NPU operator compatibility analyzed. Quantization is reported only by Félix-Jiménez et al., who also provide the only explicit power measurement in the visual-detection cluster.

Table 1. Comparative summary of representative bird-detection works on edge/mobile devices.

Work	Year	Method	Datasets	Species	Edge.	Infer.	NPU.	Qtz.	R.T.	Pwr.
Ma et al. [31]	24	Yv8+Attn.	PYL-5 (UAV)	5	–	GPU	✗	✗	–	✗
Kellenberger et al. [32]	18	Weak CNN	UAV	–	–	GPU	✗	✗	–	✗
Vo et al. [33]	23	Yv5+EffB3	eBird/self	multi	–	GPU	✗	✗	–	✗
Chalmers et al. [7]	25	Yv10	Wales	1	4G trap	Cloud	✗	✗	✓	✗
Félix-Jiménez et al. [8]	25	Yv8s+MNetV3	Zacatecas	5	Mobile	CPU/GPU	✗	✓	✓	✓
Fang et al. [9]	25	Birds-YOLO	DTH	47	–	GPU	✗	✗	–	✗
Xing et al. [34]	26	Yv8-Wetland	Wetland	multi	–	GPU	✗	✗	✓	✗
Rossi Labianca et al. [35]	26	CNN-Edge	Private	multi	Jetson	Edge	✗	✓	✓	✗
Kahl et al. [36]	21	CNN (aud.)	Xeno-canto	~3 k	RPi	CPU	✗	✗	✓	✗
Baowaly et al. [37]	24	EffNet	BirdCLEF	264	–	GPU	✗	✗	–	✗
Ours	–	eBirdNet-nano	Macao-ebird-det	24	RK3588	NPU	✓	✓	✓	✓

Note: Edge. = Edge Platform; Infer. = Inference Site; NPU. = NPU-aware design constraint; Qtz. = INT8 quantization; R.T. = on-device real-time reported; Pwr. = Power Reported; ✓ = supported/reported; ✗ = not supported/not reported.

2.2. Single-Stage Detection and Lightweight Network Design

Single-stage detectors have driven real-time vision on edge hardware since Redmon et al.'s original YOLO [10], in which localization and classification share a single forward pass—structurally better suited to constrained deployment than two-stage methods. Across the v3–v8 lineage the architecture remained essentially convolutional, with successive refinements: CSP/PANet integration with Mosaic augmentation, an industrial PyTorch toolchain, and anchor-free decoupled heads [11]. YOLOv9 introduced Programmable Gradient Information [12]; YOLOv10 adopted end-to-end non-maximum suppression (NMS)-free training [13]; YOLOv11 added a pointwise self-attention module (C2PSA) on a refined C3k2 backbone [14]. YOLOv12 [15] marked a qualitative shift: the Area-Attention module (A2C2f) replaces dense convolution with sparse, region-based self-attention to reach state-of-the-art accuracy on COCO [16]. Chandrashekhara et al. [38] refine YOLOv12 with multi-scale fusion for extremely small targets, Liang et al. [39] propose a re-parameterizable convolution and a center-guided cross-scale fusion network (LCS-YOLO) on the YOLOv12n backbone for UAV imagery. Qiu and Lin [40] introduce a lightweight architecture (LMW-YOLO) that balances detection accuracy with computational efficiency, making it suitable for real-time applications and resource-constrained environments.

Complementary lightweight recipes reduce parameters and computation while preserving capacity. MobileNet [22] factorizes standard convolutions into depthwise-separable operations; ShuffleNetV2 [23] re-examines design from a memory-access-cost perspective, showing that channel-split-and-shuffle can match MobileNet at the same FLOPs budget; GhostNet [24] regenerates redundant feature maps via cheap linear transforms; VanillaNet [41] demonstrates that pure convolutional stacks can rival attention-based designs under appropriate training. A persistent caveat is that all of these are validated against GPU or general-purpose CPU latency. Li et al. [25] showed empirically that latency on heterogeneous mobile SoCs is only loosely predicted by FLOPs, with memory access patterns and operator scheduling accounting for a large share of observed cost. The implication for NPUs is direct: parameter efficiency on a GPU does not survive transfer to an NPU if the computational graph contains operators the runtime cannot natively accelerate. For multi-scale fusion, Feature Pyramid Network (FPN) [42] and PANet [43] remain the standard neck design across the YOLO family because neither introduces dynamic shapes.

2.3. NPU-Aware Operator Design, Quantization, and Edge Deployment

Modern edge NPUs are built around systolic-array or vector-processing architectures optimized for data reuse and on-chip static random-access memory (SRAM) access. Commercial runtimes—RKNN-Toolkit2 [17], nncase [18], NCNN [19], ONNX Runtime [44]—compile a supported subset of ONNX operators into native instructions; any operator outside this subset is silently offloaded to the host CPU. For the v3–v8 family, whose graphs consist almost entirely of Conv2d, BatchNorm, ReLU and static Resize, fallback is rarely triggered. For YOLOv12-class models the situation is structurally different: Area-Attention relies on dynamic tensor partitioning and C2PSA on pointwise self-attention with variable spatial extent—both routed back to the CPU as dynamic-shape operations. Similarly, a 2026 study by Suchý and Turčaník [45] on the Raspberry Pi 5 and Jetson Orin NX confirms that NPU acceleration boosts FPS-per-watt but degrades performance during CPU-fallbacks, directly aligning with our operator-compatibility analysis and motivating our per-frame energy evaluation.

Operator-level adaptation has been explored along several directions. Replacing SiLU with ReLU6—a static Clip(0, 6) node natively supported by all major NPU toolchains—removes a common fallback source while preserving capacity, and its bounded output range tightens INT8 calibration. Depthwise-separable and channel-split structures are in principle well-matched to NPU multiply–accumulate (MAC) arrays, though throughput depends on whether the compiler fuses consecutive pointwise and depthwise layers [23]. Li and Ye [29] demonstrated an edge-YOLO variant for RK3588 by substituting incompatible activations, and Xu et al. [28] characterized the NPU-to-CPU fallback penalty for on-device LLM inference, showing scheduling overhead from unsupported operators dominates total latency in a pattern analogous to that observed for detection workloads.

Quantization is a complementary axis. Post-training quantization (PTQ) with Kullback–Leibler (KL)-divergence calibration is the dominant edge recipe: it requires no re-training and typically lands within 1–2 percentage points of FP16 on detection benchmarks while delivering 2–3× speedup on cleanly NPU-resident graphs [26]. Quantization-aware training (QAT) narrows the gap further [27]. Critically, both operate downstream of the operator graph: INT8 reduces arithmetic cost but cannot relocate operators that have already been delegated to CPU. A model whose dynamic-shape nodes fall back at FP16 retains the same fallback at INT8, with persistent dispatch overhead regardless of precision.

2.4. The Methodological Gap This Paper Addresses

Three observations summarize the state of the field as it bears on the present work. First, attention-based detectors have moved faster than NPU toolchains can absorb them. Second, FLOPs-driven lightweight design does not carry over cleanly to NPUs whose operator coverage is incomplete. Third, INT8 PTQ does not fix problems that originate in the operator graph. Together these point to a level of intervention—the operator graph itself—that has been touched only obliquely by published edge bird-detection systems. The contribution of this paper is to make that intervention concrete, to measure it carefully, and to show that on a representative consumer-NPU SoC the resulting model is Pareto-superior on the latency–parameter plane to every nano-scale YOLO variant from v8 to v12.

3. Method

3.1. NPU-Aware Lightweight Detector Design

eBirdNet-Nano is obtained from YOLOv12n by three transformations applied at progressively higher levels of the architecture: (i) operator-level substitution, in which every dynamic-shape activation is replaced by a static counterpart wrapped in a uniform NPUConv block; (ii) module-level substitution, in which the two attention-based blocks

introduced in YOLOv12 are replaced by static dense modules that recover the same channel-recalibration behavior without any dynamic shape; and (iii) head-level rebalancing, in which the three-scale detection head is compressed to a uniform 64-channel output and the distribution focal loss (DFL) reg-max is reduced to one. The three transformations are independent in the sense that each can be evaluated in isolation, and we report exactly that ablation in Section 5.3; in deployment they are applied jointly.

The block diagram in Figure 1 shows the resulting network. The backbone retains the staged feature extraction of YOLOv12 but replaces every internal block with NPU-resident counterparts. The neck retains the bidirectional Feature Pyramid Network (FPN)+path aggregation network (PAN) topology unchanged—the structure was already static-shape. The head is the only place where structural compression beyond operator substitution is applied, because the original head was sized for COCO’s 80 classes and is overspecified for our 24-species task.

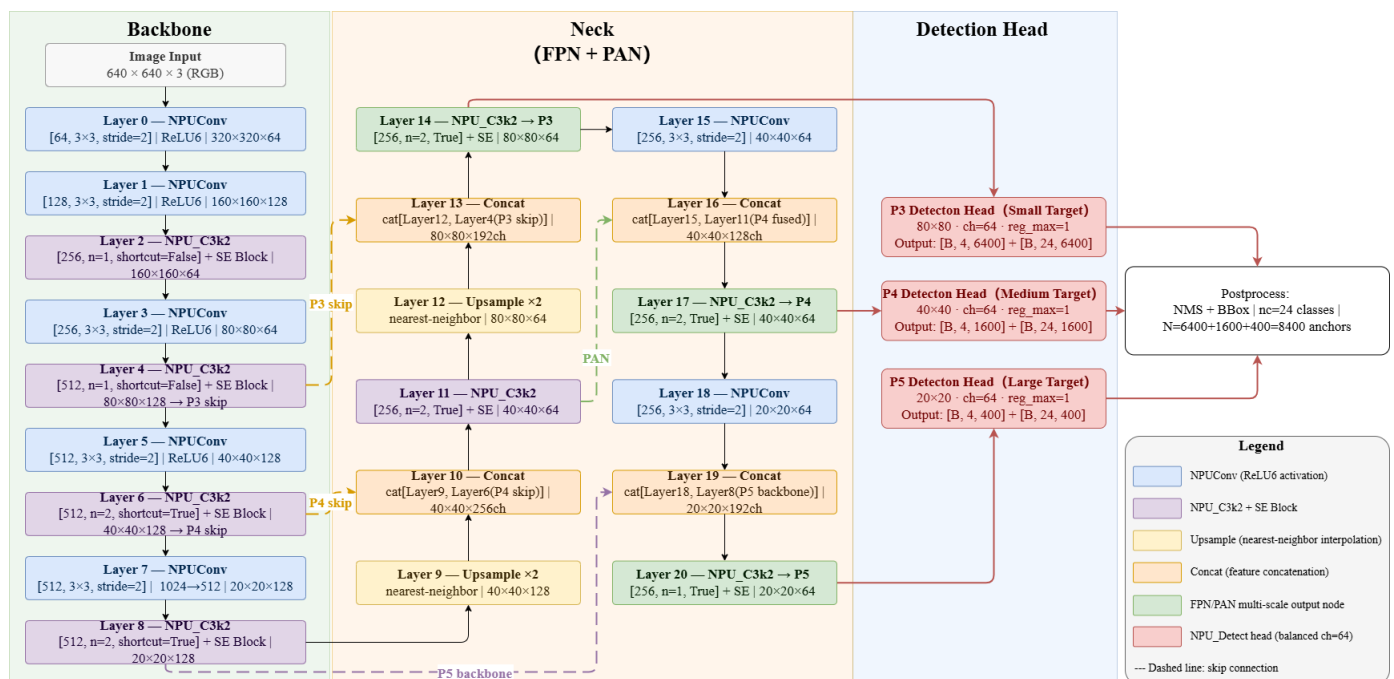


Figure 1. Overall architecture of eBirdNet-Nano.

3.1.1. Operator-Level: The NPUConv Block

The structure of the NPUConv block is shown in Figure 2. Every Conv–Activation pair in YOLOv12n uses SiLU, defined as

$$\text{SiLU}(x) = x \cdot \sigma(x), \quad (1)$$

where σ is the sigmoid. The sigmoid in turn is implemented on most consumer NPUs through a piecewise-linear approximation that is treated as a dynamic-range operator and is, in the FP16/INT8 paths of RKNN-Toolkit2 v2.0, a CPU fallback. The substitution we apply is to write every convolution as

$$\text{NPUConv}(x) = \text{ReLU6}(\text{BN}(\text{Conv}(x))) = \text{clip}(\text{BN}(\text{Conv}(x)), 0, 6). \quad (2)$$

Three properties make this substitution attractive. First, ReLU6 corresponds in the ONNX standard to a static Clip(min=0, max=6) operator and is natively supported by every NPU runtime we use. Second, the bounded output range [0, 6] gives the INT8 calibrator a tight dynamic range to fit, which substantially reduces the activation-quantization error

compared to SiLU's unbounded positive tail. Third, batch normalization positioned before the clip continues to whiten activations and stabilize training; we observed no convergence issues across either of the two datasets used in Section 5.

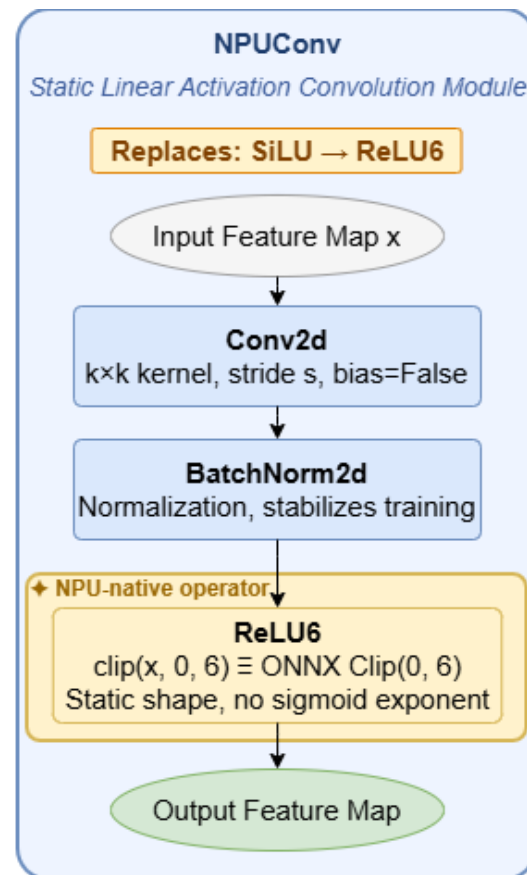


Figure 2. Structure of the NPUConv block.

