IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 April 2026, accepted 12 May 2026, date of publication 15 May 2026, date of current version 2 June 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3693881

RESEARCH ARTICLE

Hybrid CNN-KAN Architectures for Ultra-Compact Person Detection on IoT Devices: A Systematic Evaluation of Quantization Strategies

DANIELE FAGGI[1], OLEKSANDR KUZNETSOV[1,2], (Member, IEEE), STEFANO MEREU[3,4], ALESSANDRO GALDELLI[5], (Member, IEEE), EMANUELE FRONTONI[6], (Member, IEEE), AND MARCO ARNESANO[1], (Member, IEEE)

[1]Department of Theoretical and Applied Sciences (DISTA), eCampus University, 22060 Novedrate, Italy
[2]Department of Intelligent Software Systems and Technologies, School of Computer Science and Artificial Intelligence, V. N. Karazin Kharkiv National University, 61022 Kharkiv, Ukraine
[3]Department of Informatics, Bioengineering, Robotics and Systems Engineering (DIBRIS), University of Genoa, 16145 Genoa, Italy
[4]Istituto Italiano di Tecnologia, 16163 Genoa, Italy
[5]Department of Information Engineering, Marche Polytechnic University, 60131 Ancona, Italy
[6]Department of Political Sciences, Communication and International Relations, University of Macerata, 62100 Macerata, Italy

Corresponding author: Oleksandr Kuznetsov (oleksandr.kuznetsov@uniecampus.it)

This work was supported by the SMARTEST Research Center at eCampus University.

ABSTRACT Deploying visual person detection on resource-constrained Internet of Things (IoT) devices demands models that balance classification accuracy against severe memory and latency budgets. Kolmogorov–Arnold Networks (KANs), which replace fixed activation functions with learnable B-spline edges, have recently attracted interest as expressive alternatives to conventional multi-layer perceptrons, yet their suitability for embedded inference remains largely unexplored. In this work we couple a MobileNetV3-Small feature extractor with compact KAN classifiers of varying depth and evaluate the resulting hybrid architectures on the Visual Wake Words person-detection benchmark. We train 46 model variants spanning eight KAN topologies—including three ultra-compact heads with as few as two hidden neurons—six width multipliers, and three quantization strategies: native FP32, hybrid INT8 (backbone quantized, KAN in FP32), and full TFLite INT8. The hybrid INT8 pathway achieves up to 3.3× compression with accuracy losses below 0.25 percentage points, while TFLite INT8 conversion yields sub-megabyte models (0.31–1.55 MB) suitable for devices such as Seeed XIAO and ESP32-S3. We further document a notable finding: KAN layers are stored as INT64 rather than INT8 inside TFLite flatbuffers, limiting compression to 1.9–2.9× instead of the expected ∼4×. We address this limitation through an operator decomposition strategy that replaces non-standard operations (HardSigmoid, Gather) with equivalent compositions of TFLite-native operators, eliminating all INT64 and FP32 tensors from the converted model and enabling full XNNPACK delegation. Preliminary on-device measurements on an ESP32-S3 microcontroller and a Raspberry Pi 3 confirm deployment feasibility, with the smallest model (0.26 MB) fitting entirely within internal SRAM and achieving 83.1% accuracy on-device—matching the desktop result to within 0.07 pp after resolution of a per-channel quantization mismatch in the TFLite Micro runtime. Our findings offer concrete deployment guidelines for practitioners integrating KAN-based classifiers into resource-constrained inference pipelines and identify quantization of spline-based layers as a tractable engineering challenge for the TinyML community.

INDEX TERMS Internet of things, INT8 quantization, Kolmogorov–Arnold networks, MobileNetV3, model compression, person detection, TinyML, visual wake words.

The associate editor coordinating the review of this manuscript and approving it for publication was Mostafa M. Fouda.

I. INTRODUCTION

The proliferation of battery-powered cameras and microcontroller-based sensor nodes has created a pressing

need for on-device visual intelligence that operates within memory envelopes of a few hundred kilobytes to a few megabytes. Person detection—determining whether a human is present in an image frame—represents one of the most commercially relevant tasks in this space, underpinning always-on surveillance triggers, occupancy sensing for building automation, and privacy-preserving analytics at the network edge. The Visual Wake Words (VWW) benchmark, introduced by Chowdhery et al. [1], distils this task into a binary classification problem derived from MSCOCO [2], providing a standardised arena for comparing models under strict resource constraints.

