EAI Endorsed Transactions
on AI and Robotics
Research Article EAI.EU

# Hardware-Aware INT8 Quantization and FPGA Deployment of MobileNetV2 for Real-Time Facial Landmark Detection

Van-Khoa Pham*, Manh-Dung Do, Trung-Nghia Dang, Long Tran

Ho Chi Minh City University of Technology and Engineering, Vietnam

## Abstract

Facial landmark detection is a key component of always-on edge vision systems, but practical deployment requires balancing localization accuracy, model size, throughput, and power consumption. This study proposes a two-stage, hardware-aware framework for MobileNetV2-based facial landmark detection. In Stage I, a lightweight detector is developed in PyTorch and evaluated in FP32 and INT8 using post-training quantization (PTQ) and quantization-aware training (QAT). In Stage II, the quantized model is realized on the AMD/Xilinx Kria KV260 FPGA and assessed in terms of real-time throughput, power consumption, and hardware resource utilization. INT8 quantization reduces the model size from 6.59 MB to 1.65 MB, and QAT retains accuracy more effectively than PTQ (91.74% vs. 90.94%) relative to the FP32 baseline (92.42%). The hardware implementation achieves approximately 30 FPS at approximately 3 W while using 14.8% of LUTs, 8.1% of FFs, 16.3% of BRAM, and 4.5% of DSPs. Among the evaluated platforms, the KV260 delivers the highest measured energy efficiency, whereas the RTX 4060 delivers the highest throughput. Within the evaluated setup, these results support the practicality of explicitly separating software-stage quantization analysis from hardware-stage realization for real-time, low-power facial landmark detection on FPGA-based edge platforms.

**Keywords:** facial landmark detection; MobileNetV2; quantization-aware training; post-training quantization; edge computing; AMD/Xilinx Kria KV260.

Received on 25 August 2025, accepted on 20 March 2026, published on 25 March 2026



doi: 10.4108/airo.10070

*Corresponding author. Email: khoapv@hcmute.edu.vn

## 1. Introduction

Artificial intelligence (AI) is increasingly moving from cloud-centric processing to edge inference, where sensing, perception, and decision-making are performed close to the data source. This shift is particularly important in Internet-of-Things (IoT) systems, where latency, communication overhead, and service continuity are critical design constraints. Prior works on fog computing, cloudlets, and edge computing have established the value of moving computation toward the network edge to support interactive, data-intensive, and time-sensitive applications [1]–[3].

A representative application domain is intelligent transportation, in which onboard sensing systems continuously generate visual and multimodal data that must be interpreted in real time. Within this context, driver monitoring systems (DMSs) have attracted increasing attention because driver drowsiness remains a major road-safety concern [4]. For practical deployment, a DMS must operate continuously with low latency and high reliability under tight constraints on power consumption, thermal dissipation, and embedded hardware cost. These requirements make onboard edge inference more appropriate than cloud-dependent processing for in-cabin safety functions [2], [3].

Among vision-based perception modules, facial landmark detection is a particularly attractive choice for DMS because it provides a compact and non-intrusive representation of driver state through key facial geometry. Landmark localization supports downstream analysis of eye closure, mouth opening, and head pose, all of which are relevant to fatigue and attention assessment. More broadly, face detection and recognition have been widely studied in robotic vision systems [5]. Recent studies have also shown that lightweight landmark detectors can remain effective under practical conditions. In particular, PFLD demonstrates that compact landmark models can achieve strong efficiency [6], while pose-assisted alignment improves robustness under large pose variation [7] and boundary-aware modelling strengthen localization in challenging facial configurations [8].

Despite this progress, real-time deployment on low-power embedded hardware remains challenging. Lightweight backbones such as MobileNetV2 are well suited to constrained inference because their inverted residual structure and depth wise separable convolutions reduce computation while preserving strong representational efficiency [9]. In parallel, quantization has become a key deployment strategy because low-precision inference reduces model size, memory traffic, and arithmetic cost [10]. For FPGA-based edge AI, however, architectural efficiency alone is insufficient; the network must also be compatible with integer-oriented execution and practical hardware realization. This requirement motivates a hardware-aware design strategy in which model architecture, quantization, and deployment constraints are considered jointly.

Motivated by this gap, this study presents a hardware-aware quantization framework for MobileNetV2-based facial landmark detection targeting real-time FPGA deployment in edge DMS scenarios. Unlike prior studies that primarily emphasize lightweight landmark modelling [6]–[8] or general edge-computing and low-precision inference principles [1]–[3], [10], [11], this work explicitly links model-level optimization with system-level hardware realization. Specifically, the study makes three contributions. First, it develops a lightweight MobileNetV2-based facial landmark detector and evaluates INT8 quantization in software using both post-training quantization (PTQ) and quantization-aware training (QAT). Second, it realizes a hardware-oriented quantized MobileNetV2 variant on the AMD/Xilinx Kria KV260 and analyses its deployment characteristics in terms of throughput, power consumption, and resource utilization. Third, by explicitly separating software-stage quantization analysis from hardware-stage FPGA realization, it clarifies the trade-off among accuracy retention, model compactness, and practical low-power edge deployment.

The remainder of this study is organized as follows. Section II reviews related work on facial landmark detection, edge deployment, and FPGA-oriented acceleration. Section III presents the two-stage methodology, including the customized MobileNetV2 architecture, the INT8 quantization workflow, and the hardware realization on the AMD/Xilinx Kria KV260 platform. Section IV reports the software-stage and hardware-stage results, including accuracy, throughput, energy efficiency, resource utilization, and power consumption. Section V concludes the study and outlines future research directions.

## 2. Literature Review

Facial landmark detection has been extensively studied, with increasing emphasis on models that are not only accurate but also efficient enough for real-time deployment on resource-constrained platforms. Among lightweight approaches, Guo *et al*. introduced the Practical Facial Landmark Detector (PFLD), showing that compact network design can still provide strong landmark localization performance for mobile and embedded scenarios [6]. To improve robustness under large pose variations, Yang *et al*. incorporated explicit head-pose estimation into the alignment process and demonstrated its effectiveness for non-frontal facial landmark localization [7]. Wu *et al*. further improved alignment accuracy by exploiting facial boundary information to impose stronger geometric constraints during landmark regression [8]. These studies indicate that effective facial landmark detection depends on both lightweight model design and robust modelling of facial geometry.

Beyond network architecture, low-precision inference has become a key enabler for practical edge deployment. Nagel *et al*. proposed a data-free post-training quantization method based on weight equalization and bias correction, showing that 8-bit quantization can be improved even without access to the original training data [12]. Zhao *et al*. further reduced quantization-induced degradation through outlier channel splitting, addressing the sensitivity of low-bit models to extreme activation channels [13]. A foundational contribution was provided by Jacob *et al*., who formalized integer-arithmetic-only inference and introduced a training procedure that preserves accuracy under quantization, thereby establishing an important basis for efficient on-device deployment [14].