3.1.2. The NPU-C3k2 Cross-Stage Dense Convolution Module

The internal structure of the NPU-C3k2 module is illustrated in Figure 3. YOLOv12's A2C2f module computes Area-Attention over partitioned spatial regions; the partitioning step itself is a dynamic-shape gather. We replace A2C2f by a Cross-Stage Dense Convolution module (NPU-C3k2) whose forward pass is

$$\text{NPU-C3k2}(x) = \text{SE}\left(\text{CV}_2\left(\text{Concat}\left[\text{CV}_1(x), B_1(\text{CV}_1(x)), \dots, B_n(\text{CV}_1(x))\right]\right)\right). \quad (3)$$

The input feature x is first expanded to $2c$ channels by a 1×1 NPUConv (CV_1), then split into two branches—one identity branch and one residual branch processed by n NPU-Bottleneck blocks (each consisting of two 3×3 NPUConv layers with a residual connection). The branch outputs and the n intermediate maps are concatenated along the channel dimension to give $(2 + n)c$ channels and compressed back to c_2 channels by a final 1×1 NPUConv (CV_2). A lightweight NPU-SE-Block (described next) recalibrates the result.

The point of this construction is that it preserves the cross-stage connectivity and the depth of A2C2f's representation while replacing every dynamic-shape gather with a static dense convolution. The theoretical FLOPs rise modestly compared to the sparse-attention version, but the resulting graph has zero CPU fallback, which is the property that determines wall-clock latency.

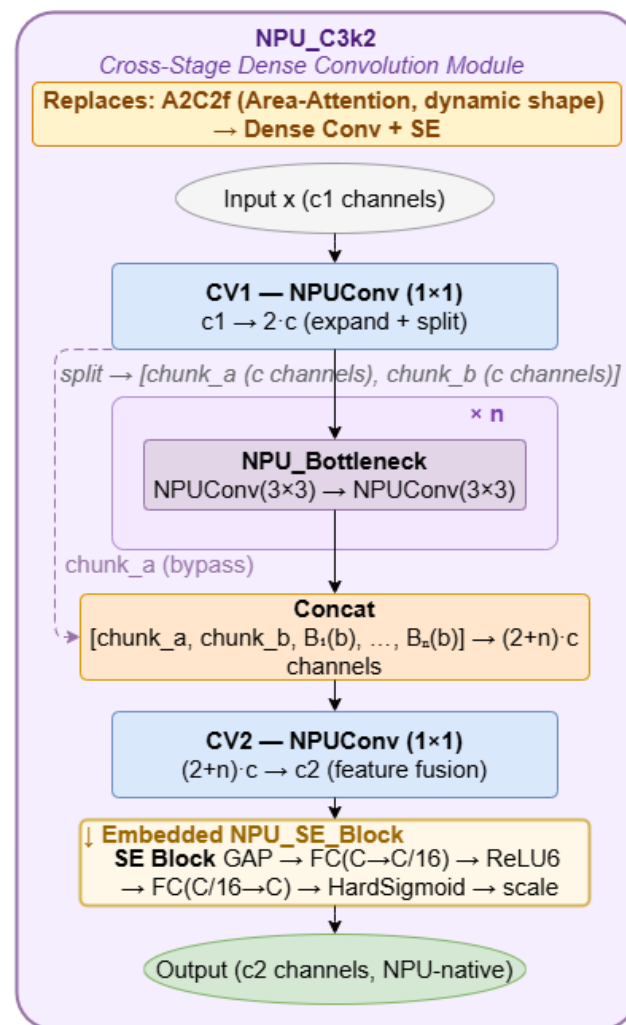


Figure 3. Structure of the NPU-C3k2 cross-stage dense convolution module.

3.1.3. The NPU-SE-Block Static Channel-Excitation Gate

The data flow of the NPU-SE-Block is shown in Figure 4. YOLOv12's C2PSA module performs pointwise self-attention. The pointwise step is itself a dynamic-shape operation. We replace C2PSA by a static channel-wise excitation block whose forward pass is written using global average pooling (GAP) and fully connected (FC) layers as

$$\text{NPU-SE-Block}(x) = x \otimes \text{HardSigmoid}(\text{FC}_2(\text{ReLU6}(\text{FC}_1(\text{GAP}(x))))). \quad (4)$$

Global average pooling collapses the spatial dimensions to a $B \times C$ descriptor. A bottleneck of two fully connected layers (with reduction ratio $r = 16$, no bias, ReLU6 between them) produces channel-wise gating logits, and HardSigmoid maps them to the unit interval. The gating vector is broadcast back along the spatial axes and multiplied element-wise with the original features. Every operator (AdaptiveAvgPool2d, Linear, ReLU6, HardSigmoid) is static-shape and natively supported by RKNN, nncase and NCNN. The block adds approximately 8 K parameters and, in our measurements, eliminates the CPU fallback that C2PSA introduced.

Substantively, the block is an NPU-friendly variant of the SE module [46] with HardSigmoid replacing the original sigmoid; its novelty here is the system-level choice of using SE in this position to absorb the role that C2PSA would have played while keeping the resulting graph entirely NPU-resident.

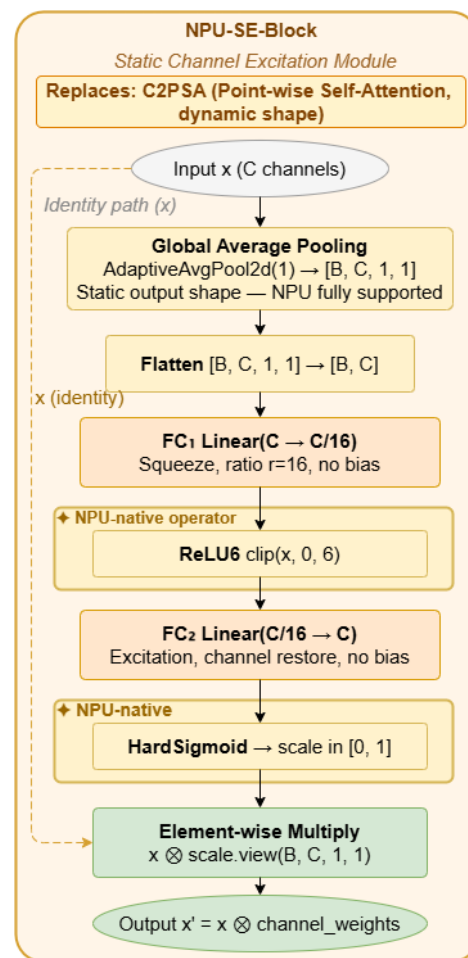


Figure 4. Structure of the NPU-SE-Block static channel-excitation module.

3.1.4. Head-Level: Balanced Compression

The standard YOLOv12n detection head outputs [256, 512, 512] channels across the P3, P4 and P5 scales. This unbalanced allocation reflects COCO's 80-class statistics, in which large objects dominate the loss and the deeper P5 head benefits from extra capacity. The bird-detection workload is structured differently. The 24 species in Macao-ebird-det are discriminated chiefly by gross morphology—silhouette, beak geometry, plumage color blocks—and the typical wetland scene contains small, distant targets. Detection difficulty across the three scales is roughly comparable, and the unbalanced head leaves substantial parameter budget on the table at P4 and P5.

Three structural compressions follow from this observation:

1. Channel rebalancing. All three detection branches are compressed to a uniform 64-channel output, which removes the channel asymmetry between P3 and P4/P5 and aligns the parameter budget with the task structure.
2. DFL simplification. Setting $\text{reg-max} = 1$ reduces the distribution focal loss (DFL) position regression from a multi-point distribution to a single-point prediction, which removes the softmax + gather overhead on the NPU (worth approximately 2–3 ms per frame in our measurements). The effect on localization accuracy is small in the bird-detection regime— mAP@0.5:0.95 moves from 0.849 to 0.848 when the simplification is applied on top of the channel-and-depth-compressed network—because at the intersection-over-union (IoU) thresholds that matter for ecological monitoring (mean average precision at IoU 0.5, mAP@0.5), bounding-box position is dominated by classification correctness rather than by sub-pixel regression. The same simplification

would be more costly in benchmarks where every percentage point of mAP@0.5:0.95 matters; we discuss this boundary explicitly in Section 5.3.

3. Backbone depth reduction. The P5 channel count is compressed from 1024 to 512, and the per-stage repetition count in the backbone is halved from 4 to 2, reducing parameter redundancy on a dataset of approximately 14 k training instances and lowering the risk of overfitting.

3.2. Inference-Aware Deployment Pipeline

To exploit that property, the model is taken through a deployment pipeline (Figure 5) whose stages are ordered so that operator-graph decisions precede precision-related decisions. Platform-agnostic front end. The FP32 PyTorch model is exported to ONNX and passed through onnx-simplifier for constant folding, dead-node removal, shape inference, operator fusion and static-shape fixation. The resulting FP32 ONNX graph is the routing anchor of the pipeline—lossless and platform-neutral.

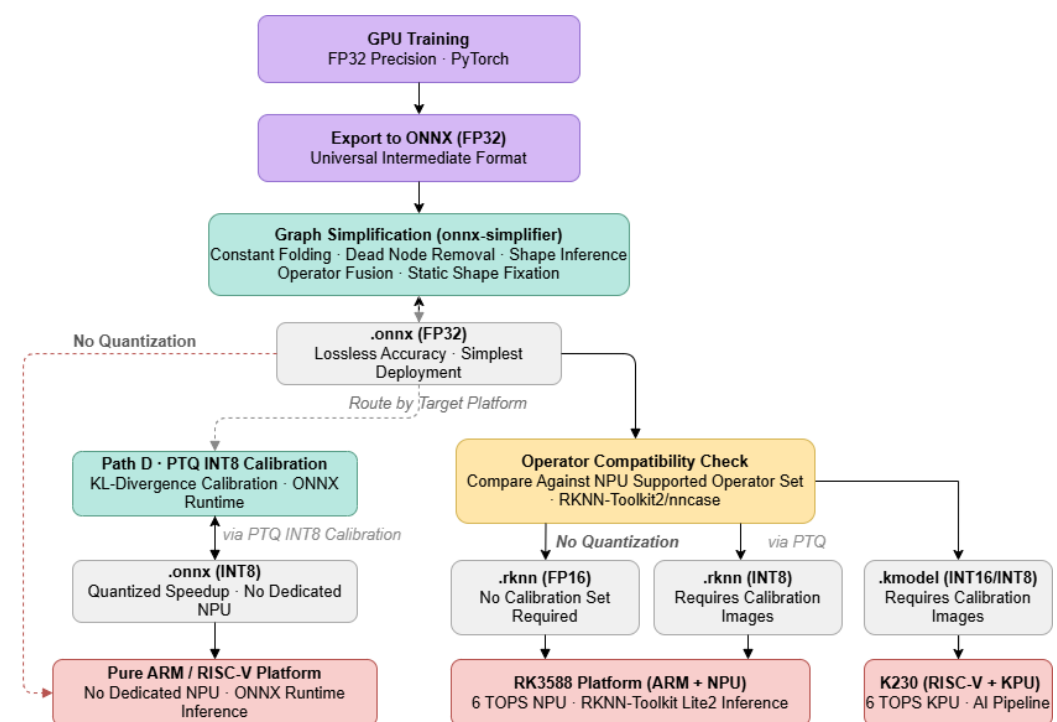


Figure 5. Inference-aware deployment pipeline.

Routing by target hardware, not by precision. From the anchor the pipeline forks on hardware first, because operator compatibility is hardware-conditional and quantization applied beforehand could either be wasted or produce nodes the target compiler cannot ingest. Three branches are instantiated:

- CPU-only branch—Raspberry Pi 4B (ARM Cortex-A72) and LicheePi 4A (RISC-V C910), both running the ONNX Runtime framework. With no accelerator present, the only optimization axis is precision: the FP32 graph is run as-is (lossless reference), or KL-divergence-calibrated to .onnx (INT8). INT8 speedup here comes from single-instruction multiple-data (SIMD) integer throughput (ARM NEON/RISC-V Vector, RVV) rather than a dedicated MAC array, and is therefore bounded.
- NPU branch—Rockchip RK3588 (LubanCat-4). The first action is not quantization but an operator-compatibility check against the RKNN-Toolkit2/nncase supported-operator set. Nodes outside the set are flagged for CPU fallback at compile time, breaking the NPU's static schedule. By construction, every node in eBirdNet-Nano clears

this gate. The compatibility-checked graph is then compiled either directly to `.rknn` (FP16)/`.kmodel` (no calibration) or with KL-divergence calibration to `.rknn`/`.kmodel` (INT8); both are loaded at runtime by RKNN-Toolkit Lite2 in single-core mode.

- KPU branch—Kendryte K230 (dual-core T-Head C908 RISC-V with a 6-TOPS KPU). Similar to the NPU branch, the anchor `.onnx` graph is first run through an operator-compatibility check against the nncase compiler’s supported operator set. Once compatibility is verified, the graph is compiled into a quantized `.kmodel` binary under either INT16 or INT8 precision. This compilation is driven by PTQ using the representative calibration dataset, ensuring zero-fallback, hardware-accelerated execution natively on the Canaan KPU.

4. Experimental Setup and Evaluation Metrics

4.1. Datasets and Training Settings

4.1.1. Dataset Details

We use two datasets that play complementary roles. Macao-ebird-det is the in-domain benchmark on which the design decisions are made [30]. It covers 24 protected and endangered species from the Macao region, with 7287 detection images split into training, validation and test partitions in an approximately 70/20/10 ratio. The annotation pipeline combines a Grounding DINO open-vocabulary detector [47], used to produce candidate bounding boxes from species-named text prompts with a manual verification pass. The dataset has been released publicly under a non-commercial research license to support reproduction.