State-of-the-art results on VWW have been driven by neural architecture search (NAS). MCUNet [3] jointly optimised architecture and inference scheduling to reach 87.4% accuracy within 1 MB of Flash, and its successor MCUNetV2 [4] pushed accuracy beyond 90% under only 32 kB of SRAM through patch-based execution. MobileNetV3 [5], while not specifically designed for microcontrollers, offers a favourable accuracy-to-parameter ratio that makes its ''Small'' variant a popular backbone for embedded vision pipelines. All of these architectures rely on conventional multi-layer perceptron (MLP) heads—typically one or two fully connected layers with ReLU or hard-swish activations—for the final classification stage.

Kolmogorov–Arnold Networks (KANs), proposed by Liu et al. [6], take a fundamentally different approach: rather than fixing activation functions on nodes and learning weight matrices, KANs place learnable univariate functions on edges and eliminate linear weights entirely. Each edge is parameterised as a B-spline, and the resulting architecture draws theoretical motivation from the Kolmogorov–Arnold superposition theorem [7], [8]. Early empirical studies demonstrated that KANs can match or outperform much larger MLPs in scientific computing tasks involving symbolic regression and PDE solving. Subsequent work has extended the idea to convolutional settings (convolutional KANs) [9], wavelet-based variants (Wav-KAN) [10], and efficient implementations that reduce the computational overhead of spline evaluation [11].

Despite this rapid progress, a significant gap persists between KAN research and practical deployment on constrained hardware. Three specific problems remain open:

1) *Quantization compatibility.* B-spline operations do not map onto the INT8 kernels available in standard inference runtimes such as TFLite and XNNPACK. Whether a KAN classifier can survive post-training quantization without catastrophic accuracy loss is not well characterised.
2) *Architecture scaling.* It is unclear how KAN depth and width interact with backbone width multipliers when the combined model must fit a specific memory budget.
3) *Deployment realism.* Most KAN papers report FP32 results on desktop GPUs; none, to our knowledge, have traced the full conversion pipeline from PyTorch through TFLite to an IoT-class target and documented the engineering obstacles encountered along the way.

In this paper we address these gaps through a systematic experimental campaign. Our contributions are as follows:

- We design a hybrid CNN-KAN architecture that pairs MobileNetV3-Small with B-spline KAN classifiers at eight topology levels—from ultra-compact 2-neuron heads to 4-layer cascades—and six backbone width multipliers ($\alpha = 0.25$ to 1.0), yielding 46 distinct model configurations.
- We implement and compare three quantization pipelines—FP32, hybrid INT8, and full TFLite INT8—and report detailed accuracy, size, latency, parameter count, and peak activation memory measurements for every variant.
- We document two important findings for the TinyML community: (i) KAN spline parameters are stored as INT64 tensors inside TFLite flatbuffers rather than the expected INT8, limiting compression; and (ii) initial XNNPACK delegate failures for $\alpha = 1.0$ models were traced to an environment configuration issue rather than an architectural incompatibility, demonstrating that KAN-based TFLite models are compatible with XNNPACK acceleration when the toolchain is properly configured.
- We propose and validate an operator decomposition strategy that replaces non-quantizable operations (HardSigmoid, Gather) with compositions of standard TFLite operators, fully eliminating the INT64 storage anomaly and reducing the operator count from 28 to 13. We validate the resulting models on-device on an XIAO ESP32S3 Sense microcontroller and a Raspberry Pi 3.
- We provide concrete deployment guidelines mapping each model to a specific IoT device class, from sub-500 kB nodes (Seeed XIAO, Nordic nRF52840) through mid-range SoCs (ESP32-S3) to Linux-capable boards (Raspberry Pi Zero 2W, Raspberry Pi 4).

The remainder of this paper is organised as follows. Section II introduces the mathematical foundations of KANs and the engineering constraints of TinyML. Section III surveys lightweight CNN design, KAN variants, and model compression techniques. Section IV details our architecture, training protocol, and quantization pipelines. Section V presents experimental results across all 46 configurations. Section VI interprets findings, situates them relative to state-of-the-art benchmarks, and discusses limitations. Section VIII offers concluding remarks and outlines future work. Section VII provides a consolidated visual overview of the complete experimental campaign.