FPGAs have emerged as a promising platform for efficient deep neural network acceleration because of their reconfigurability, parallelism, and favourable energy efficiency. Zhang *et al*. showed that accelerator performance on FPGA is strongly constrained by the balance between computation throughput and memory bandwidth, making dataflow and memory-access optimization central to efficient CNN deployment [15]. Nurvitadhi *et al*. compared FPGAs and GPUs for next-generation deep neural networks and highlighted the trade-off between peak throughput and performance-per-watt, showing that FPGAs can be highly competitive when reduced precision and sparsity are effectively exploited [16]. Focusing on lightweight backbones, Trogoderma's *et al.* explored sparse MobileNetV2 deployment on FPGAs using high-level synthesis and sparse matrix techniques,

demonstrating that model-aware hardware optimization can reduce latency and resource utilization in embedded inference settings [17].

Related evidence from other vision tasks also suggests that deployment-oriented performance does not depend only on model compactness, but also on the quality and diversity of training data. Setiadi *et al*. showed that augmenting a limited mammogram dataset with synthetic images improved the performance of an EfficientNetV2L-based classifier [18]. Although this study targets medical image classification rather than facial landmark detection, it supports a broader observation relevant to edge vision systems: when real annotated data are limited, data-centric enhancement can complement model-level efficiency techniques to improve robustness and generalization.

This perspective is consistent with broader developments in neural network quantization. Nagel *et al*. summarized the PTQ-QAT landscape and emphasized the practical trade-off that PTQ is attractive for rapid 8-bit deployment, whereas QAT is often needed when stronger robustness to quantization noise is required [19]. Cai *et al*. extended the data-free direction by proposing Zero, a zero-shot quantization framework that enables calibration without access to the original training or validation data [20]. Earlier work by Houbara *et al*. showed that low-precision weights and activations can be incorporated directly during training, providing an important conceptual foundation for later training-based quantization methods [21]. More recently, He *et al*. proposed Efficient, a parameter-efficient quantization-aware fine-tuning framework for low-bit diffusion models that seeks to approach QAT-level performance with substantially reduced adaptation cost [22]. Although developed for diffusion models rather than lightweight CNNs, this work reflects a broader trend in quantization research toward reducing the overhead of fine-tuning while retaining the accuracy benefits of training-aware optimization. In parallel, Wuraola and Patel showed that activation-function design has a direct impact on arithmetic cost and hardware efficiency, further motivating the use of hardware-friendly operators when mapping compact neural networks to accelerator fabrics [23].

Despite these advances, prior work has largely treated facial landmark detection, neural network quantization, and FPGA acceleration as separate research directions. Existing facial landmark studies mainly focus on localization robustness and geometric modelling [6]–[8], while quantization studies emphasize generic low-precision inference strategies [12]–[14], [19]–[22], and FPGA studies predominantly address accelerator architecture and efficient CNN mapping [15]–[17], [23]. Therefore, a gap remains in the joint optimization of a lightweight facial landmark detector for both quantized inference and real-time FPGA deployment. Motivated by this gap, this work investigates a hardware-aware INT8 quantization strategy for a MobileNetV2-based facial landmark detector and validates its effectiveness on an FPGA platform for real-time, low-power edge inference.

## 3. Methods

### 3.1. Two-Stage Hardware-Aware Process

This study is organized as a two-stage hardware-aware methodology in order to clearly separate software-side model optimization from hardware-side FPGA realization, as illustrated in Fig. 1. Stage I covers model design, FP32 training, and INT8 quantization analysis for a lightweight facial landmark detector implemented in PyTorch. Stage II covers the realization of a hardware-oriented INT8 model on the AMD/Xilinx Kria KV260 and its evaluation in terms of throughput, power consumption, and hardware resource utilization. This separation is deliberate: the first stage isolates algorithmic behaviour under a reproducible deep-learning workflow, whereas the second stage evaluates whether the resulting quantized model can be executed efficiently under practical deployment constraints.

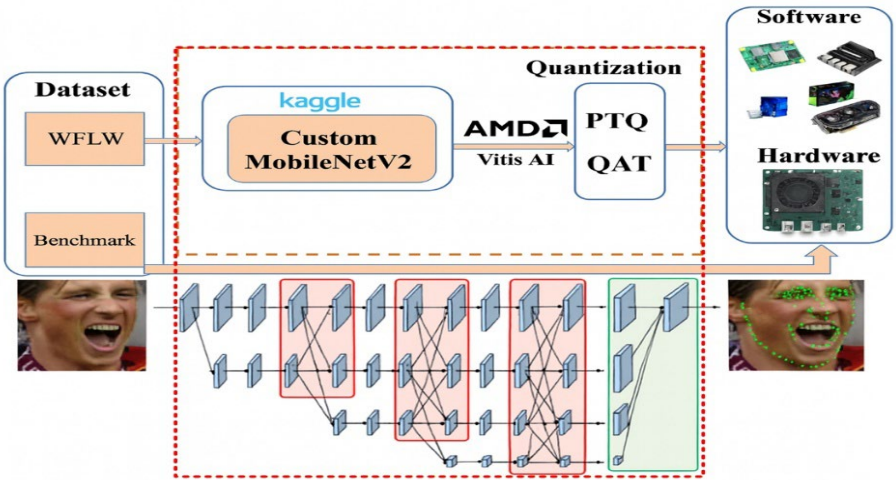


**Figure 1.** Overall structure of the proposed two-stage methodology. Stage I covers software-stage model development, FP32 training, and INT8 quantization analysis using PTQ and QAT. Stage II covers hardware-stage

Table 1. Summary of training, quantization, and deployment settings

| Metric | Parameter | Value |
|---|---|---|
| Dataset | Benchmark | WFLW |
| | Data split | 90% training, 10% validation; separate test set used for evaluation |
| Training | Framework/ device | PyTorch; mixed precision; CUDA GPU or CPU fallback |
| | Input Image | Face crop, grayscale conversion, normalization, resizing to 112×112×1 |
| | Loss | Weighted pose-aware MSE with auxiliary Euler-angle supervision |
| | Optimizer | Adam |
| | Weight decay | $1\times10^{-5}$ |
| | Batch size | 64 |
| | Epochs | typical convergence in 60–80 epochs |
| | Learning-rate schedule | $1\times10^{-3}$ to $1\times10^{-5}$ |
| | Early stopping | Patience = 10 epochs |
| | Gradient clipping | L2 norm clip = 10 |
| Model | Activation | ReLU |
| | Extra layers | Additional 3×3 Conv-BatchNorm-ReLU layers before the regression head |
| | Output | 98 facial landmark coordinates |
| Quantization | Bit-width | INT8 weights and activations |
| | Methods | PTQ and QAT |
| | PTQ calibration subset | Representative calibration subset |
| | QAT setting | Fake quantization + fine-tuning |
| Deployment | Toolchain | PyTorch→ONNX→ hardware-aware FPGA compilation flow |
| | Target platform | AMD/Xilinx Kria KV260 |
| Measurement | Accuracy metrics | Normalized landmark error and landmark accuracy |
| | Throughput/ power | Throughput (FPS), energy efficiency (FPS/W), and FPGA resource utilization |

The methodological novelty of this study lies in the explicit co-design of three elements that are often treated separately: (i) a customized MobileNetV2-based facial landmark detector, (ii) comparative INT8 quantization using both PTQ and QAT, and (iii) a deployment-stage FPGA realization on the AMD/Xilinx Kria KV260. Accordingly, the proposed contribution should be interpreted not as a new backbone or a new quantization algorithm in isolation, but as a unified software-to-hardware optimization and deployment framework for real-time, low-power facial landmark detection at the edge.