The annotation pipeline of Macao-ebird-det is a single-stream protocol in which Grounding DINO produces candidate bounding boxes from species-named text prompts and trained annotators with ornithological background then reviews, refines, or rejects these candidates using Labellmg in batched per-species verification passes. The full protocol and the verifier qualifications are documented in the companion paper [30], which reports that 54 of 7341 candidate images were rejected during manual verification, yielding the 7287-image set used here at a labeling success rate of 99.18%. Direct evidence that residual annotation noise on the held-out test split does not materially affect the figures reported here is provided by the cross-model consistency: inter-architecture variance across nano-scale YOLO baselines is small (mAP@0.5 within 0.972–0.977, precision–recall gaps below 0.02), and the wider ten-model sweep reported in [30] reaches the same conclusion. See [30] for full details.

4.1.2. Training Settings

All models are trained with PyTorch 2.5.0 and CUDA 11.2 on a workstation equipped with two NVIDIA A10 GPUs (16 GB VRAM each) and an Intel Xeon Gold 6326 CPU. For the YOLO-series baselines, we use distributed data parallel training, automatic mixed precision, a batch size of 16, and pretrained YOLOv12n weights as initialization. The training schedule comprises 300 epochs with early stopping at a patience of 100. Input images are resized to 640×640 during training. Augmentations follow the YOLOv12 default recipe (Mosaic with probability 1.0 for the first 290 epochs, HSV jitter, horizontal flipping, scaling, and translation). We deliberately keep the augmentation policy fixed across all models so that any difference in the comparison tables is attributable to architecture rather than to training.

The seven non-YOLO baselines reported in Section 5.2 are trained separately under the TensorFlow 2.x Object Detection application programming interface (API), each initialized from its official COCO-pretrained checkpoint and fine-tuned on Macao-ebird-det using the model’s default pipeline configuration. Because each baseline retains its own

optimizer, schedule, and augmentation policy, these models serve as cross-framework accuracy references rather than as a controlled architectural comparison.

4.2. Deployment Platforms

The four platforms used in this evaluation were chosen to differ in architecture. Each represents an execution model a real deployment would actually have to pick between, as shown in Table 2. The Rockchip RK3588, on a LubanCat-4 board (EmbedFire, Dongguan, China), is the ARM-plus-NPU case: an eight-core ARM SoC ($4 \times$ Cortex-A76 at 2.4 GHz, $4 \times$ Cortex-A55 at 1.8 GHz) paired with a dedicated 6-TOPS NPU that runs INT8 and FP16 inference natively, an order of magnitude beyond what the ARM cores alone deliver on the same chip. On this board the model is compiled to the `.rknn` format and executed through RKNN-Toolkit2 v2.0. The Kendryte K230, on a CanMV-K230 board (01Studio, Shenzhen, China), is the RISC-V-plus-knowledge processing unit (KPU) case: a dual-core T-Head C908 at 1.6 GHz paired with a Cannaan KPU, a vector-digital signal processor (DSP)-class hardware accelerator with native 16-bit integer (INT16)/INT8 support. The model is compiled to `.kmodel` and executed through nncase-2.9.0. The Raspberry Pi 4B (quad-core Cortex-A72 at 1.5 GHz) (Raspberry Pi Foundation, Cambridge, UK) is the pure ARM-CPU case, a mature platform with nothing in the inference path but the CPU itself. The LicheePi 4A (quad-core Alibaba T-Head C910 at 1.85 GHz, TH1520 SoC) (Sipeed, Shenzhen, China) is the pure RISC-V case, which takes the comparison off the ARM instruction set entirely. Its TH1520 does carry a general-purpose neural network accelerator (NNA), but we leave it disabled on purpose: the point of including this board is to see what a RISC-V CPU does on its own, without the result being colored by how mature a second accelerator's toolchain happens to be. Both CPU-only boards run the model in the `.onnx` format through ONNX Runtime 1.14.1, the same stack on each, so that the ARM-versus-RISC-V comparison is not confounded by a change of runtime.

Table 2. Hardware specifications of the four deployment platforms.

Configuration	RK3588 (LubanCat-4)	K230 (01Studio CanMV-K230)	Raspberry Pi 4B	LicheePi 4A
Architecture	ARM + NPU	RISC-V + KPU	ARM (no dedicated NPU)	RISC-V (NPU not used)
CPU cores	Cortex-A76 $\times 4$ @ 2.4 GHz + Cortex-A55 $\times 4$ @ 1.8 GHz	T-Head C908 $\times 2$ (1 @ 1.6 GHz + 1 @ 0.8 GHz)	Cortex-A72 $\times 4$ @ 1.5 GHz	T-Head C910 $\times 4$ @ 1.85 GHz
RAM	8 GB LPDDR4	2 GB LPDDR4	4 GB LPDDR4	8 GB LPDDR4
AI accelerator	Rockchip NPU, 6 TOPS	Kendryte KPU, 6 TOPS @ INT8	—	TH1520 NNA, 4 TOPS @ INT8 (not used) [†]
Inference framework	RKNN-Toolkit2 v2.0	nncase-2.9.0	ONNX Runtime 1.14.1	ONNX Runtime 1.14.1
Model format	<code>.rknn</code> (FP16/INT8)	<code>.kmodel</code> (INT16/INT8)	<code>.onnx</code> (FP32/INT8)	<code>.onnx</code> (FP32/INT8)

[†] The TH1520 NNA is intentionally not used in this study; LicheePi 4A is evaluated through its RISC-V CPU with ONNX Runtime to isolate the ARM-versus-RISC-V CPU comparison without introducing accelerator-toolchain effects.

4.3. Metrics

4.3.1. Detection Accuracy

Detection accuracy is reported in terms of Precision, Recall, mean average precision at an IoU threshold of 0.5 (mAP@0.5), and mAP@0.5:0.95, evaluated on the held-out test

split in FP32 at 640×640 . For each model and precision mode, we record $N = 100$ frames after a 10-frame thermal warm-up. Let $\{t_1^{\text{inf}}, t_2^{\text{inf}}, \dots, t_N^{\text{inf}}\}$ denote the observed pure inference times (CPU/NPU forward pass only), and let $\{t_1, t_2, \dots, t_N\}$ denote the corresponding total-pipeline times, with each t_i decomposed into pre-processing, inference, and post-processing components.

4.3.2. Latency and Throughput

- Mean inference latency, the arithmetic mean of pure inference time across the measurement window:

$$\bar{t}^{\text{inf}} = \frac{1}{N} \sum_{i=1}^N t_i^{\text{inf}}. \quad (5)$$

- Mean total latency, the arithmetic mean of end-to-end frame time, including pre-processing, inference, and post-processing:

$$\bar{t} = \frac{1}{N} \sum_{i=1}^N t_i. \quad (6)$$

- P90 latency, the 90th percentile of $\{t_i\}$, used as a tail-behavior summary:

$$t_{\text{P90}} = \text{Percentile}_{90}(\{t_i\}). \quad (7)$$

- Jitter, the standard deviation of the total-pipeline distribution:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (t_i - \bar{t})^2}. \quad (8)$$

- Stab (%), the fraction of frames whose total-pipeline time falls within $1.2 \times$ the per-model median $\bar{t}$ —a normalized concentration measure that we found more informative than raw standard deviation when comparing models with very different median latencies:

$$\text{Stab}_{\%} = \frac{|\{i : t_i \leq 1.2 \bar{t}\}|}{N} \times 100\%. \quad (9)$$

- End-to-end frame rate, computed from the mean total-pipeline time, where $\bar{t}_{\text{ms}}$ denotes $\bar{t}$ expressed in milliseconds:

$$\text{FPS} = \frac{1000}{\bar{t}_{\text{ms}}}. \quad (10)$$

4.3.3. Power and Energy

Power and energy are measured with an FNB58 Universal Serial Bus Type-C (USB-C) inline meter sampling at 1 Hz on the supply rail of the RK3588 board. The first 10% of frames in each 5000-frame run are discarded to skip the thermal ramp-up, and the remaining window ($T \geq 120$ s) is used for steady-state averaging.

Over this $T \geq 120$ s window, rather than relying on instantaneous power samples—whose effective sampling rate is too coarse to resolve the ms-level per-frame inference workload—we record the meter's cumulative energy reading E_{cum} (in Wh) and compute the steady-state average power by integration:

$$P_{\text{inf}} = \frac{E_{\text{cum}} \cdot 3600}{T} \quad (\text{W}), \quad (11)$$

where the factor 3600 converts watt-hours to joules. The same protocol is applied identically to two workloads: with the board running the detector in steady state we obtain P_{inf} , and

with the board idle (no inference scheduled) we obtain P_{idle} . Both quantities are therefore time-integrated averages over $T \geq 120$ s rather than instantaneous samples.

- Inference-induced net power, the workload-dependent component of board power with the idle baseline subtracted:

$$\Delta P = P_{\text{inf}} - P_{\text{idle}} \quad (\text{W}). \quad (12)$$

- Energy per frame, the electrical energy consumed to process one frame; the figure of merit for battery-bounded operation:

$$E_{\text{frame}} = \frac{P_{\text{inf}}}{\text{FPS}} \times 1000 \quad (\text{mJ/frame}). \quad (13)$$

- Energy efficiency, throughput delivered per watt; the figure of merit under a fixed power budget:

$$\eta = \frac{\text{FPS}}{P_{\text{inf}}} \quad (\text{FPS/W}). \quad (14)$$

5. Experimental Results and Analysis

5.1. Training Convergence

To verify that the proposed configuration yields a properly converged model, we examine the loss and mAP trajectories on the validation set throughout the entire 300-epoch schedule. Figure 6 shows the validation dynamics of eBirdNet-Nano over 300 training epochs. As illustrated in Figure 6a, the three validation loss components (Val Box Loss, Val Cls Loss, and Val DFL Loss) decrease steadily and stabilize after approximately epoch 100, without any late-stage rebound. Meanwhile, the validation mAP curves in Figure 6b rise rapidly during the early epochs and gradually approach a plateau after around epoch 200. The standard deviation of mAP@0.5 over the final 50 epochs is only 7×10^{-4} , indicating highly stable performance. Collectively, these observations demonstrate that eBirdNet-Nano achieves stable convergence within the 300-epoch training process. The same training schedule and optimization hyper-parameters described in Section 4.1.2 are applied identically to all YOLO series. For conciseness, the loss and mAP trajectories of the baseline models—which exhibit qualitatively similar convergence behavior—are not reproduced here.

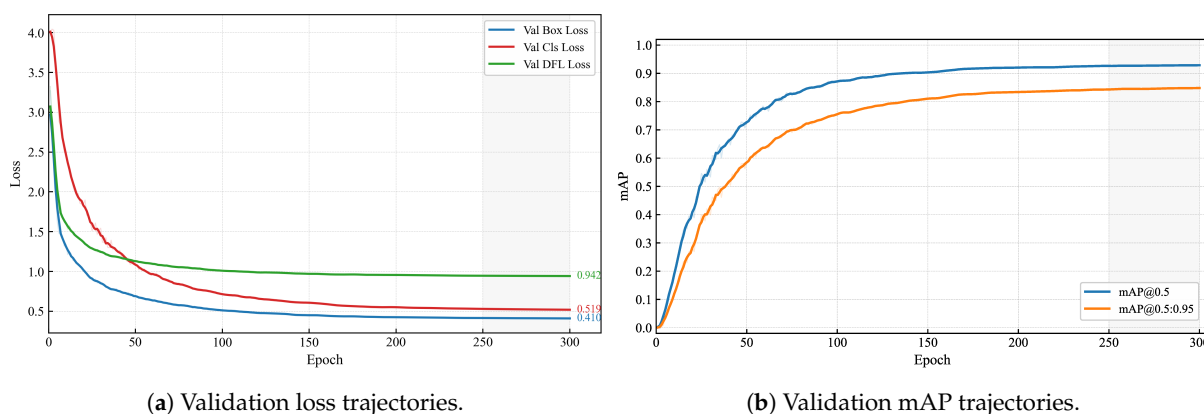


Figure 6. Validation dynamics of eBirdNet-Nano over 300 training epochs: (a) validation loss trajectories including box, classification, and distribution focal loss; (b) validation mAP trajectories for mAP@0.5 and mAP@0.5:0.95. The grey shaded area indicates the epoch-wise variability band around the smoothed validation curves.

To further evaluate the practical implications of the lightweight design's mAP reduction, Figure 7 presents the normalized confusion matrix of eBirdNet-Nano on the Macaoebird-det validation set. The matrix demonstrates that errors are not distributed randomly but are strongly tied to morphological similarity. For instance, mutual misclassification is notably high (21–23%) between *Platalea minor* and *Platalea leucorodia*—two spoonbill species that share nearly identical plumage and gross morphology, differing mainly in subtle facial features. Similarly, raptor species such as *Milvus migrans* show elevated confusion rates (up to 16%) with other birds of prey like *Circus spilonotus*.