II. BACKGROUND

This section provides the conceptual foundations necessary to appreciate the design choices and experimental results presented later. We first recall the Kolmogorov–Arnold representation theorem and its modern neural-network incarnation, then summarise the constraints that govern

deep-learning inference on microcontroller-class hardware, and finally discuss the quantization paradigms relevant to our experiments.

A. FROM THE KOLMOGOROV–ARNOLD THEOREM TO KAN LAYERS

The Kolmogorov–Arnold superposition theorem [7] states that every continuous multivariate function $f\colon [0,1]^n \to \mathbb{R}$ can be written as

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right), \tag{1}$$

where $\phi_{q,p}\colon [0,1] \to \mathbb{R}$ and $\Phi_q\colon \mathbb{R} \to \mathbb{R}$ are continuous univariate functions. The theorem implies, in a profound sense, that multivariate complexity can always be decomposed into compositions and sums of one-dimensional maps. While the original proof is non-constructive and the inner functions $\phi_{q,p}$ may be highly irregular, the theorem provides a theoretical foundation for architectures that represent high-dimensional mappings through compositions of learned univariate transformations.

Liu et al. [6] observed that Equation (1) naturally corresponds to a two-layer network whose edges—rather than nodes—carry learnable activation functions, and generalised this idea to arbitrary depth. In a KAN layer connecting n_{in} inputs to n_{out} outputs, the transformation is

$$\mathbf{y} = \sum_{i=1}^{n_{\text{in}}} \varphi_i(x_i), \quad \varphi_{i,j}\colon \mathbb{R} \to \mathbb{R}, \tag{2}$$

where each $\varphi_{i,j}$ is parameterised as a cubic B-spline with G grid intervals and spline order k:

$$\varphi(x) = w_b\,\mathrm{SiLU}(x) + w_s\,B(x), \quad B(x) = \sum_m c_m\,B_m^k(x). \tag{3}$$

Here w_b and w_s are learnable scalars that balance a residual activation (SiLU) against the spline basis B_m^k, and c_m are the spline coefficients. A KAN layer with n_{in} inputs, n_{out} outputs, G grid points, and order k therefore has

$$P_{\text{KAN}} = n_{\text{in}} \cdot n_{\text{out}} \cdot (G + k + 1) + 2\,n_{\text{in}} \cdot n_{\text{out}} \tag{4}$$

spline and scalar parameters. This count grows quadratically with layer width, which has important implications for model size that we examine in Section V. For our default configuration ($G = 5$, $k = 3$), each edge contributes $5 + 3 + 1 + 2 = 11$ learnable parameters, compared with a single weight in a standard linear layer. This parameter amplification explains why even narrow KAN heads (e.g., $[8 \to 2]$) can capture non-trivial decision boundaries.

B. TINYML CONSTRAINTS AND THE VWW BENCHMARK

Microcontroller units (MCUs) such as the ARM Cortex-M4 and Cortex-M7 families typically offer 256–512 kB of SRAM and 1–2 MB of Flash storage. Inference on such devices must satisfy three simultaneous constraints:

1) *Model size* (Flash): the serialised model, including all weights and metadata, must fit within available non-volatile memory.
2) *Peak activation memory* (SRAM): the largest intermediate tensor that must coexist in RAM during any single inference pass. We denote this quantity "Peak (KB)" throughout this paper, computed by profiling the computational graph and recording the maximum resident memory across all operator evaluations. For a given layer ℓ, the activation memory equals the product of the output tensor dimensions multiplied by the element size in bytes; the peak is $\max_\ell \mathrm{mem}(\ell)$. This metric is critical because, on bare-metal MCUs, SRAM is typically the binding constraint: even a model that fits in Flash may be undeployable if its peak activation exceeds the available SRAM budget.
3) *Latency*: a per-frame budget of tens to hundreds of milliseconds, depending on whether the application is always-on (low duty cycle) or interactive.

The VWW dataset [1] provides a controlled benchmark for evaluating models against these constraints. Derived from the MSCOCO 2014 training split, it contains 115 927 images labelled as `person` (at least one bounding box with area $> 5\%$ of the image) or `non-person`, and is typically evaluated at input resolutions of 96×96 or 224×224 pixels. The validation split comprises 8 059 images. State-of-the-art results within the 250 kB Flash budget fall in the 85–90% accuracy range; relaxing the budget to 1–2 MB permits accuracies above 90%.