## 3.2. PyTorch-Based Facial Landmark Detector

In Stage I, the detector is implemented in PyTorch using a PFLD-style formulation that is well aligned with lightweight edge inference [6]. For both training and inference, each face crop is converted to grayscale, normalized, and resized to 112×112×1. This choice reduces computational cost and memory traffic while preserving the facial geometric information required for landmark localization. The software-stage training setup uses mixed precision, Adam optimization, early stopping, gradient clipping, and a validation-driven learning-rate schedule. On the WFLW benchmark, 7,500 images are used for training and 2,500 for testing, with 10% of the training split reserved for validation.

The selected training hyperparameters are summarized in Table 1. In particular, the model is trained with Adam, weight decay $1\times10^{-5}$, batch size 64, and up to 100 epochs, with practical convergence typically observed in 60–80 epochs. A cosine-annealing schedule decreases the learning rate from $1\times10^{-3}$ to $1\times10^{-5}$, while early stopping and gradient clipping are used to stabilize optimization. These settings are intended to provide a reproducible and deployment-oriented FP32 baseline before INT8 quantization is applied.

## 3.3. Customized MobileNetV2-Based Facial Landmark Detector

The proposed detector is built on a customized MobileNetV2 backbone, shown in Fig. 2. The design retains the lightweight inverted residual and linear bottleneck principles of MobileNetV2 because these operators provide an efficient balance between representational capacity and computational cost. On top of this backbone, the network includes a compact regression head for 98 facial landmark coordinates and a training-only auxiliary pose branch that is removed during inference. This auxiliary branch improves robustness under large head-pose variation without introducing any runtime overhead [24], [25].

Two architectural modifications are central to the proposed design. First, the original ReLU6 activations are replaced with standard ReLU. Second, additional 3×3 Conv-BatchNorm-ReLU layers are inserted before the final pooling and fully connected regression head. These changes preserve the compactness of the original backbone while improving its compatibility with low-precision inference and FPGA-oriented execution. The activation choice is made explicitly hardware-aware. Standard ReLU and ReLU6 are defined as:

$$\mathrm{ReLU}(x)=\max(0,x) \tag{1}$$

and

$$\mathrm{ReLU6}(x)=\min(\max(0,x),6). \tag{2}$$

Equation (1) requires only a lower-bound comparison, whereas (2) requires both a lower-bound and an upper-bound saturation operation. On LUT-based FPGA fabrics, this additional compare-select stage increases combinational logic complexity and potentially extends the critical path. For this reason, REL is used throughout the customized backbone as the more hardware-friendly activation for the KV260 realization considered in this study.

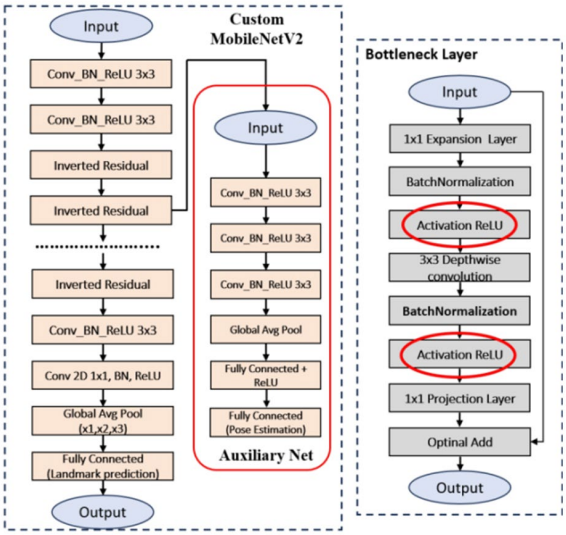


**Figure 2.** Customized MobileNetV2 architecture used for facial landmark detection. The network includes a lightweight backbone, a regression head for 98 landmark coordinates, and a training-only auxiliary pose branch that is removed during inference

## 3.4. Quantization Strategy

After FP32 training, the model is converted to INT8 weights and activations using two complementary strategies, namely post-training quantization (PTQ) and quantization-aware training (QAT), as summarized in Fig. 3 [26]. In both cases, the objective is to reduce model size, memory traffic, and arithmetic cost while preserving landmark accuracy as much as possible. PTQ provides a low-cost path to deployment by avoiding retraining, whereas QAT provides a training-aware route that more explicitly compensates for low-precision effects. For an activation tensor $x$, uniform affine quantization [26] is written as

$$q = \mathrm{clip}\left(round\left(\frac{x}{s}\right) + z,\ q_{min}, q_{max}\right),$$
$$\hat{x} = s(q - z) \tag{3}$$

where $q$ is the quantized integer tensor, $\hat{x}$ is the dequantized approximation, $s$ is the scale, $z$ is the zero-point, and $q_{min}, q_{max}$ are the representable INT8 limits. The scale $s$ determines the numerical step size between adjacent integer levels, while the zero-point $z$ aligns the real-valued range with the signed or unsigned integer domain. In PTQ, $s$ and $z$ are estimated from calibration data after training. In QAT, the same quantization behaviour is simulated during fine-tuning by means of fake-quantization modules so that the network can adapt to rounding and clipping errors before final conversion.

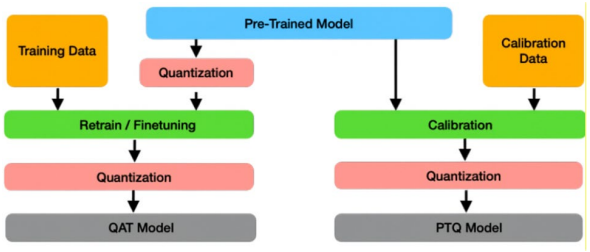


**Figure 3.** Workflows for model quantization

In this study, PTQ is configured using 500 WFLW training images, which corresponds to approximately 6.7% of the 7,500-image training split. This choice provides a practical compromise between calibration stability and calibration cost: it is large enough to capture major variation in pose, occlusion, and illumination, but still small enough to preserve the simplicity that makes PTQ attractive for rapid deployment.

QAT is expected to preserve accuracy better than PTQ for this architecture because MobileNetV2-style inverted residual blocks are especially sensitive to quantization noise. First, depth wise convolution processes each channel independently, so quantization error introduced in one channel is less likely to be averaged out later. Second, the linear bottleneck compresses features into a lower-dimensional representation with limited redundancy, making the block more sensitive to clipping and perturbation. Third, when residual connections are present, any mismatch between the quantized residual branch and the identity branch can propagate through element-wise addition and distort downstream landmark regression. Under QAT, the network adapts its intermediate feature distributions to become more

quantization-friendly, thereby reducing outlier magnitude, saturation probability, and branch-statistics mismatch before residual addition.