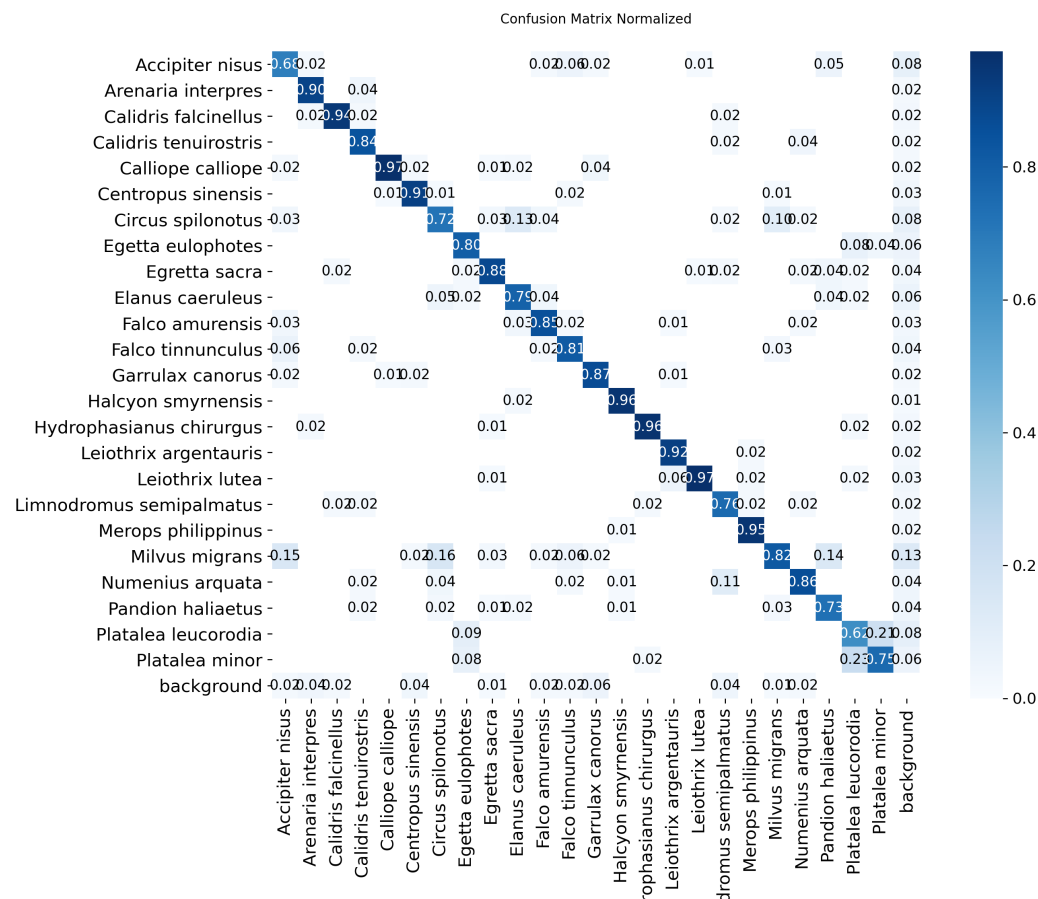


Figure 7. The normalized confusion matrix of eBirdNet-Nano on the Macaoebird-det validation set.

Crucially, missed detections (background misclassifications) remain low (≤ 0.08) for flagship species. In field patrols, the edge terminal primarily serves for data triage—filtering empty footage. Because it reliably logs bird presence, species-level confusions can be easily corrected during offline expert review. eBirdNet-Nano thus preserves the essential foreground–background separation required for a deployable conservation tool.

5.2. Detection Accuracy on Macao-Ebird-Det

Table 3 places eBirdNet-Nano alongside five contemporary nano-scale YOLO variants and seven representative non-YOLO state-of-the-art detectors on Macaoebird-det. The non-YOLO group spans the three main detection paradigms: anchor-free single-stage (EfficientDet-D0/D1, CenterNet-ResNet50-V2), anchor-based single-stage (SSD with MobileNet-V1-FPN, MobileNet-V2-FPNlite, and ResNet50-V1-FPN backbones), and anchor-based two-stage (Faster R-CNN with ResNet50-V1 backbone).

Table 3. Detection performance on Macao-ebird-det (FP32, 640 × 640).

Model	Params (M)	GFLOPs	Precision	Recall	mAP@0.5	mAP@0.5:0.95
YOLOv8n [11]	3.01	8.1	0.951	0.943	0.975	0.939
YOLOv9t [12]	1.98	7.6	0.954	0.938	0.977	0.945
YOLOv10n [13]	2.73	8.3	0.957	0.941	0.977	0.945
YOLOv11n [14]	2.59	6.3	0.951	0.938	0.972	0.938
YOLOv12n [15]	2.56	6.3	0.942	0.936	0.972	0.936
EfficientDet-D0 [48]	3.90	2.54	—	—	0.932	0.863
EfficientDet-D1 [48]	6.60	6.60	—	—	0.943	0.889
SSD-MobileNet-V1-FPN [22,49]	4.96	6.51	—	—	0.957	0.887
SSD-MobileNet-V2-FPNlite [49,50]	4.13	3.43	—	—	0.929	0.851
SSD-ResNet50-V1-FPN [49,51]	31.93	162.83	—	—	0.932	0.866
CenterNet-ResNet50-V2 [51,52]	29.83	87.05	—	—	0.652	0.705
Faster R-CNN-ResNet50-V1 [51,53]	28.32	210.35	—	—	0.965	0.874
eBirdNet-Nano (ours)	1.05	5.8	0.895	0.868	0.929	0.848

Note: Bold indicates the lowest parameter count in the table. Precision and Recall are unavailable for the non-YOLO baselines as the Tensorflow Model Zoo released checkpoints do not include those metrics in standard evaluation logs; mAP@0.5 and mAP@0.5:0.95 carry the same accuracy signal aggregated over thresholds and are sufficient for the cross-family accuracy comparison.

The transformation reduces the parameter count by 59% and GFLOPs by 8% relative to YOLOv12n, at a 4.3-percentage-point cost in mAP@0.5 (0.972 to 0.929). The accuracy gap to YOLOv8n through YOLOv11n is similar in magnitude—those models reach 0.972–0.977 on mAP@0.5—but they carry $1.9\times$ to $2.9\times$ more parameters and continue to depend on the same dynamic-shape operators that triggered the deployment problem in the first place. This compute reduction further establishes eBirdNet-Nano as the lowest-GFLOPs entry in the entire YOLO group: at 5.8 GFLOPs it runs 24% below YOLOv9t (7.6), the parameter-closest YOLO variant, and 28–30% below YOLOv8n and YOLOv10n (8.1 and 8.3 GFLOPs, respectively). Against YOLOv9t specifically, eBirdNet-Nano delivers a further 47% parameter reduction and 24% compute reduction while keeping mAP@0.5 within 4.8 percentage points, demonstrating that the proposed operator-level redesign extracts additional efficiency beyond what the latest nano YOLO variants already provide. For practical bird-detection driven by object-level detection (mAP@0.5), eBirdNet-Nano’s 0.929—at the smallest parameter count and lowest GFLOPs in the group—meets the requirements of typical field deployments.

The cross-paradigm picture is consistent. eBirdNet-Nano holds the smallest parameter count by a substantial margin: $3.9\times$ fewer parameters than the closest lightweight non-YOLO baseline (SSD-MobileNet-V2-FPNlite, 4.13 M), $3.7\times$ fewer than EfficientDet-D0 (3.90 M), and 27–30 $\times$ fewer than the heavier Faster R-CNN, CenterNet, and SSD-ResNet50 variants. Against SSD-MobileNet-V2-FPNlite specifically—the matched-accuracy lightweight comparator—mAP@0.5 is identical (both 0.929) and mAP@0.5:0.95 differs by only 0.3 percentage points (0.848 vs. 0.851) at roughly a quarter of the parameters. The 3.6-percentage-point mAP@0.5 gap to the strongest baseline, Faster R-CNN (0.965), is exchanged for a $27\times$ parameter penalty and roughly $36\times$ higher GFLOPs (210.35 vs. 5.8), an exchange that is inconsistent with the embedded-NPU deployment scenario targeted by this work.

A note on baseline choice. We retain YOLOv12n as the architectural baseline because it is the dynamic-shape operators introduced in v12 that motivate the entire transformation, and the M0–M3 ablation in Section 5.3 is most cleanly read against that starting point. We additionally include YOLOv8n and YOLOv11n in the cross-model latency comparison (Section 5.4.1) because both are themselves NPU-friendly, which provides an independent reference point for what “no transformation needed” looks like on the same hardware.

5.3. Three-Level Ablation

Table 4 isolates the contribution of each transformation level on Macao-ebird-det. In the analyses that follow we refer to the four cumulative configurations as M0 (unmodified YOLOv12n baseline), M1 (operator replacement only, NPU operator replacement (NPU-OR)), M2 (operator replacement plus channel-and-depth compression, NPU-OR + CDC) and M3 (the full eBirdNet-Nano, NPU-OR + CDC + head equalization (HE)). Each subsequent row enables one additional transformation level on top of the previous row.

Table 4. Four-configuration ablation on Macao-ebird-det (FP32, 640×640).

Config	Params (M)	GFLOPs	mAP@0.5	mAP@0.5:0.95
M0 (YOLOv12n)	2.56	6.3	0.972	0.936
M1 (NPU-OR)	2.41	7.4	0.943	0.880
M2 (NPU-OR + CDC)	1.36	6.2	0.930	0.849
M3 (eBirdNet-Nano, +HE)	1.05	5.8	0.929	0.848

The three transitions tell three different stories.

M0 $\rightarrow$ M1 introduces the operator-level substitutions—SiLU $\rightarrow$ ReLU6, A2C2f $\rightarrow$ NPU-C3k2, C2PSA $\rightarrow$ NPU-SE-Block—and the parameter count drops by only 5.9%, while GFLOPs rise by 17% (6.3 $\rightarrow$ 7.4). The reason is that dense convolutions produce a more structured computational graph than sparse attention at the cost of slightly higher theoretical complexity. Mean average precision (mAP) falls by 2.9 percentage points at the IoU = 0.5 threshold. Read against the $1.78\times$ FP16 end-to-end latency improvement ($2.10\times$ on NPU-only inference time) (Section 5.4.1), the trade-off is the central methodological observation of the paper: a 5.9% parameter reduction together with an increase in FLOPs nonetheless yields the largest stage-wise speedup, because the speedup is dominated by the elimination of CPU fallback rather than by arithmetic compression.

M1 $\rightarrow$ M2 applies channel and depth compression. The parameter count falls sharply by 44%, from 2.41 M to 1.36 M, while GFLOPs settle to 6.2 and mAP@0.5 moves only from 0.943 to 0.930. The asymmetry indicates that the original model carries substantial redundancy for the fine-grained recognition of 24 morphologically distinct species; a deeper backbone offers no meaningful accuracy gain on this dataset while raising the risk of overfitting on the limited training set.

M2 $\rightarrow$ M3 rebalances the detection head. Parameters drop by a further 23% and GFLOPs settle at 5.8. Accuracy at mAP@0.5 falls by 0.001 and mAP@0.5:0.95 holds at 0.848. The interpretation is that for distant, small bird targets the difficulty across the P3, P4 and P5 scales is comparable, the unbalanced [256, 512, 512] allocation brings no differentiated accuracy gain on this task, and balanced compression to 64 channels removes redundant parameters at essentially no cost in accuracy.

5.4. Inference Latency on the RK3588 NPU

5.4.1. Cross-Model Latency Benchmark on RK3588

Table 5 places M3 in absolute terms by comparing it against five contemporary nano-scale YOLO detectors. Each model is exported through RKNN-Toolkit2 v2.0 and benchmarked at 320×320 on a single RK3588 NPU core.

Parameter efficiency. A useful derived quantity is INT8 throughput per million parameters. eBirdNet-Nano (M3) reaches 35.62 FPS/M, followed by YOLOv12n at 13.83, YOLOv8n at 13.22, YOLOv11n at 12.66, YOLOv9t at 12.42 and YOLOv10n at 9.82. The $2.58\times$ lead over the runner-up is the table's sharpest single result. The ratio matters because the parameter count maps almost linearly to on-chip SRAM footprint and DRAM bandwidth on heterogeneous SoCs.

Table 5. Cross-model NPU inference latency on RK3588 (LubanCat-4), 320×320 .

Model	Params (M)	FPS	INT8 @ 320				FPS	FP16 @ 320			
			Inf (ms)	Total (ms)	Stab (%)	Jit (ms)		Inf (ms)	Total (ms)	Stab (%)	Jit (ms)
YOLOv8n	3.01	39.8	12.93	25.11	83.4	10.20	42.3	17.60	23.62	90.3	8.25
YOLOv9t	1.98	24.6	26.03	40.58	73.0	15.51	14.8	53.74	67.57	85.9	28.63
YOLOv10n	2.73	26.8	29.24	37.38	86.5	14.49	15.2	57.01	65.85	87.1	26.08
YOLOv11n	2.59	32.8	18.57	30.47	83.5	9.59	16.4	45.17	61.06	74.7	35.88
YOLOv12n (M0)	2.56	35.4	14.47	28.22	87.6	7.24	10.6	78.94	94.20	85.0	52.65
eBirdNet-Nano (M3)	1.05	37.4	13.83	26.76	89.5	8.57	31.5	24.29	31.71	64.8	22.99

Note: Inf = NPU inference time. Jit = Jitter.

Latency stability. Under INT8, M3 reaches the highest Stab% in the table (89.5%, 1.9 percentage points above YOLOv12n). The structural explanation comes from utilization: M3, YOLOv8n and YOLOv12n all spend roughly 51% of total frame time on the NPU itself, indicating that none of their operators fall back to the CPU. Jitter is dominated by CPU–NPU dispatch overhead, and a fully NPU-resident graph removes most of that overhead. The contrast with FP16 is sharp. Under FP16, YOLOv12n’s NPU latency balloons from 14.47 ms to 78.94 ms because at FP16 the dynamic-shape operators that were folded into INT8 calibration are no longer absorbed; M3, in contrast, holds at 24.29 ms.

Latency–parameter trade-off. In the design space defined by NPU inference latency and parameter count, M3 occupies the lower-left corner under both precisions—a region that no YOLO baseline reaches. At INT8, M3 runs at 13.83 ms with 1.05 M parameters, 0.90 ms behind YOLOv8n’s 12.93 ms (which carries $2.87 \times$ more parameters) and 0.64 ms ahead of YOLOv12n. At FP16, the gap to YOLOv8n widens to 6.7 ms (38%), but the more telling number is the M0 $\rightarrow$ M3 reduction: FP16 NPU latency drops from 78.94 ms to 24.29 ms, a $3.25 \times$ improvement.