C. QUANTIZATION PARADIGMS

Post-training quantization (PTQ) converts FP32 weights and activations to lower-precision integers, reducing both model size and inference latency on hardware that supports integer arithmetic. The standard INT8 quantization maps a floating-point tensor x to

$$\hat{x} = \mathrm{clamp}\left(\left\lfloor \frac{x}{s} \right\rceil + z,\ 0,\ 255 \right), \tag{5}$$

where s is a per-tensor or per-channel scale factor and z is a zero-point offset. For convolutional and linear layers, INT8 PTQ is well supported by frameworks such as TensorFlow Lite [32] and XNNPACK. The expected size reduction is approximately $4\times$ (32 bits $\to$ 8 bits), with typical accuracy losses of 0.5–2.0 percentage points for MobileNet-class models [23], [24].

However, operators that fall outside the standard set—including the B-spline evaluations central to KANs—are typically left in FP32 or, as we discover in this work, cast to INT64, defeating the purpose of quantization. This motivates the hybrid quantization strategy explored in Section IV-C, where only the CNN backbone and linear projector are quantized while the KAN classifier retains FP32 precision.

III. RELATED WORK

A. LIGHTWEIGHT CONVOLUTIONAL ARCHITECTURES FOR MCUs

The MobileNet family [5], [15], [16] introduced depthwise separable convolutions and inverted residual blocks to dramatically reduce the parameter count of convolutional networks. MobileNetV1 [15] demonstrated that factoring standard convolutions into depthwise and pointwise operations could yield comparable accuracy at a fraction of the computational cost. MobileNetV2 [16] added inverted residual blocks with linear bottlenecks, achieving improved gradient flow and representational efficiency. MobileNetV3-Small [5], optimised through a combination of NAS and NetAdapt [17], further incorporated squeeze-and-excitation modules and hard-swish activations, achieving an ImageNet top-1 accuracy of 67.4% with only 2.5 M parameters—roughly 6.6 percentage points above MobileNetV2 at comparable latency.

For the microcontroller regime, Lin et al. [3] proposed MCUNet, which co-designs the neural architecture (TinyNAS) and inference engine (TinyEngine) to maximise accuracy under SRAM and Flash constraints. On VWW, MCUNet reached 87.4% accuracy at 0.74 MB Flash. MCUNetV2 [4] introduced patch-based inference to break the memory bottleneck of early convolutional layers, pushing accuracy beyond 90% under only 32 kB of SRAM. More recently, MicroENet [18] targeted the STM32F746 with peak memory under 100 kB, and ColabNAS [19] automated architecture design on Raspberry Pi Zero 2 in under 10 hours.

EfficientNet [21] and its TinyML-oriented derivatives [22] have also been explored, though their compound scaling strategy tends to produce models that are less MCU-friendly than the MobileNet lineage. The Wake Vision dataset [20], a 6-million-image successor to VWW with improved label quality, has recently begun to displace VWW as the community benchmark, reporting that training on Wake Vision improved MobileNetV2 accuracy by 1.93 percentage points over VWW alone.

B. KOLMOGOROV–ARNOLD NETWORKS AND VARIANTS

Since the seminal KAN paper by Liu et al. [6], a substantial body of follow-up work has emerged. The `efficient-kan` implementation by Blealtan [11] reduces the computational cost of spline evaluation by fusing grid operations, making deeper KAN architectures practically trainable. Bozorgasl and Chen [10] replaced B-splines with wavelet bases in Wav-KAN, improving frequency-domain representation at a modest parameter overhead. Bodner et al. [9] embedded KAN operations inside convolutional filters, creating convolutional KANs that can process spatial data directly. Azam and Akhtar [14] provided one of the first systematic assessments of KANs on standard computer vision benchmarks, reporting mixed results: KANs improved sample efficiency on small datasets but lagged behind MLPs on ImageNet-scale tasks.