## 3.5. Batch Normalization Folding and Fused INT8 Inference

Batch Normalization (Batch Norm) plays two roles in this work. During training, it stabilizes feature distributions and improves convergence. During deployment, it is folded into the preceding convolution to remove a separate normalization operator and reduce quantization boundaries. Let the convolution output be

$$y = W*x + b, \quad (4)$$

and let Batch Norm be parameterized by $\gamma$, $\beta$, $\mu$, $\sigma^2$, and $\varepsilon$. The fused convolution parameters are then

$$W' = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} W, \quad (5)$$

$$b' = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} (b - \mu) + \beta \quad (6)$$

In (5) and (6), $\gamma$ and $\beta$ are the Batch Norm affine scale and shift, $\mu$ and $\sigma^2$ are the running mean and variance, and $\varepsilon$ is a small constant for numerical stability. This transformation preserves the inference output while collapsing convolution and normalization into a single equivalent linear operator. After fusion, the activation function is applied directly to the fused output, which simplifies the execution graph and is beneficial for efficient INT8 inference. In the proposed backbone, the use of standard ReLU after the fused convolution further constrains activations to non-negative values and simplifies range handling for quantized execution.

## 3.6. Quantization-Aware Training Workflow

The QAT procedure used in Stage I is summarized as follows. A pre-trained FP32 landmark detector is first loaded. Eligible layers are then fused for quantization-aware execution, and fake-quantization observers are inserted. The optimizer and learning-rate schedule are initialized, and the model is fine-tuned over mini-batches of pre-processed face crops and landmark targets. During each iteration, forward propagation is executed with simulated quantization, the weighted pose-aware loss is computed, gradients are backpropagated, and parameters are updated. The validation set is evaluated periodically. After fine-tuning converges, the network is converted to a final INT8 representation for software-stage analysis. Algorithm 1 summarizes the QAT procedure used in this study.

**Algorithm 1:** Quantization-Aware Training workflow

1. Load the pre-trained FP32 landmark detector.
2. Fuse eligible layers for quantization-aware execution.
3. Insert fake-quantization observers and QAT modules.
4. Initialize the optimizer and learning-rate scheduler.
5. For each epoch:
6. Load a mini-batch of preprocessed face crops and landmark targets.
7. Run forward propagation with simulated quantization.
8. Compute the weighted pose-aware MSE loss.
9. Backpropagate gradients and update parameters.
10. Evaluate on the validation set when scheduled.
11. End for.
12. Convert the fine-tuned model to the final INT8 representation for software-stage deployment analysis.

## 3.7. Hardware-Stage Deployment Flow

Stage II translates the software-stage quantized model into a deployable FPGA implementation. As shown in Fig. 4, the trained model is first exported from PyTorch to ONNX, after which it is processed by an FPGA-oriented compilation flow to generate a hardware accelerator suitable for the AMD/Xilinx Kria KV260 platform [27]. The generated accelerator is then integrated into the programmable logic and connected to the processing system through standard control and data-movement interfaces.

This stage is intentionally described as a hardware realization process rather than as a direct continuation of software training. Its objective is not to revisit optimization in PyTorch, but to evaluate whether the quantized detector can be executed efficiently in a live embedded setting. Accordingly, the hardware stage focuses on runtime behaviour, including throughput, power consumption, and FPGA resource utilization.

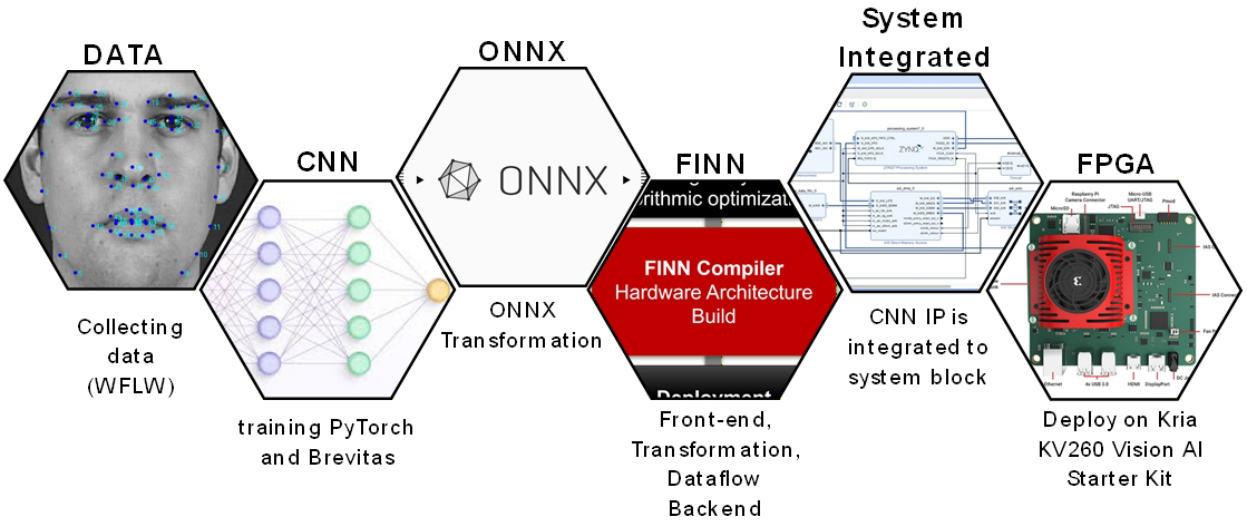


**Figure 4.** Hardware-aware deployment workflow of the proposed model, from PyTorch-based FP32 training and INT8 quantization to ONNX export, FINN/Vitis-based compilation, accelerator generation, and execution on the AMD/Xilinx Kria KV260. The workflow explicitly separates software-stage model optimization from hardware-stage realization

## 3.8. Standardized Runtime Pipeline

As illustrated in Fig. 5, the runtime workflow is standardized across the evaluated platforms and divided into three stages: preprocessing, inference, and post-processing. In preprocessing, the host captures the input frame, extracts the face region, converts it to grayscale, normalizes it, and resizes it to 112×112×1. In inference, the resulting INT8 tensor is passed to the available compute backend. For the FPGA implementation, the tensor is transferred from the processing system to the programmable logic through DMA, where the INT8-quantized MobileNetV2 landmark detector is executed. In post-processing, the network output is decoded into facial landmark coordinates, converted to FP32 where necessary, and overlaid on the input frame for visualization. By keeping preprocessing and post-processing structurally consistent across platforms, the study enables practical cross-platform comparison while allowing the execution backend to remain platform specific. The hardware-stage runtime on the AMD/Xilinx Kria KV260 is summarized in Algorithm 2.