Two caveats need to be stated explicitly. First, YOLOv8n is a stronger NPU-side competitor than YOLOv12n; treating it (and YOLOv11n) as separate references rather than as the primary baseline is the right way to read the speedup figures we report. Second, the conclusion that M3 sits at the Pareto-optimal corner is tied to the operator-support matrix of RKNN-Toolkit2 v2.0; a future toolchain that adds dynamic-shape support would shift the boundary, and the conclusion in this paper should be read with that toolchain version explicit.

5.4.2. Stage-Wise NPU Latency and Quantization Decomposition

The acceleration factors quoted in the Abstract are derived from per-stage FP16 NPU measurements on the same RK3588 board. Table 6 reports NPU inference time, end-to-end total time, FPS, latency stability, and jitter for each of the four ablation configurations M0–M3 at 320×320 .

Table 6. Stage-wise NPU inference behavior on RK3588 (LubanCat-4) for the four-configuration ablation. FP16 precision, 320×320 input, single NPU core, mean values over 100 frames after a 10-frame warm-up. Speedup is computed against M0 within the same column.

Config	Params (M)	GFLOPs	Inf (ms)	Total (ms)	FPS	Stab (%)	Jit (ms)	Inf $\times$ vs. M0
M0 (YOLOv12n)	2.56	6.3	78.94	94.20	10.6	85.0	52.65	1.00 $\times$
M1 (NPU-OR)	2.41	7.4	37.63	53.03	18.9	65.2	25.44	2.10 $\times$
M2 (NPU-OR + CDC)	1.36	6.2	19.62	25.53	39.2	68.3	10.06	4.02 $\times$
M3 (eBirdNet-Nano)	1.05	5.8	24.29	31.71	31.5	64.8	22.99	3.25 $\times$

Stage-wise architectural dynamics. Three observations emerge from the FP16 ablation: First, the operator-replacement step alone M_0 to M_1 delivers a $2.10\times$ NPU speedup (78.94 to 37.63 ms) despite a 17% increase in theoretical FLOPs. This confirms that relocating the SiLU, A2C2f, and C2PSA nodes from the CPU back to the NPU eliminates a dispatch overhead that heavily outweighs the arithmetic cost of denser convolutions. Second, the remaining stages compound on a fully NPU-resident graph and behave as standard arithmetic compression predicts. M_1 to M_2 delivers a further $1.92\times$ speedup with a 44% parameter cut, while M_2 to M_3 trades a marginal amount of pure NPU throughput ($0.81\times$) for crucial stability improvements. The cumulative FP16 NPU speedup from M_0 to M_3 is $3.25\times$. Third, latency stability is non-monotonic: dropping from 85.0% to 65.2% during intermediate compression (M_1 , M_2), but restored to a robust operating point by the final head-equalization step in M_3 (rising to 89.5% under INT8, as shown in Table 5).

Disentangling architectural and quantization speedups. Cross-referencing the baseline (M_0) and eBirdNet-Nano (M_3) across precisions in Table 5 isolates the source of the final acceleration. The end-to-end speedup from M_0 -FP16 to M_3 -INT8 decomposes into a dominant architectural factor ($2.97\times$) and a minor quantization factor ($1.19\times$), whose product ($2.97 \times 1.19 \approx 3.54\times$) matches the measured $3.53\times$ total gain. Conversely, the baseline's massive $3.34\times$ quantization speedup merely reflects its inefficient FP16 starting point burdened by CPU fallbacks. Even after INT8 calibration, the baseline's pure NPU time (14.47 ms) trails eBirdNet-Nano (13.83 ms), confirming that operator-aware redesign fundamentally dictates the latency floor.

Per-level attribution summary. Reading these transitions together reveals distinct deployment roles across the optimization stack: operator substitution ($M_0 \rightarrow M_1$) acts as the primary latency lever by eliminating CPU fallbacks; module compression ($M_1 \rightarrow M_2$) provides parameter economy (44% reduction) with minimal accuracy loss; and head rebalancing ($M_2 \rightarrow M_3$) restores latency stability. Ultimately, structural NPU compliance governs edge real-time performance far more than numerical precision scaling.

5.4.3. Board-Level Power and Per-Frame Energy

For a battery-powered field terminal, however, latency alone is not sufficient: the same workload must also fit within an energy envelope that determines how long the device can run between charges. This section therefore reports steady-state power draw and per-frame energy consumption on the same hardware, measured through the cumulative energy integration protocol of Section 4.3. Table 7 summarizes the results under INT8 at both deployment resolutions (320×320 and 640×640).

Table 7. Power and energy consumption on the RK3588 (INT8). Hardware idle power $P_{\text{idle}} = 1.17$ W is identical across all conditions and omitted.

Method	Resolution	P_{inf} (W)	ΔP (W)	FPS	E_{frame} (mJ)	η (FPS/W)
M0 (YOLOv12n)	320×320	1.48	0.31	35.4	41.85	23.90
M3 (eBirdNet-Nano)	320×320	1.54	0.37	37.4	41.14	24.30
M0 (YOLOv12n)	640×640	1.85	0.68	7.8	237.80	4.21
M3 (eBirdNet-Nano)	640×640	1.62	0.45	14.7	110.18	9.08

Note: Bold indicates the best value within each resolution group.

Two patterns stand out. At 320×320 the two models land essentially on top of each other in efficiency: P_{inf} for eBirdNet-Nano is marginally higher (1.54 vs. 1.48 W), but its FPS is also higher (37.4 vs. 35.4), and the two effects cancel— E_{frame} differs by only 1.7% and η by 1.7%. Under INT8, both models execute entirely on the NPU at 320×320 with no CPU fallback, so the remaining gap at small resolution sits within measurement noise.

At 640×640 the picture changes sharply. The proposed model draws 12% less total power (1.62 vs. 1.85 W), its inference-induced net power ΔP drops by 34% (0.45 vs. 0.68 W), and FPS almost doubles (14.7 vs. 7.8). These effects compound: E_{frame} falls by 54% and η rises by $2.16\times$. The same mechanism identified in the latency analysis is at work here—the baseline’s dynamic-shape operators fall back to the CPU at 640×640 once the activation memory exceeds a certain threshold, and pulling the A76 big cores into the loop is both slower and considerably more power-hungry than keeping the NPU multiply–accumulate (MAC) array exclusively engaged. What appears in the latency comparison as a $1.9\times$ speedup shows up simultaneously as a 54% drop in per-frame energy.

5.5. Cross-Architecture Platform Comparison

Section 5.4.1 established eBirdNet-Nano on the RK3588 NPU and Section 5.4.2 isolated the contributions of architecture. This section moves the same model across architectures—NPU-accelerated RK3588, ARM-CPU Raspberry Pi 4B, RISC-V LicheePi 4A—to test whether the design choices generalize beyond their primary deployment target.

5.5.1. Protocol

All configurations are profiled at a deployment resolution of 320×320 . Baseline precisions match native hardware capabilities: FP16 for the RK3588 NPU, INT16 for the K230 KPU, and FP32 for the CPU-only boards (Raspberry Pi 4B and LicheePi 4A).

Crucially, the exact same calibration benchmark—comprising 200 stratified images randomly sampled across the 24 species from the Macao-ebird-det dataset—is utilized across all four heterogeneous platforms. This uniform calibration eliminates dataset bias, isolating compiler-side quantization mechanics. NPU/KPU measurements follow the benchmark setup described in Section 5.4, while CPU latencies are averaged over 100 frames after a 10-frame warm-up.

5.5.2. Results and Findings

Absolute throughput and cross-vendor generalization. Looking at the absolute throughput in Table 8, two observations establish the deployment reality. First, the directional lead holds universally: M3 produces lower total latency than M0 in every one of the eight platform–precision configurations. Second, the M3-INT8 endpoint reveals independent hardware gaps spanning an order of magnitude. The RK3588 NPU reaches 37.4 FPS at the top of the range. The K230 KPU reaches 10.2 FPS—accelerator-class but constrained by shared DRAM bandwidth between the image signal processor (ISP) and the knowledge processing unit (KPU) rather than KPU compute itself; critically, the M0 $\rightarrow$ M3 architectural step still transfers, with KPU-only inference dropping from 72.49 ms (M0) to 37.50 ms (M3)—a $1.93\times$ speedup that closely tracks the $2.10\times$ FP16 speedup measured on RK3588 (Table 6, M0 $\rightarrow$ M1), direct evidence that the architectural lever generalizes across NPU vendors and toolchains (nncase versus RKNN-Toolkit2). The Pi 4B reaches 17.7 FPS on mature NEON SIMD vectorization alone, and the LicheePi 4A reaches only 2.3 FPS, bounded by the maturity of the T-Head C910 RVV backend. The $2.1\times$ NPU-to-ARM gap (RK3588 $\rightarrow$ Pi 4B) quantifies what fixed-function silicon buys over mature ARM cores; the $7.7\times$ ARM-to-RISC-V gap (Pi 4B $\rightarrow$ LicheePi 4A) quantifies what a mature SIMD-vectorizing backend (NEON) buys over an early-generation one (T-Head C910 RVV). Platform selection within the edge AI class spans an order of magnitude in absolute capability.

Table 8. Cross-architecture inference comparison at 320×320 .

Platform	Precision	M0 (YOLOv12n)			M3 (eBirdNet-Nano)		
		Inf (ms)	Total (ms)	FPS	Inf (ms)	Total (ms)	FPS
RK3588 (RKNN)	FP16	78.94	94.20	10.6	24.29	31.71	31.5
	INT8	14.47	28.22	35.4	13.83	26.76	37.4
K230 (kmodel)	INT16	75.65	137.17	7.3	36.80	97.77	10.2
	INT8	72.49	133.58	7.5	37.50	98.29	10.2
Pi 4B (ONNX)	FP32	147.65	156.16	6.4	92.65	100.96	9.9
	INT8	70.36	78.34	12.8	48.50	56.45	17.7
LicheePi 4A (ONNX)	FP32	856.94	867.69	1.2	758.43	772.57	1.3
	INT8	729.53	742.29	1.3	430.26	443.74	2.3

Note: Inf = NPU/KPU/CPU inference time. Each platform is evaluated at two precisions: a baseline precision matching the platform's highest natively supported precision (FP16 for RK3588, INT16 for K230, FP32 for Pi 4B and LicheePi 4A), and INT8. Bold marks the per-platform best configuration.

Sources of the platform-specific speedup. While Table 8 shows that M3 is always faster, Figure 8 makes concrete that the source of that speedup shifts dramatically depending on the hardware architecture. As seen in Figure 8a, the architectural redesign yields a massive $2.97\times$ speedup on the RK3588 NPU at FP16—the largest single-step gain anywhere in the analysis. On the other three platforms the same step delivers a substantially narrower band of improvement: $1.36\times$ – $1.40\times$ on the K230 KPU, $1.39\times$ – $1.55\times$ on RPi 4B, and $1.12\times$ – $1.67\times$ on LicheePi 4A. This asymmetry proves our central hypothesis: the architectural redesign specifically targets and exploits a discrete inefficiency—operator fallback to the host CPU—that exists only on the RK3588 NPU compiler path. Once that specific bottleneck is replaced by something else—ISP-to-KPU bandwidth on K230, or raw FLOPs and SIMD throughput on a general-purpose CPU—the room for architectural speedup naturally shrinks.

The drop on the NPU side has its own structural reason. Section 5.4.2 already decomposed this within the RK3588, but only for M3 (a $1.19\times$ quantization gain). Figure 8b widens the view by adding the M0 column on the same NPU and the M0/M3 columns on the other three platforms. The RK3588 M0 quantization gain is $3.34\times$ —the largest in the figure—while M3 is only $1.19\times$. The RKNN INT8 toolchain independently restructures the operator graph during calibration, absorbing much of the operator-fallback inefficiency the architectural redesign was designed to remove; once absorbed, the room left for the architectural step shrinks to almost nothing. The two optimizations are partial substitutes on the NPU. K230 sits at the opposite extreme of the figure: both M0 and M3 quantization speedups collapse to near unity ($1.03\times$ / $0.99\times$), because INT8's arithmetic saving cannot translate into end-to-end speedup when the binding bottleneck is memory bandwidth rather than MAC density. On both CPU platforms the M0 and M3 quantization speedups stay close ($1.99\times$ / $1.79\times$ on the Pi; $1.17\times$ / $1.74\times$ on the LicheePi) and act on a bottleneck—single-instruction multiple-data multiply–accumulate (SIMD–MAC) density—the architectural step does not touch, so the two compose roughly multiplicatively.

The combined reading of Table 8 and Figure 8 translates into clear deployment heuristics. On an NPU (RK3588), the operator graph is the primary lever—Figure 8a shows it delivers the foundational gain that quantization alone cannot physically relocate. On a bandwidth-bound KPU (K230), the operator graph still applies but requires data-path engineering to translate the on-chip gain into end-to-end speedup, since INT8 quantization saves essentially nothing on its own (Figure 8b). On a mature ARM CPU (Pi 4B) where operator fallback is not the bottleneck, the architectural lever shrinks and INT8 quantization becomes the practical first step. On a first-generation RISC-V CPU (LicheePi 4A), absolute frame rate is bounded by the

maturity of the vectorizing backend rather than by the model—eBirdNet-Nano runs natively without modification, confirming cross-platform portability.

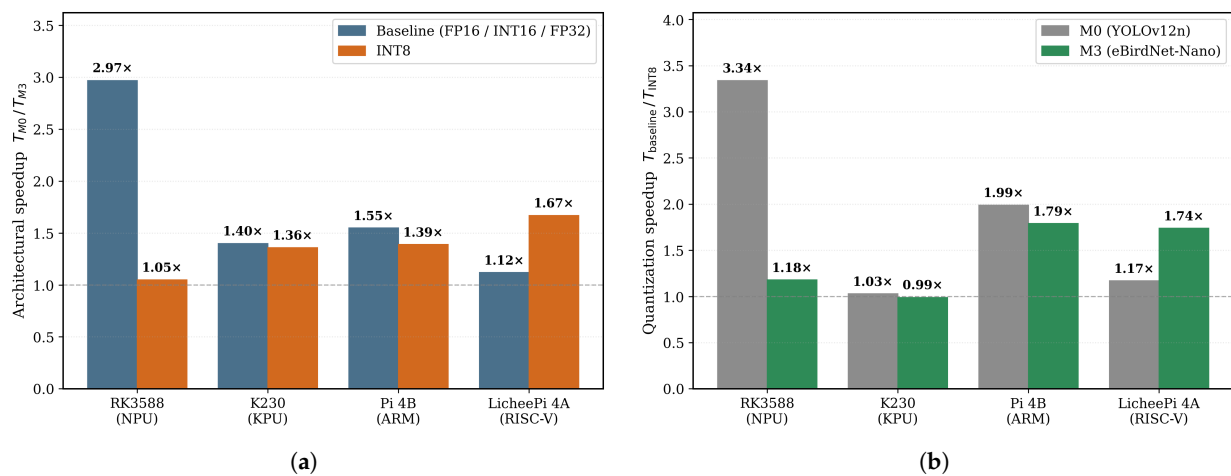


Figure 8. Speedup decomposition of eBirdNet-Nano across the four platforms, computed from the Total column of Table 8: (a) architectural speedup T_{M0}/T_{M3} at matched precision; (b) quantization speedup $T_{baseline}/T_{INT8}$ at matched model.

5.5.3. Deployed-Accuracy Implications Across Deployment Platform and Toolchains

Table 9 reveals that toolchain choice also affects per-class accuracy distribution, not only end-to-end latency.

Table 9. Per-class deployed mAP@0.5 of eBirdNet-Nano across the FP32 reference and three INT8 deployment toolchains.

Class	Instances (Inst.)	FP32 (PyTorch, PT)	INT8 ONNX (Pi 4b/LicheePi 4A)	INT8 kmodel (K230)	INT8 RKNN (RK3588)
All classes	1447	0.883	0.858	0.837	0.736
<i>Platalea leucorodia</i>	53	0.705	0.676	0.615	0.590
<i>Platalea minor</i>	53	0.710	0.604	0.692	0.548
<i>Limnodromus semipalmatus</i>	45	0.754	0.709	0.628	0.700
<i>Numenius arquata</i>	51	0.831	0.775	0.849	0.690
<i>Circus spilonotus</i>	82	0.833	0.764	0.762	0.679
<i>Falco amurensis</i>	55	0.846	0.848	0.797	0.709
<i>Falco tinnunculus</i>	54	0.854	0.837	0.847	0.750
<i>Calidris tenuirostris</i>	45	0.871	0.818	0.766	0.722
<i>Pandion haliaetus</i>	56	0.872	0.869	0.834	0.732
<i>Accipiter nisus</i>	62	0.875	0.845	0.824	0.700
<i>Calidris falcinellus</i>	50	0.877	0.841	0.826	0.793
<i>Milvus migrans</i>	77	0.884	0.838	0.772	0.746
<i>Elanus caeruleus</i>	61	0.889	0.897	0.842	0.721
<i>Merops philippinus</i>	62	0.911	0.895	0.881	0.798
<i>Garrulax canorus</i>	54	0.912	0.898	0.922	0.753
<i>Centropus sinensis</i>	56	0.916	0.928	0.828	0.676
<i>Arenaria interpres</i>	50	0.937	0.909	0.921	0.820
<i>Egretta eulophotes</i>	65	0.943	0.914	0.869	0.764
<i>Egretta sacra</i>	75	0.944	0.947	0.896	0.813
<i>Hydrophasianus chirurgus</i>	50	0.953	0.957	0.930	0.800
<i>Leiothrix argentauris</i>	71	0.953	0.924	0.896	0.718

Table 9. Cont.

Class	Instances (Inst.)	FP32 (PyTorch, PT)	INT8 ONNX (Pi 4b/LicheePi 4A)	INT8 kmodel (K230)	INT8 RKNN (RK3588)
<i>Leiothrix lutea</i>	76	0.972	0.947	0.938	0.765
<i>Halcyon smyrnensis</i>	67	0.977	0.975	0.974	0.925
<i>Calliope calliope</i>	77	0.979	0.968	0.975	0.760

Note: Bold indicates the aggregate all-class result. All measurements at 320×320 on the same Macao-ebird-det test split. FP32 (pt model) : PyTorch reference at deployment resolution.

Per-class evaluation shows that under ONNX Runtime hybrid INT8 and nncase INT8, the FP32 bottom-tier ranking is preserved with proportional drops. Under RKNN-Toolkit2 v2.0 INT8 (RK3588), the deployed accuracy distribution differs: eight species show per-class drops above 15 pp, including five (*Leiothrix argentea*, *Leiothrix lutea*, *Calliope calliope*, *Centropus sinensis*, *Garrulax canorus*) that the other two INT8 toolchains preserve at near-FP32 accuracy. The morphological-confusion bottom tier (*Platalea* pair, *Limnodromus semipalmatus*) is preserved across all four precisions. The instance-count vs. mAP@0.5 correlation is weakly positive at the FP32 reference and weakens further under RKNN INT8, indicating that no systematic bias exists toward rare species or less-frequent classes in either the morphological-confusion bottom tier or the toolchain-specific drops.

The residual gap therefore decomposes into a model-intrinsic morphological-similarity component (preserved across all precisions) and a toolchain-specific component that is recoverable through post-deployment toolchain choice: two of the three INT8 pipelines evaluated (ONNX Runtime hybrid INT8, nncase) preserve near-FP32 accuracy across the full class set, and the RKNN-specific drops on the affected passerine classes are addressable by selecting a finer-grained PTQ pipeline rather than by redesigning the model.

5.5.4. Multi-Platform Latency–Accuracy Trade-Off

To provide a comprehensive, on-device visualization of the hardware–software co-design space, Figure 9 plots the measured end-to-end throughput (FPS, log scale) against the actually-deployed mAP@0.5 across all four heterogeneous edge platforms. Crucially, rather than relying on theoretical offline FP32 accuracy proxies, this trade-off curve utilizes the empirical accuracies evaluated directly on the compiled quantized binaries running on the target hardware.

As illustrated in Figure 9, the proposed eBirdNet-Nano (M3, circles) consistently shifts to the right of the YOLOv12n baseline (M0, squares) across every single evaluated platform. This consistent horizontal shift represents a tangible, deployable throughput gain. On the high-performance Rockchip RK3588, M3 achieves the absolute highest throughput in the evaluation at 37.4 FPS. Although the RKNN compiler imposes a more pronounced quantization drop on M3 (0.736 mAP) compared to M0 (0.876 mAP) due to its strict integer quantization mapping of the customized blocks, M3 compensates by unlocking a full real-time frame rate.

On the resource-constrained Kendryte K230, M3 achieves a highly efficient 10.2 FPS (a $1.36\times$ speedup over M0's 7.5 FPS) while preserving a robust deployed accuracy of 0.837 mAP. On general-purpose CPU platforms (Pi 4B and LicheePi 4A), where operator fallbacks are not the binding bottleneck, M3's parameter-shrunk backbone still yields substantial throughput improvements (e.g., reaching 17.7 FPS compared to M0's 12.8 FPS on Pi 4B). This multi-platform trade-off analysis confirms that eBirdNet-Nano successfully pushes the Pareto-efficient boundary for low-power edge terminal deployment.

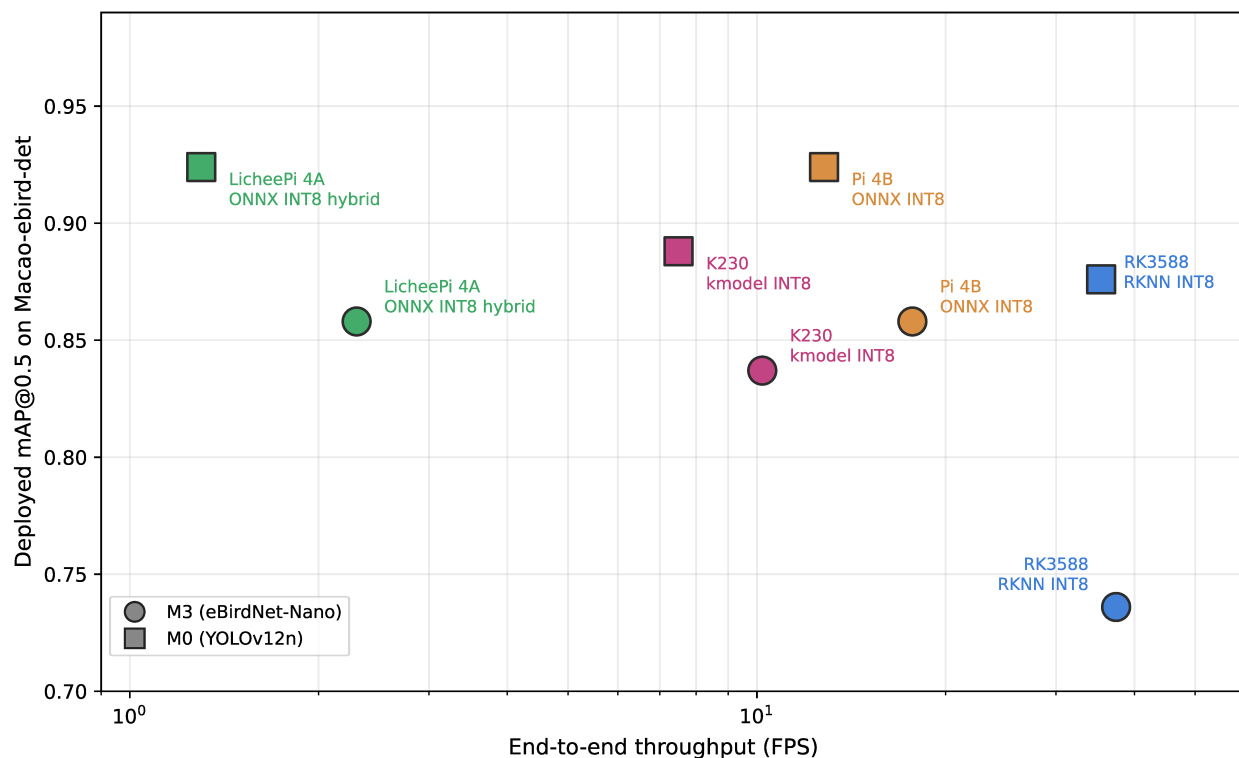


Figure 9. Latency–accuracy trade-off across deployment toolchains at 320×320 resolution.

5.6. System-Level Deployment on the EbirdEye Terminal

Sections 5.4.1–5.4.3 show that eBirdNet-Nano clears the latency, stability and energy budgets on a development board under a controlled benchmark loop. Field-deployment real-time, however, is a stronger claim: it requires the same throughput once the board is wrapped into a complete terminal with capture, display, multi-threading and a battery-bound power envelope. This section reports field-measured runtime on the EbirdEye terminal, an RK3588s-based observation node whose architecture is shown in Figure 10.

The terminal pairs the LubanCat-RK3588s board (RK3588s SoC, 4 GB LPDDR4, 32 GB eMMC, Debian 11) with three optical paths tuned for different observation ranges—a short-range MIPI-CSI module (OV8858, OmniVision Technologies, Inc., Hong Kong, China), a mid-range USB UVC camera (SPCA22381 + OV2735, Shenzhen Jinyang Century Technology Development Co., Ltd., Shenzhen, China), and a long-range CSI-plus-telescope adapter—together with a 5.5-inch MIPI touchscreen for on-site control, an HDMI 2.0 monitor output for external display, and a 4G LTE/global navigation satellite system (GNSS) MiniPCI-E module (EC20) for data uplink and time/location synchronisation. The terminal is powered by a 15,600 milliampere-hour (mAh)/5 V lithium battery with optional 20 W solar augmentation. The three optical paths share a common inference pipeline running eBirdNet-Nano through RKNN-Toolkit2 v2.0 in single-NPU-core mode; the field-runtime numbers reported below are measured on the mid-range UVC path, which is the configuration used most often during outdoor sessions. After a 10-frame warm-up and 30 s of cold-start transients, statistics were aggregated over a steady-state window of more than 120 s. The analysis covers two axes: a per-thread runtime breakdown under live capture, and a system-level power and endurance budget that ties laboratory throughput to deployable field runtime.

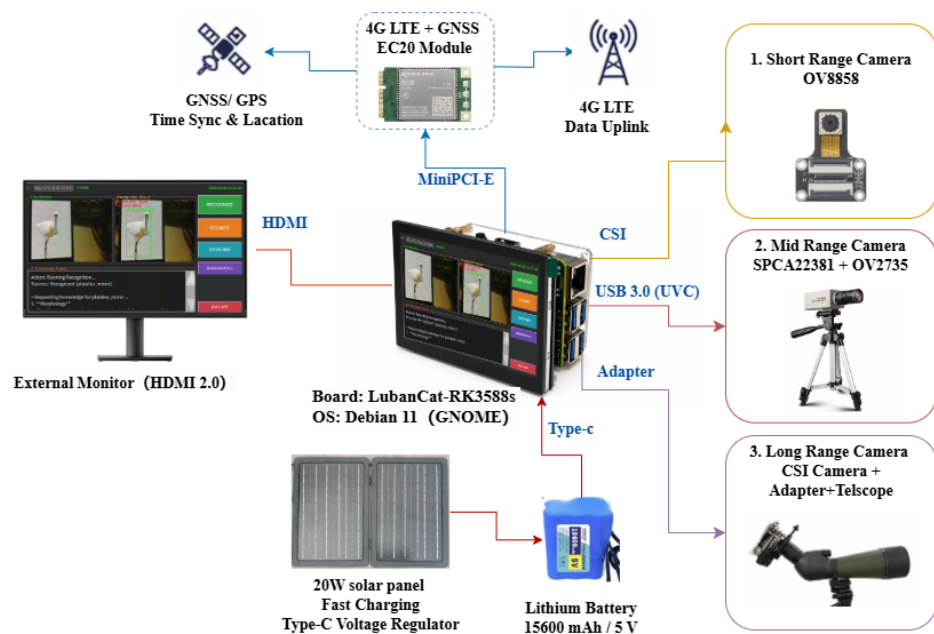


Figure 10. Hardware architecture of the EbirdEye terminal.