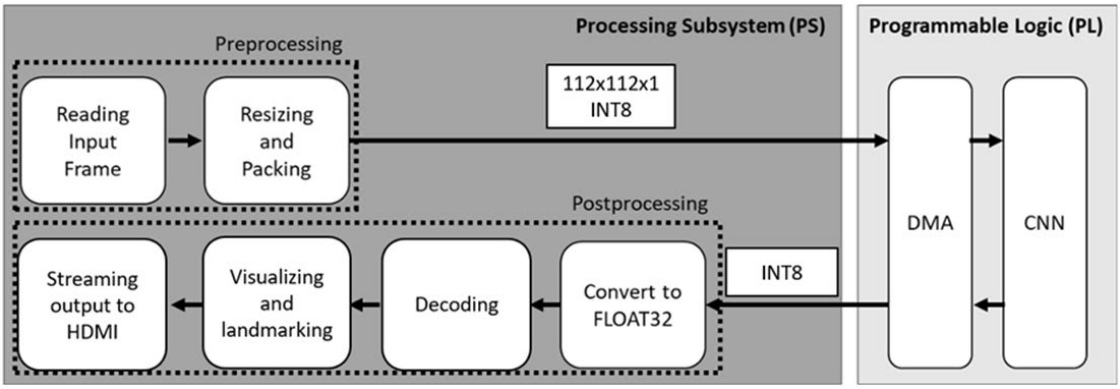


**Figure 5.** Standardized runtime pipeline used for deployment-stage comparison across GPU, CPU, embedded-board, and FPGA platforms, including host-side preprocessing, backend-specific inference execution, and landmark post-processing. For the FPGA branch, inference is executed by the hardware accelerator in programmable logic while the host manages frame acquisition, tensor transfer, and result visualization

**Algorithm 2:** AMD/Xilinx Kria KV260 inference

1. Load the FPGA overlay and initialize accelerator control interfaces.
2. Initialize the USB camera and display pipeline.
3. For each input frame:
4. Capture the current video frame.
5. Convert the face region to the expected input format and normalize it.
6. Transfer the input tensor to the FPGA accelerator through DMA.
7. Trigger accelerator execution through the control interface.
8. Retrieve the output tensor from DMA.
9. Decode the predicted landmark coordinates.
10. Overlay the landmarks on the output frame.
11. Display the annotated frame and update FPS statistics.
12. End loop.

## 4. Results and Discussion

The results are organized to mirror the two-stage methodology introduced in Section 3. Stage I reports the software-stage behaviour of the proposed MobileNetV2-based facial landmark detector, with emphasis on quantization accuracy, model compactness, and the trade-off between PTQ and QAT. Stage II reports the hardware-stage behaviour of the quantized model after FPGA realization on the AMD/Xilinx Kria KV260, with emphasis on throughput, power consumption, energy efficiency, and FPGA resource utilization under the standardized runtime pipeline.

### 4.1. Software-Stage Quantization Performance

In the software stage, experiments are conducted on the WFLW benchmark using 7,500 images for training and 2,500 for testing, with 10% of the training split reserved for validation. The detector is implemented in PyTorch and trained using the weighted pose-aware objective with auxiliary Euler-angle supervision during training only. For both training and inference, the face region is converted to grayscale, normalized, and resized to 112×112×1, thereby matching the input configuration defined in Section 3. Representative qualitative examples are shown in Fig. 6.

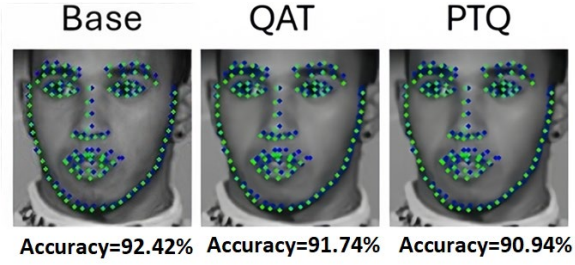


**Figure 6.** Software-stage qualitative comparison of landmark predictions for the FP32 baseline, QAT-INT8 model, and PTQ-INT8 model

Figure 6. provides a qualitative comparison of landmark predictions produced by the FP32 baseline, the QAT-INT8 model, and the PTQ-INT8 model. In each example, the green points denote ground-truth landmarks, and the blue points denote predicted landmarks. The FP32 model shows the closest visual alignment to the ground truth, while the QAT model preserves a similar prediction pattern with only minor deviations. The PTQ model shows larger displacement in several landmark groups, indicating that calibration alone is less effective at compensating for the quantization sensitivity of the customized MobileNetV2 backbone. These visual observations are consistent with the quantitative comparison reported in Table 2.

Table 2. Software-stage comparison of FP32, PTQ-INT8, and QAT-INT8 models

| Model | Accuracy (%) | Model size (MB) | Quantization time | Compression ratio |
|---|---|---|---|---|
| FP32 baseline | 92.42 | 6.59 | – | 1× |
| PTQ (INT8) | 90.94 | 1.65 | ~10 min | 4× |
| QAT (INT8) | 91.74 | 1.65 | ~180 min | 4× |

As summarized in Table 2, both INT8 methods reduce the model size from 6.59 MB to 1.65 MB, corresponding to a 4× compression ratio. In terms of landmark accuracy, the FP32 baseline yielded an accuracy of 92.42%, while the QAT-INT8 and PTQ-INT8 variants achieved 91.74% and 90.94%, respectively. Thus, both quantized variants remain practically usable, but QAT preserves accuracy more effectively than PTQ, reducing the accuracy gap to the FP32 baseline from 1.48 percentage points to 0.68 percentage points. This behaviour is consistent with the Stage I expectation that QAT better absorbs quantization noise in MobileNetV2-style inverted residual blocks.

The benefit of QAT is accompanied by a higher optimization cost. PTQ requires only approximately 10 min of calibration, whereas QAT requires approximately 180 min of fine-tuning. This result clarifies the intended roles of the two quantization strategies in the proposed framework: PTQ is attractive when rapid deployment and minimal retraining overhead are the primary objectives, whereas QAT is preferable when additional optimization time is acceptable in exchange for stronger accuracy retention. The software-stage results therefore establish a clear trade-off among accuracy, compression, and quantization cost.

## 4.2. Deployment-Stage Runtime Performance Across Platforms

To assess practical deployment behaviour, a cross-platform runtime comparison is conducted on five representative platforms: RTX 4060, Intel Core i9, Jetson Nano, Raspberry Pi 4, and AMD/Xilinx Kria KV260. All platforms process the same camera stream under a functionally standardized pipeline with comparable preprocessing and post-processing stages, whereas the execution backend remains platform specific. For this reason, Table 3 should be interpreted as a deployment-stage runtime comparison rather than as a strictly identical software benchmark.

Table 3. Deployment-stage cross-platform comparison of facial landmark inference across reference software platforms and the KV260 FPGA realization

| | Platform | FPS | Power (W) | FPS/W | Accuracy (%) |
|---|---|---|---|---|---|
| Software | GPU RTX 4060 2.46 GHz | 430 | 110 | 3.91 | 92.42 |
| | CPU Intel i9 3.20 GHz | 108 | 80 | 1.35 | 92.42 |
| | Jetson Nano 475 MHz | 8.95 | 10 | 0.90 | 92.42 |
| | Raspberry Pi 4 600 MHz | 6 | 7 | 0.86 | 92.42 |
| Hardware | Kria KV260 400 MHz | 30 | 3 | 10 | 91.74 |

The results in Table 3 reveal a clear separation between maximum throughput and energy-efficient real-time operation. While the RTX 4060 delivered the maximum raw throughput (430 FPS), the Intel Core i9 followed at 108 FPS. These platforms are preferable when absolute frame rate is the dominant objective, but they do so at substantially higher power levels, namely 110 W and 80 W, respectively. In contrast, the Kria KV260 achieves 30 FPS at only 3 W, which is sufficient for real-time facial landmark detection in the evaluated setup and yields the highest measured energy efficiency among the tested platforms at 10 FPS/W.