5.6.1. Per-Thread Runtime Behavior Under Live Capture

We first examine how the end-to-end frame time is realized at the thread level. In the deployed runtime the camera, inference and display threads advance independently and need not share a common cadence, so three frame rates and four latency components are tracked separately. The results, recorded on the LubanCat-RK3588s at 320×320 FP16 over a steady-state window of more than 120 s, are summarized in Table 10.

Table 10. Field-measured runtime statistics of the EbirdEye terminal (LubanCat-RK3588s, 320×320 , FP16; steady-state window > 120 s, mean values).

Metric	Value	Description
capFPS	~14.3 FPS	Camera frame acquisition rate.
infFPS	~42.3 FPS	Maximum inference throughput (no blocking).
dspFPS	~6.1 FPS	Display refresh rate limited by graphical user interface (GUI) rendering.
Latency (end-to-end)	~23.5 ms	Total time from frame capture to result output.
Pre-processing	~4.4 ms	Input preparation including resize and normalisation.
NPU inference	~18.3 ms	Forward pass execution on the NPU accelerator.
Post-processing	~1.0 ms	Decoding and NMS operations on CPU.

Several patterns emerge from these numbers. The inference path sustains about 42.3 FPS on average with an end-to-end latency of roughly 23.5 ms, of which 18.3 ms ($\approx 78\%$) is spent on the NPU itself, 4.4 ms on pre-processing and 1.0 ms on post-processing. These on-device numbers are directionally consistent with the bench result reported in Section 5.4.1 for the same model and precision (24.29 ms NPU mean/31.71 ms total at 320×320 FP16, configuration M3); the small residual gap reflects the difference between a controlled single-image benchmark loop and a live capture pipeline. The per-frame NPU latency fluctuates only within roughly ± 2 ms around its mean across the entire 120 s observation window, which confirms that the static-shape design of eBirdNet-Nano keeps the graph fully resident on the NPU and prevents CPU-side operator fallback or scheduling jitter from leaking into the steady state.

The three observed frame rates differ by roughly one order of magnitude, which is the expected signature of three decoupled processing stages rather than a sign of contention. The camera path is bounded by the configured image signal processor (ISP) cadence to

14.3 FPS on average; the inference path consumes the latest captured frame on demand and runs as fast as the NPU allows, reaching 42.3 FPS—about $3\times$ the capture rate, meaning a freshly captured frame is consumed well before the next one arrives, while inference never blocks acquisition; the display path is rate-limited to 6.1 FPS by the user interface (UI) refresh interval, kept deliberately low so that GUI repaint does not compete with inference for memory bandwidth. The relation $\text{infFPS} \gg \text{capFPS} \gg \text{dspFPS}$ is the operational evidence that perception, inference and rendering are genuinely decoupled. The average end-to-end latency of 23.5 ms sits well within the 33 ms (30 FPS) real-time threshold conventionally adopted for industrial vision pipelines, and the per-frame value never exceeds this threshold by a meaningful margin during the entire observation window. The “real-time” claim made for EbirdEye in this work is therefore a quantitative rather than nominal statement: it is measured in the field, on the deployed hardware.

5.6.2. System-Level Power and Endurance

Real-time inference is only useful in the field if the hardware that delivers it can run long enough to be useful. To complete the deployment profile, the total system power of the EbirdEye terminal—including the RK3588s board running eBirdNet-Nano at 320×320 FP16, the mid-range USB UVC camera and the 5.5-inch MIPI capacitive touchscreen—was recorded with the same FNB58 Universal Serial Bus Type-C (USB-C) inline meter under the same steady-state conditions as Table 7. The results are summarized in Table 11.

Table 11. System-level power and endurance of the EbirdEye terminal in field configuration (LubanCat-RK3588s, 320×320 FP16, eBirdNet-Nano).

Item	Value
Idle power P_{idle} (board only, no inference)	1.17 W
Total system power P_{total} (camera + screen + inference)	5.77 W
Battery capacity	15,600 mAh @ 5 V (78 Wh)
Continuous runtime (battery only)	~13.5 h
Solar charging power	~20 W

On the RK3588s, the total system power of 5.77 W under continuous inference yields about 13.5 h of autonomous runtime on a single charge of the 78 Wh battery, which covers a full daylight monitoring cycle in the field (typically 12–13 h between sunrise and sunset in Macao). With a supplementary solar panel rated at ~20 W—comfortably exceeding the 5.77 W steady-state draw—the terminal can in principle sustain indefinite daylight operation, with the battery acting as a buffer for nighttime and overcast periods. The 4.6 W gap between P_{idle} and P_{total} is dominated by the screen and the UVC camera rather than by the NPU itself, which is consistent with the highly efficient operating point already characterized in Table 7 (1.54 W P_{inf} for board-level inference at 320×320). For deployments that do not require on-site visual feedback—for example, headless detection nodes uploading only species labels—disabling the screen would push runtime well past 24 h; this configuration is left as a deployment option rather than as a separate experiment.

Taken together, the per-thread runtime breakdown (Table 10) and the system-level power budget (Table 11) establish two things that the board-level numbers in Sections 5.4.1–5.4.3 alone could not. First, the EbirdEye terminal is real-time on the chosen hardware, and that real-time figure is delivered end-to-end at the thread level rather than by the NPU alone—the ~23.5 ms latency is what the user sees, not just what the NPU produces. Second, the resulting throughput can be sustained for a full daylight cycle on a battery and, under solar augmentation, indefinitely, which closes the gap between “real-time” as a benchmark claim and “real-time” as a field-deployment property. The board-level

efficiency figures of Section 5.4.3 carry over to the integrated terminal essentially intact: the NPU's 1.54 W operating point sits well below the camera-and-screen overhead, so adding peripherals does not undo the energy advantage that the operator-friendly architecture earns at the silicon level.

5.7. Inference Visualization on Real-World Images

This section visualizes the model's inference results on real-world images captured along the riverbank near the University of Macau, with all inferences performed on the EbirdEye terminal at FP16 precision. Figure 11 presents six representative examples: three for the flagship species *Platalea minor* (Black-faced Spoonbill) under different conditions, and three for other priority species from the Macao-ebird-det dataset. The six examples cover the main scenarios that show up in field monitoring, and the detector handles them all successfully.

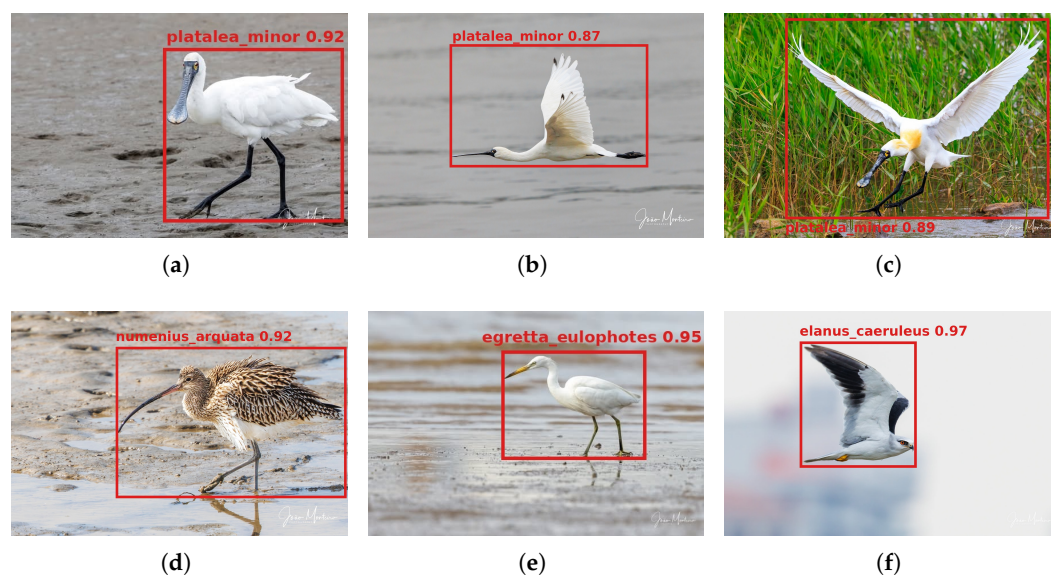


Figure 11. Inference visualization of eBirdNet-Nano on real-world images captured along the riverbank. Source images courtesy of João Monteiro. (a) *Platalea minor*—side view walking; (b) *Platalea minor*—flight pose; (c) *Platalea minor*—cluttered background; (d) *Numenius arquata* on intertidal flat; (e) *Egretta eulophotes* foraging; (f) *Elanus caeruleus* in flight.

The three *Platalea minor* images (a–c) are the most instructive. Case (a) shows the species in a clean side-view walking posture, where the diagnostic spatulate bill is fully visible against a plain mudflat background—effectively the easiest case. Case (b) captures the same species in flight with wings fully spread, a completely different outline from (a). Case (c) pushes difficulty further by combining an in-flight breeding adult with a cluttered reed background. Confidence scores remain high across all three poses (0.92, 0.87, 0.89), and the species label stays correct throughout. This matters in practice because bird posture and background in the wild are never something the camera operator can control, and a detector that only works on canonical poses would be of limited operational use.

Among the bottom row, the case worth calling out is (d). The *Numenius arquata* has brown, streaked plumage that blends almost seamlessly into the mottled intertidal substrate, and to the human eye the bird takes a second to spot. The model still produces a confident (0.92) and tightly fitted bounding box. This is a meaningful test of the channel/depth-compressed backbone: a more aggressive compression would be expected to lose exactly this kind of fine-grained texture sensitivity, yet the compressed network preserves enough of it to separate the bird from a color-matched background. The remaining cases (e) and (f) confirm the same conclusion under easier conditions, with per-detection model output

confidence scores (class probability $\times$ objectness) of 0.95 (e) and 0.97 (f), respectively. Taken together with the quantitative results of the preceding sections, these examples show that eBirdNet-Nano's efficiency gains do not translate into visibly worse detection.

6. Conclusions and Future Work

This paper presented eBirdNet-Nano, a 1.05-million-parameter detector for endangered-bird monitoring on heterogeneous edge SoCs, derived from the YOLOv12n baseline through a three-level NPU-friendly transformation acting at the operator (NPUConv), module (NPU-C3k2 and NPU-SE-Block), and detection-head (balanced 64-channel output and reg-max = 1) levels. The resulting model attains an mAP@0.5 of 0.929 on the curated Macao-ebird-det dataset and, on a Rockchip RK3588 NPU under INT8 quantization, achieves 13.83 ms NPU inference latency and 26.76 ms end-to-end latency per frame at 37.4 FPS, together with 89.5% latency stability, 24.30 FPS per watt (FPS/W) energy efficiency at 320×320 , and a 54% reduction in per-frame energy consumption at 640×640 relative to the YOLOv12n baseline. Integrated into the EbirdEye terminal—a LubanCat-RK3588s-based observation node equipped with UVC capture, a MIPI touchscreen, and a 78 watt-hour (Wh) battery pack—the same model sustains 23.5 ms thread-level end-to-end latency during live operation while supporting approximately 13.5 h of autonomous runtime per charge.

More fundamentally, the central contribution of this work lies in the deployment-oriented methodology revealed through systematic ablation analysis. A four-stage ablation study conducted on Macao-ebird-det demonstrates that operator type rather than parameter count is the dominant factor governing NPU inference efficiency under current consumer deployment toolchains. Specifically, a modest 5.9% parameter reduction introduced solely through operator replacement yields a $1.78\times$ FP16 end-to-end speedup ($2.10\times$ on NPU-only inference time), while the architectural optimization factor ($2.97\times$) and INT8 quantization factor ($1.19\times$) compose multiplicatively to yield the measured $3.53\times$ end-to-end speedup. Cross-architecture profiling further reinforces this observation from the opposite direction: when the hardware bottleneck shifts from operator scheduling on NPUs to raw single-instruction multiple-data multiply-accumulate (SIMD-MAC) throughput on ARM and RISC-V CPUs, the architectural advantage correspondingly diminishes and INT8 quantization becomes the dominant optimization mechanism. The practical implication is therefore clear: when a lightweight detector fails to satisfy real-time requirements on an edge NPU, the first optimization target should be the operator graph and execution compatibility rather than merely reducing channel counts or applying more aggressive quantization schemes.

While our results validate the efficiency of eBirdNet-Nano, several limitations naturally suggest directions for future work: (i) The most immediate concern is dataset expansion. Macao-ebird-det currently covers 24 species with annotations generated using Grounding DINO and subsequently verified manually; extending the dataset toward broader Pearl River Estuary avifauna coverage, together with a fully expert-annotated consistency evaluation on a held-out subset, will allow residual annotation bias to be quantitatively characterized rather than qualitatively acknowledged. (ii) Bridging the accuracy trade-off of our lightweight network motivates future investigation into complementary compression strategies operating below the operator-graph level—such as QAT, structured channel sensitivity pruning, and knowledge distillation from larger YOLOv12 teacher models—which compose additively to recover accuracy while fully preserving our NPU-resident latency profile. (iii) Automating the proposed three-level transformation as compiler passes represents a key technical objective: current open infrastructure (Tensor Virtual Machine, TVM; Multi-Level Intermediate Representation–Intermediate Representation Exe-

cution Environment, MLIR-IREE) and proprietary toolchains (RKNN, nncase) do not yet expose extensible operator-support-matrix matching, leaving room for an ‘NPU-friendliness linter’ that proposes L1/L2 substitutions against a target manifest. (iv) Broadening hardware generalization is crucial to validating the framework’s scalability. While Section 5.5 validates the methodology on both the RK3588 NPU and the Kendryte K230 KPU, extending it to other accelerator families—Qualcomm Hexagon NPU, MediaTek application processing unit (APU), Intel/Movidius vision processing unit (VPU), or future toolchains adding dynamic-shape support—will further characterize how the operator-graph leverage point varies across operator-support matrices and compiler ecosystems. (v) Although eBirdNet-Nano was developed specifically for regional endangered-bird monitoring, the core backbone and module-level transformations remain task-agnostic. Evaluating zero-shot transfer or training under our operator-aware pipeline on broader, large-scale object detection benchmarks such as MS COCO or VisDrone will help characterize how these NPU-resident design principles scale as the class cardinality and scene diversity increase.