The embedded reference platforms operate at the lower end of the performance spectrum. The Jetson Nano reaches approximately 8.95 FPS at 10 W, corresponding to 0.90 FPS/W, while the Raspberry Pi 4 reaches approximately 6 FPS at 7 W, corresponding to 0.86 FPS/W. Compared with these platforms, the KV260 simultaneously improves throughput and operates within a smaller power envelope, making it the most favourable operating point among the tested embedded platforms for always-on edge deployment. Within the evaluated setup, these results indicate that the FPGA realization offers the best compromise between real-time performance and power efficiency, whereas the GPU remains the preferred option when throughput dominates all other considerations.

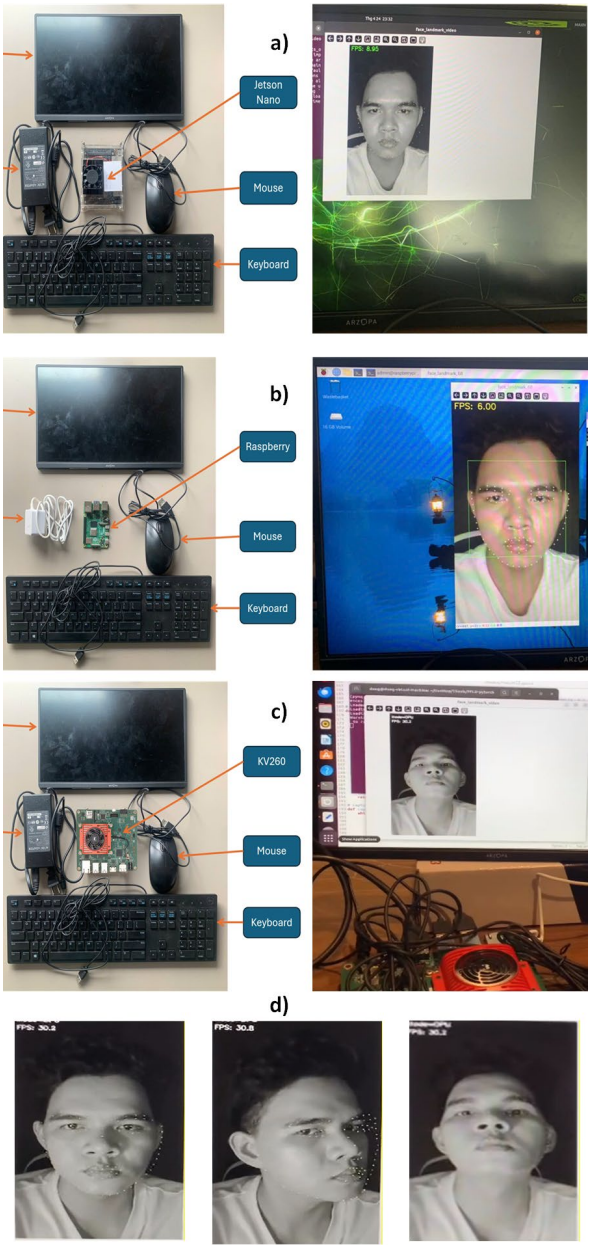


**Figure 7.** Deployment-stage experimental setups and live demonstrations of the proposed facial-landmark detection system on representative edge platforms: (a) NVIDIA Jetson Nano, (b) Raspberry Pi 4, (c) AMD/Xilinx Kria KV260 FPGA realization, and (d) qualitative landmark-detection results in challenging scenarios

Accuracy should be interpreted together with this deployment-stage comparison. The software platforms report 92.42% accuracy, while the KV260 reports 91.74%,

reflecting the expected effect of INT8 deployment rather than a change in task or evaluation protocol. In other words, the FPGA result represents the quantized model obtained in Stage I and executed in Stage II. The accuracy reduction is modest relative to the substantial gain in energy efficiency, which is precisely the trade-off targeted by the proposed hardware-aware design.

## 4.3. Deployment-Stage Behaviour in Challenging Scenarios

Figure 7. illustrates the deployment-stage experimental setups and representative live demonstrations of the proposed facial-landmark detection system on three edge-computing platforms, namely, the NVIDIA Jetson Nano, Raspberry Pi 4, and AMD/Xilinx Kria KV260, as shown in Fig. 7(a)–(c). These results verify the successful realization of the proposed system across heterogeneous hardware platforms. Furthermore, the qualitative examples presented in Fig. 7(d) provide additional evaluation of the KV260-based implementation under more challenging operating conditions, including large pose variations and low-illumination environments. Despite these adverse conditions, the system continues to deliver reliable landmark localization, thereby demonstrating its practical applicability in realistic deployment scenarios. It is also noteworthy that the KV260 implementation maintains a throughput of 30.8 FPS, indicating that real-time operation is preserved even in the presence of increased visual complexity. Collectively, these observations provide qualitative support for the practical robustness and deployment readiness of the proposed system.

Table 4. Hardware-stage FPGA resource utilization of the KV260 facial-landmark accelerator

| Resource | Utilization | Available | % |
|---|---|---|---|
| LUT | 17309 | 117120 | 14.8 |
| LUTRAM | 1483 | 57600 | 2.6 |
| FF | 19101 | 234240 | 8.1 |
| BRAM | 23.50 | 144 | 16.3 |
| DSP | 56 | 1248 | 4.5 |
| BUFG | 6 | 352 | 1.7 |
| MMCM | 1 | 4 | 25.0 |

## 4.4. Hardware-Stage FPGA Resource Utilization

Post-implementation hardware utilization is summarized in Table 4. The hardware-stage implementation on the AMD/Xilinx Kria KV260 reveals a compact accelerator footprint. The deployed design uses 17,309 LUTs, 19,101 FFs, 23.50 BRAMs, and 56 DSPs, corresponding to 14.8%, 8.1%, 16.3%, and 4.5% of the available resources, respectively. Additional control resources, including 6 BUFGs and 1 MMCM, also remain within a modest fraction of the platform budget. These results confirm that the proposed detector is not only executable on the KV260, but also occupies a relatively small fraction of the available programmable logic.

The observed hardware utilization remains modest, and this characteristic is significant from two implementation perspectives. First, it confirms that the achieved operating point of 30 FPS at 3 W is realized without approaching resource exhaustion on the target FPGA platform, thereby reinforcing the practicality of the proposed design for embedded deployment. Second, it indicates that sufficient architectural headroom is retained for subsequent scaling, such as increasing the degree of parallelism, instantiating multiple accelerator engines, or integrating the proposed detector with additional perception modules in a larger driver-monitoring framework. A particularly important observation is the DSP utilization of only 4.5%. This result suggests that the INT8 quantized implementation, together with the adopted compiler mapping strategy, is realized primarily through LUT-oriented computation and efficient dataflow scheduling rather than through extensive dependence on dedicated multiply–accumulate resources. From an implementation standpoint, such a utilization pattern is consistent with the intended energy-efficient operating regime of fixed-point FPGA inference. To further relate this favourable utilization profile to the underlying hardware-aware design methodology, a controlled post-implementation comparison between ReLU6 and ReLU is presented in Table 5.

As reported in Table 5, the comparison between ReLU6 and ReLU was performed under tightly controlled post-implementation conditions on the KV260 platform. Specifically, both designs were evaluated using the same 150 MHz timing constraint and identical synthesis and implementation settings, with the activation function being the only variable. This comparison is of relevance because the replacement of ReLU6 with ReLU constitutes one of the explicit hardware-aware modifications introduced in Section 3. From the FPGA hardware perspective, ReLU is implemented using only a lower-bound comparison, whereas ReLU6 requires an additional upper-bound comparison together with a saturation operation. On LUT-dominated FPGA fabrics, this added logic may increase resource consumption and contribute to a longer critical combinational path. Accordingly, the results in Table 5 provide direct implementation evidence that the adoption of ReLU improves hardware efficiency, thereby supporting the effectiveness of the proposed hardware-aware optimization strategy.

The results clearly Favor REL over ReLU6 for the target FPGA realization. ReLU6 requires 7 LUTs and 15 FFs, whereas ReLU requires only 1 LUT and 16 FFs. Thus, replacing ReLU6 with ReLU reduces LUT usage by 6 LUTs, corresponding to an 85.7% reduction in

combinational logic for the activation block, while leaving register cost essentially unchanged. This indicates that the main benefit comes from reduced combinational complexity rather than from changes in sequential storage.

Table 5. Post-implementation comparison of ReLU6 and ReLU on the KV260 at a 150 MHz timing constraint: resource usage, timing metrics, and estimated $Fmax$

| Activation | LUTs | Registers (FF) | WNS @150 MHz | Worst max-path | Freq. (MHz) |
|---|---|---|---|---|---|
| ReLU6 | 7 | 15 | 3.446 ns | 2.733 ns | 310.6 |
| ReLU | 1 | 16 | 3.622 ns | 2.257 ns | 328.6 |
| Δ (ReLU − ReLU6) | −6 | +1 | +0.176 ns | −0.476 ns | +18.0 (+5.8%) |

Using

$$F_{max} \approx \frac{1000}{T_{req} - WNS} \quad (7)$$

with $\boldsymbol{Treq}$=6.67 ns for the 150 MHz timing constraint, the estimated maximum operating frequency increases from 310.6 MHz for ReLU6 to 328.6 MHz for ReLU, corresponding to a gain of approximately 18.0 MHz (5.8%). These results further confirm that, under the same post-implementation conditions on the KV260, ReLU provides a more favorable hardware realization than ReLU6. In particular, the worst negative slack (WNS) improves from 3.446 ns to 3.622 ns, while the worst max-path datapath delay decreases from 2.733 ns to 2.257 ns. Therefore, replacing ReLU6 with ReLU reduces combinational complexity, shortens the critical path, and improves timing margin. This post-implementation evidence directly supports the hardware-aware activation choice adopted in the customized MobileNetV2 backbone.

Taken together, the results provide a coherent view across the two stages of the study. In Stage I, INT8 quantization substantially reduces model size, while QAT preserves accuracy more effectively than PTQ when additional fine-tuning is feasible. In Stage II, the resulting quantized model is deployed on the Kria KV260 with real-time throughput, low power consumption, and the highest measured energy efficiency among the evaluated platforms. The hardware utilization results show that this performance is achieved with a modest FPGA footprint, while the controlled post-implementation comparison between ReLU6 and ReLU justifies one of the key hardware-aware design choices at the operator level. Collectively, these findings position the proposed system as a practical deployment-oriented solution that bridges lightweight landmark detection and low-power FPGA realization.

Table 6. Contextual positioning of representative prior studies and the present work in relation to facial landmark detection and edge deployment

| Ref. | Main focus | Deployment relevance | Platform class | Relevance to this study |
|---|---|---|---|---|
| [6] | PFLD-based facial landmark detection | Lightweight landmark modelling | GPU-class | Compact landmark baseline in the literature. |
| [7] | Pose-assisted alignment | Pose robustness | Algorithmic | Supports pose-aware landmark localization. |
| [8] | Boundary-aware alignment | Structural landmark guidance | Algorithmic | Supports boundary-guided localization. |
| [26] | Integer quantization | Low-precision inference | General | Supports quantization methodology. |
| [30] | Edge-AI driver monitoring | Real-time edge deployment | Edge-AI hardware | Motivates hardware-aware deployment. |
| [31] | TinyML-based driver monitoring | Ultra-low-power inference | MCU-class | Represents the TinyML efficiency regime. |
| This work | Facial landmark detection for edge driver-monitoring context | INT8 FPGA deployment | FPGA edge | Combines quantization analysis and FPGA realization. |

*Note: Table 6 is intended as a contextual positioning summary rather than a strict head-to-head benchmark. The cited studies differ in task definition, dataset, preprocessing pipeline, hardware platform, precision setting, and deployment scope.*

*Accordingly, the table is used to situate the present study within representative directions in the literature, rather than to claim direct numerical superiority*

The significance of the present results can be understood more clearly through the contextual positioning summarized in Table 6. Specifically, the table situates the proposed study alongside several representative research directions, including lightweight facial landmark modelling, pose-robust alignment, structural landmark guidance, integer quantization, edge-AI driver monitoring, and TinyML-oriented ultra-low-power inference. From this perspective, earlier landmark-detection studies such as PFLD, pose-assisted alignment, and boundary-aware alignment remain directly relevant to the present work because they establish important algorithmic foundations for efficient and robust facial landmark localization. In parallel, prior studies on integer quantization provide the methodological basis for low-precision inference [28], whereas edge-AI driver-monitoring and TinyML-oriented systems motivate the importance of real-time and energy-efficient operation under strict resource constraints [29], [30], [31]. Relative to these representative directions, the distinguishing feature of the present study is that it combines them within a single deployment-oriented framework by integrating a lightweight MobileNetV2-based facial landmark detector, INT8 PTQ/QAT analysis, and FPGA realization on the AMD/Xilinx Kria KV260. In this sense, the contribution of the study lies not in proposing an entirely new landmark model or a new quantization algorithm in isolation, but in explicitly linking software-stage quantization analysis with hardware-stage FPGA deployment for an edge driver-monitoring context.

At the same time, Table 6 also clarifies the proper scope of this comparison. As explicitly noted in the manuscript, the table is intended as a contextual positioning summary rather than as a strict head-to-head benchmark, since the cited studies differ substantially in task definition, dataset, preprocessing pipeline, hardware platform, precision setting, and deployment scope. Accordingly, the controlled quantitative evidence of the present work remains that reported in Tables 2 and 3. Within that evaluated scope, the study shows that INT8 quantization reduces the model size from 6.59 MB to 1.65 MB, that QAT preserves accuracy more effectively than PTQ, and that the resulting design can be realized on the AMD/Xilinx Kria KV260 with real-time throughput and low power consumption. Therefore, the comparative value of the present work is best understood as filling a specific gap at the intersection of lightweight facial landmark detection, low-precision optimization, and FPGA edge deployment, rather than as claiming direct numerical superiority over the prior studies listed in Table 6.