Author Contributions: Conceptualization, X.H., L.S. and S.-K.T.; methodology, X.H.; software, X.H.; validation, X.H.; formal analysis, X.H.; investigation, X.H. and L.S.; resources, S.-K.T.; data curation, X.H.; writing—original draft preparation, X.H.; writing—review and editing, L.S. and S.-K.T.; visualization, X.H.; supervision, S.-K.T.; project administration, S.-K.T.; funding acquisition, S.-K.T. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded by the Institutional Educational Quality Project of Dongguan University of Technology (grant numbers 2024020105 and 202502006), and the Undergraduate Science and Technology Innovation Cultivation Special Fund Project of Dongguan University of Technology (grant number pdjh2025bk213), and the Undergraduate Innovation and Entrepreneurship Training Program of Dongguan University of Technology (grant numbers 202511819008 and 202511819161). Additional support was provided in part by the research grant (RP/FCA-09/2023) offered by Macao Polytechnic University and in part by Faculty of Applied Sciences, Macao Polytechnic University (fca.321d.97af.8).

Data Availability Statement: The Macao-ebird dataset is publicly available at <https://www.kaggle.com/datasets/huangxiaoyuan/macao-ebird> (accessed on 26 June 2025). Source code is released in GitHub repositories at <https://github.com/huangxiaoyuan/ultralytics-rknn-hxy> (accessed on 20 March 2026).

Acknowledgments: The authors would like to express their sincere gratitude to João Monteiro for generously granting permission to use his field photographs in this work. His high-quality wildlife photography and long-term dedication to bird observation and conservation provided valuable visual resources that significantly supported the presentation and ecological context of this study.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. IUCN. The IUCN Red List of Threatened Species. Version 2023-1. 2023. Available online: <https://www.iucnredlist.org> (accessed on 2 March 2025).
2. BirdLife International. *State of the World’s Birds 2022: Insights and Solutions for the Biodiversity Crisis*; BirdLife International: Cambridge, UK, 2022. Available online: <https://www.birdlife.org/state-of-the-worlds-birds/> (accessed on 10 June 2025).
3. Yu, Y.-T.; Hong Kong Bird Watching Society. *International Black-faced Spoonbill Census 2023 Report*; Black-faced Spoonbill Research Group, The Hong Kong Bird Watching Society: Hong Kong, China, 2023. Available online: <https://www.hkbws.org.hk/cms/en/hkbws/work/endangered-species/bfs-en/bfs-census-2023-en> (accessed on 14 March 2025).
4. Macao Environmental Protection Bureau (DSPA). *Report on the State of Macao’s Environment 2024*; Macao Special Administrative Region: Macao, China, 2025. Available online: <https://www.dspa.gov.mo/> (accessed on 15 March 2026).
5. Zhang, M.; Zhou, F.; Zhang, G.; Chen, S.; Li, Z. Human disturbance effect on black-faced spoonbill *Platalea minor* wintering in Macao. *Chin. J. Zool.* **2010**, *45*, 75–81.
6. Wong, K.-K.; Ho, K.-C.; Hon, P.-S.; Tse, K.-H. Prosperity and environmental challenge in a gaming city: Environmental awareness of citizens in the Macao Special Administrative Region, China. *Int. J. Environ. Stud.* **2014**, *71*, 139–153. [CrossRef]

7. Chalmers, C.; Fergus, P.; Wich, S.; Longmore, S.N.; Walsh, N.D.; Oliver, L.; Warrington, J.; Quinlan, J.; Appleby, K. AI-Driven Real-Time Monitoring of Ground-Nesting Birds: A Case Study on Curlew Detection Using YOLOv10. *Remote Sens.* **2025**, *17*, 769. [CrossRef]
8. Félix-Jiménez, A.F.; Sánchez-Lee, V.S.; Acuña-Cid, H.A.; Ibarra-Belmonte, I.; Arredondo-Morales, E.; Ahumada-Tello, E. Integration of YOLOv8 Small and MobileNet V3 Large for Efficient Bird Detection and Classification on Mobile Devices. *AI* **2025**, *6*, 57. [CrossRef]
9. Fang, S.; Shen, Y.; Zou, H.; Yin, Y.; Jin, W.; Zhou, H. Birds-YOLO: A Bird Detection Model for Dongting Lake Based on Modified YOLOv11. *Biology* **2025**, *14*, 1515. [CrossRef] [PubMed]
10. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
11. Jocher, G.; Chaurasia, A.; Qiu, J. YOLOv8 by Ultralytics. 2023. Available online: <https://github.com/ultralytics/ultralytics> (accessed on 14 March 2025).
12. Wang, C.-Y.; Yeh, I.-H.; Liao, H.-Y.M. YOLOv9: Learning what you want to learn using programmable gradient information. In Proceedings of the European Conference on Computer Vision (ECCV); Springer: Cham, Switzerland, 2024; pp. 1–21.
13. Wang, A.; Chen, H.; Liu, L.; Chen, K.; Lin, Z.; Han, J.; Ding, G. YOLOv10: Real-Time End-to-End Object Detection. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 10–15 December 2024.
14. Jocher, G.; Qiu, J. Ultralytics YOLO11. 2024. Available online: <https://github.com/ultralytics/ultralytics> (accessed on 14 July 2025).
15. Tian, Y.; Ye, Q.; Doermann, D. YOLOv12: Attention-Centric Real-Time Object Detectors. *Adv. Neural Inf. Process. Syst.* **2025**, *38*, 78433–78457.
16. Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In Proceedings of the European Conference on Computer Vision (ECCV), Zürich, Switzerland, 6–12 September 2014; pp. 740–755.
17. Electronics, Rockchip. RKNN-Toolkit2 (v2.0) Reference Manual. 2024. Available online: <https://github.com/airockchip/rknn-toolkit2> (accessed on 14 March 2025).
18. Canaan, Inc. *Nncase: Open Deep Learning Compiler Stack for Kendryte AI Accelerators*; GitHub: San Francisco, CA, USA, 2024. Available online: <https://github.com/kendryte/nncase> (accessed on 14 March 2025).
19. Tencent. NCNN: A High-Performance Neural Network Inference Framework. 2017. Available online: <https://github.com/Tencent/mncnn> (accessed on 14 March 2025).
20. Quadric. *NPU Fallback to CPU: The Hidden Performance Killer in Edge AI Chips*; Quadric Blog: Burlingame, CA, USA, 2022. Available online: <https://quadric.ai/blog/npu-fallback-cpu-performance-edge-ai> (accessed on 14 March 2025).
21. Kong, C.; Li, F.; Yan, X.; Yang, J.; Mo, P.; Luo, Q.; Mao, R. Object Detection on Low-Compute Edge SoCs: A Reproducible Benchmark and Deployment Guidelines. *Sci. Rep.* **2026**, *16*, 5875. [CrossRef] [PubMed]
22. Gavai, N.R.; Jakhade, Y.A.; Tribhuvan, S.A.; Bhattad, R. MobileNets for flower classification using TensorFlow. In Proceedings of the 2017 International Conference on Big Data, IoT and Data Science (BIGDATA), Pune, India, 11–12 December 2017; pp. 154–158.
23. Ma, N.; Zhang, X.; Zheng, H.-T.; Sun, J. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 10–12 September 2018; pp. 116–131.
24. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. GhostNet: More Features from Cheap Operations. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 16–18 June 2020; pp. 1580–1589.
25. Li, Z.; Paolieri, M.; Golubchik, L. Inference Latency Prediction for CNNs on Heterogeneous Mobile Devices and ML Frameworks. *Perform. Eval.* **2024**, *165*, 102429. [CrossRef]
26. Banner, R.; Nahshan, Y.; Soudry, D. Post Training 4-Bit Quantization of Convolutional Networks for Rapid-Deployment. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 8–14 December 2019.
27. Esser, S.K.; McKinstry, J.L.; Bablani, D.; Appuswamy, R.; Modha, D.S. Learned Step Size Quantization. In Proceedings of the International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, 26–30 April 2020.
28. Xu, D.; Zhang, H.; Yang, L.; Liu, R.; Huang, G.; Xu, M.; Liu, X. Fast On-Device LLM Inference with NPUs. In Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2025), Rotterdam, The Netherlands, 30 March–3 April 2025; pp. 445–462. [CrossRef]
29. Li, J.; Ye, J. Edge-YOLO: Lightweight Infrared Object Detection Method Deployed on Edge Devices. *Appl. Sci.* **2023**, *13*, 4402. [CrossRef]
30. Huang, X.; Mirri, S.; Tang, S.-K. Macao-eBird: A Curated Dataset for Artificial-Intelligence-Powered Bird Surveillance and Conservation in Macao. *Data* **2025**, *10*, 84. [CrossRef]
31. Ma, J.; Guo, J.; Zheng, X.; Fang, C. An Improved Bird Detection Method Using Surveillance Videos from Poyang Lake Based on YOLOv8. *Animals* **2024**, *14*, 3353. [CrossRef] [PubMed]
32. Kellenberger, B.; Marcos, D.; Tuia, D. Detecting Mammals in UAV Images: Best Practices to Address a Substantially Imbalanced Dataset with Deep Learning. *Remote Sens. Environ.* **2018**, *216*, 139–153. [CrossRef]

33. Vo, H.-T.; Thien, N.N.; Mui, K.C. Bird Detection and Species Classification: Using YOLOv5 and Deep Transfer Learning Models. *Int. J. Adv. Comput. Sci. Appl.* **2023**, *14*, 939–947. [[CrossRef](#)]
34. Xing, C.; Qu, C.; Zhang, P.; Qin, X. An efficient method for monitoring small bird targets in wetland environments based on object detection. *Sci. Rep.* **2026**, *16*, 15828. [[CrossRef](#)] [[PubMed](#)]
35. Rossi Labianca, L.; Álvarez Urueña, J.; García Serrada, R.; Curto Hernández, J. On-Edge Artificial Intelligence Architecture for Real-Time Bird Identification for Ecosystem Protection. In *International Symposium on Distributed Computing and Artificial Intelligence*; Springer: Cham, Switzerland, 2025; pp. 55–66. [[CrossRef](#)]
36. Kahl, S.; Wood, C.M.; Eibl, M.; Klinck, H. BirdNET: A Deep Learning Solution for Avian Diversity Monitoring. *Ecol. Inform.* **2021**, *61*, 101236. [[CrossRef](#)]
37. Baowaly, M.K.; Sarkar, B.C.; Walid, M.A.A.; Ahamad, M.M.; Singh, B.C.; Alvarado, E.S.; Ashraf, I.; Samad, M.A. Deep Transfer Learning-Based Bird Species Classification Using Mel Spectrogram Images. *PLoS ONE* **2024**, *19*, e0305708. [[CrossRef](#)] [[PubMed](#)]
38. Chandrashekhar, A.; Satyanarayana, B.; Gorrepati, R.R.; Vasanthi, P.; Prasanna, K.L. An efficient YOLOv12-based framework for detecting extremely small-scale objects. *Sci. Rep.* **2025**, *15*, 1042.
39. Liang, X.; Zhang, Y.; Wu, N.; Yu, H. LCS-YOLO: A lightweight and efficient small object detection algorithm for UAV. *Earth Sci. Inform.* **2026**, *19*, 75. [[CrossRef](#)]
40. Qiu, Y.; Lin, Z. Lightweight model LMW-YOLO for small object detection in remote sensing images. *Sci. Rep.* **2026**, *16*, 3241. [[CrossRef](#)]
41. Chen, H.; Wang, Y.; Guo, J.; Tao, D. VanillaNet: The Power of Minimalism in Deep Learning. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023.
42. Lin, T.-Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature Pyramid Networks for Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 22–25 July 2017; pp. 2117–2125.
43. Liu, S.; Qi, L.; Qin, H.; Shi, J.; Jia, J. Path Aggregation Network for Instance Segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 8759–8768.
44. ONNX Runtime Developers. ONNX Runtime. 2017. Available online: <https://onnxruntime.ai/> (accessed on 14 July 2025).
45. Suchý, I.; Turčaník, M. Review of large YOLOv8 and RT-DETR energy efficiency on edge devices for real-time detection. *Sci. Rep.* **2026**, *16*, 5184. [[CrossRef](#)]
46. Hu, J.; Shen, L.; Sun, G. Squeeze-and-Excitation Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 7132–7141.
47. Liu, S.; Zeng, Z.; Ren, T.; Li, F.; Zhang, H.; Yang, J.; Li, C.; Yang, J.; Su, H.; Zhu, J.; et al. Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection. In Proceedings of the European Conference on Computer Vision (ECCV), Milano, Italy, 1–4 October 2024; pp. 38–55.
48. Tan, M.; Pang, R.; Le, Q.V. EfficientDet: Scalable and efficient object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 10781–10790. [[CrossRef](#)]
49. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; Berg, A.C. SSD: Single Shot MultiBox Detector. In Proceedings of the European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 11–14 October 2016; Springer: Cham, Switzerland, 2016; pp. 21–37. [[CrossRef](#)]
50. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520. [[CrossRef](#)]
51. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778. [[CrossRef](#)]
52. Zhou, X.; Wang, D.; Krähenbühl, P. Objects as points. *arXiv* **2019**. [[CrossRef](#)]
53. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 7–12 December 2015; Volume 28, pp. 91–99.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.