Within the evaluated scope, the present results define a practical deployment-oriented baseline rather than a complete endpoint. Although the proposed two-stage framework shows that a quantized MobileNetV2-based detector can preserve acceptable accuracy while enabling real-time, low-power execution on the AMD/Xilinx Kria KV260, future work should further strengthen the validation through a more tightly controlled cross-platform benchmark, a dedicated robustness analysis under pose, occlusion, and illumination variations, and the exploration of alternative lightweight backbones and mixed-precision deployment schemes [32]. Even so, the current framework already provides a technically grounded foundation for FPGA-based facial landmark detection at the edge and for downstream facial-geometry-driven applications in driver-monitoring scenarios, such as blink-related state analysis [33].

## 5. Conclusion

This study presents a two-stage hardware-aware framework for MobileNetV2-based facial landmark detection at the edge. In the software stage, INT8 quantization reduces model size from 6.59 MB to 1.65 MB, and QAT preserves accuracy more effectively than PTQ (91.74% vs. 90.94%) relative to the FP32 baseline (92.42%). In the hardware stage, the AMD/Xilinx Kria KV260 realization achieves approximately 30 FPS at approximately 3 W while using a modest fraction of available FPGA resources. These results support the feasibility of real-time, low-power facial landmark detection on an FPGA edge platform within the evaluated setup. Future work should focus on unified cross-platform benchmarking, robustness evaluation under more challenging real-world conditions, and the investigation of alternative lightweight backbones and mixed-precision deployment schemes.

### Acknowledgements

This research was funded by Ho Chi Minh City University of Technology and Engineering, Vietnam, under grant No. T2026-146.

### Conflict of Interest

The authors declare no conflict of interest.

## References

[1] Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the Internet of Things. *In: Proceedings of the First ACM Workshop on Mobile Cloud Computing*; 2012; p. 13-16.

[2] Satyanarayana M. The emergence of edge computing. *Computer*. 2017;50(1):30-39.

[3] Singh KD, Singh P. Fog cloud computing and IoT integration for AI-enabled autonomous systems in robotics. *EAI Endorsed Trans AI and Robotics*. 2024;3.

[4] National Highway Traffic Safety Administration. Traffic safety facts research note: drowsy driving. Washington (DC): U.S. Department of Transportation; 2017. Report No.: DOT HS 812 446.
[5] Uddin NMI, et al. The face detection/recognition, perspective and obstacles in robotic: a review. *EAI Endorsed Trans AI and Robotics*. 2022;1(1):e14.
[6] Guo X, Li S, Yu J, Zhang J, Ma J, Ma L, Liu W, Ling H. PFLD: a practical facial landmark detector. *arXiv* [Preprint]. 2019;arXiv:1902.10859.
[7] Yang H, Mou W, Zhang Y, Patras I, Gunes H, Robinson P. Face alignment assisted by head pose estimation. *In: Proceedings of the British Machine Vision Conference;* 2015.
[8] Wu W, Qian C, Yang S, Wang Q, Cai Y, Zhou Q. Look at boundary: a boundary-aware face alignment algorithm. *In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; 2018; p. 2129-2138.
[9] 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*; 2018;. p. 4510-4520.
[10] Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: a white paper. *arXiv* [Preprint]. 2018;arXiv:1806.08342.
[11] Khan WZ, Ahmed E, Hakak S, Yaqoob I, Ahmed A. Edge computing: a survey. *Future Gener Comput Syst*. 2019; 97: 219-235.
[12] Nagel M, van Baalen M, Blankevoort T, Welling M. Data-free quantization through weight equalization and bias correction. *In: Proceedings of the IEEE/CVF International Conference on Computer Vision*; 2019;. p. 1325-1334.
[13] Zhao R, Hu Y, Dotzel J, De Sa C, Zhang Z. Improving neural network quantization without retraining using outlier channel splitting. *In: Proceedings of the 36th International Conference on Machine Learning*; 2019;. p. 7543-7552.
[14] Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; 2018;. p. 2704-2713.
[15] Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J. Optimizing FPGA-based accelerator design for deep convolutional neural networks. *In: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*; 2015;. p. 161-170.
[16] Nurvitadhi E, Venkatesh G, Sim J, Marr D, Huang R, Ong JGH, Liew YT, Srivatsan K, Moss D, Subhaschandra S, Boudoukh G. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? *In: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*; 2017;. p. 5-14.
[17] Tragoudaras A, Stoikos P, Fanaras K, Tziouvaras A, Floros G, Dimitriou G, Kolomvatsos K, Stamoulis G. Design space exploration of a sparse MobileNetV2 using high-level synthesis and sparse matrix techniques on FPGAs. *Sensors*. 2022;22(12):4318.
[18] Sutjiadi R, Sendari S, Herwanto HW, Kristian Y. Leveraging synthetic mammograms to enhance deep-learning performance for breast cancer classification using EfficientNetV2L architecture. *EAI Endorsed Trans AI and Robotics*. 2025;4:. doi:10.4108/airo.9749.
[19] Nagel M, Fournarakis M, Amjad RA, Bondarenko Y, van Baalen M, Blankevoort T. A white paper on neural network quantization. *arXiv* [Preprint]. 2021;arXiv:2106.08295.
[20] Cai Y, Yao Z, Dong Z, Gholami A, Mahoney MW, Keutzer K. ZeroQ: a novel zero-shot quantization framework. *In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; 2020;. p. 13169-13178.
[21] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Quantized neural networks: training neural networks with low-precision weights and activations. *J Mach Learn Res*. 2018;18(187):1-30.
[22] He Y, Liu J, Wu W, Zhou H, Zhuang B. EfficientDM: efficient quantization-aware fine-tuning of low-bit diffusion models. *In: Proceedings of the International Conference on Learning Representations*; 2024.
[23] Wuraola A, Patel N. Resource efficient activation functions for neural network accelerators. *Neurocomputing*. 2022;482:163-185. doi:10.1016/j.neucom.2021.11.032.
[24] Jourabloo A, Liu X. Pose-invariant 3D face alignment. *In: Proceedings of the IEEE International Conference on Computer Vision*; 2015.
[25] Kumar A, Chellappa R. Disentangling 3D pose in a dendritic CNN for unconstrained 2D face alignment. *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*; 2018.
[26] Wu H, Judd P, Zhang X, Isaev M, Micikevicius P. Integer quantization for deep learning inference: principles and empirical evaluation. *arXiv* [Preprint]. 2020;arXiv:2004.09602.
[27] AMD. Vitis AI user guide (UG1414). Version 3.5.: AMD; 2023 Sep 28.
[28] Weng O. Neural network quantization for efficient inference: a survey. *arXiv* [Preprint]. 2021;arXiv:2112.06126.
[29] Tanama F, Xiang H, Cheung WK, Liang X, Zhou B. Quantized distillation: optimizing driver activity recognition models for resource-constrained environments. *arXiv* [Preprint]. 2023;arXiv:2311.05970.
[30] Hariharan M, Varior R, Karunakaran P. Real-time driver monitoring systems on edge AI device. *arXiv* [Preprint]. 2023;arXiv:2304.01555.
[31] Ghasemi M, Samie F, Ejlali A. A TinyML-based driver drowsiness detection system using a novel low-power CNN architecture. *IEEE Trans Circuits Syst II Exp Briefs*. 2022;69(7):3257-3261.
[32] 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*; 2018;. p. 116-131.
[33] Cech J, Soukupova T. Real-time eye blink detection using facial landmarks. Prague: Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague; 2016. p. 1-8.