KAN 2.0 [12] introduced tools for feature attribution, modular structure discovery, and symbolic regression, broadening the utility of KANs in scientific applications. A critical assessment by Hou et al. [13] cautioned that much of KANs' advantage stems from the flexibility of B-spline activations rather than from the Kolmogorov–Arnold architecture per se, and documented 1.36–100× computational overhead relative to comparably sized MLPs. The authors further demonstrated that standard MLPs with learnable activation functions can match KAN accuracy in most settings, questioning the practical necessity of the spline parameterisation.

Despite this rich literature, we are aware of no prior work that has deployed KAN-based classifiers on IoT hardware or attempted to quantize KAN layers for embedded inference. Our work fills this gap by providing a systematic empirical evaluation of the full training-to-deployment pipeline for hybrid CNN-KAN architectures.

C. MODEL COMPRESSION FOR EMBEDDED DEPLOYMENT

Post-training quantization (PTQ) [23], [24] and quantization-aware training (QAT) [25] are the most widely used compression techniques in industry. For MobileNet-class models, INT8 PTQ typically incurs accuracy losses of 0.5–2.0 percentage points while reducing model size by approximately 4× and enabling efficient integer-only inference. Mixed-precision approaches [26], [27] assign different bit widths to different layers, optimising a size–accuracy Pareto front; HAWQ [26] uses Hessian-based sensitivity analysis to guide bit-width assignment automatically.

Knowledge distillation [28], [29] transfers knowledge from a large teacher to a smaller student, and is complementary to quantization. Pruning [30], [31] removes redundant connections; the lottery ticket hypothesis [31] demonstrated that dense networks contain sparse subnetworks that can match the original accuracy when trained in isolation. TFLite [32], which we use as our deployment target, supports representative-dataset-calibrated PTQ and can delegate compatible operators to XNNPACK for optimised execution on ARM CPUs.

To the best of our knowledge, no prior study has combined KAN classifiers with CNN backbones and systematically evaluated multiple quantization strategies in a TinyML context. This gap motivates the present work.

IV. METHODOLOGY

A. ARCHITECTURE DESIGN

Our architecture, illustrated in Fig. 1, comprises three stages:

1) **Feature extractor.** A MobileNetV3-Small backbone [5] with width multiplier $\alpha \in \{0.25, 0.334, 0.5, 0.668, 0.75, 1.0\}$. The backbone accepts $224 \times 224 \times 3$ input images and produces a feature vector of dimensionality $\lfloor 576\alpha \rfloor$ after global average pooling. We use the PyTorch-provided ImageNet-pretrained weights where available ($\alpha = 1.0$); for non-standard width multipliers we train from random initialisation.

2) **Linear projector.** A projection block consisting of a linear layer, batch normalisation, and ReLU activation maps the backbone feature vector to the KAN input dimensionality d_{in}, which depends on the first KAN hidden width. This projector serves as a dimensionality bridge between the variable-width backbone output and the fixed KAN input.
3) **KAN classifier.** We evaluate eight KAN topologies, denoted by their hidden-layer widths: [8 → 2] (2-layer, ultra-compact), [8 → 4] (2-layer), [8 → 8] (2-layer), [16 → 4] (2-layer), [24 → 16] (2-layer), [32 → 16] (2-layer), [32 → 24 → 16] (3-layer), and [64 → 32 → 24 → 16] (4-layer). All KAN layers use cubic B-splines ($k = 3$) with $G = 5$ grid intervals, following the `pykan` [6] implementation with custom patches for ONNX export compatibility (Section IV-C). The final KAN layer outputs two logits for the person/non-person decision.

The three ultra-compact topologies ([8 → 2], [8 → 4], [8 → 8]) are designed to test the lower bound of KAN expressiveness: with only 8 input neurons and as few as 2 hidden neurons, these classifiers compress the backbone representation through an extreme bottleneck. If adequate accuracy is maintained, such topologies could enable deployment on the most memory-constrained devices.

B. DATASET AND TRAINING PROTOCOL

We train on the Visual Wake Words dataset [1], which partitions the MSCOCO 2014 training set into 115 927 images (roughly 40% person, 60% non-person). We resize all images to 224 × 224 pixels. For evaluation we use a balanced subset of 6 000 images (3 000 per class) sampled from the validation split. We apply standard augmentation: random horizontal flip ($p = 0.5$), random rotation ($\pm 10°$), and random erasing ($p = 0.05$).

All models are trained for 50 epochs using AdamW [33] with an initial learning rate of 3×10^{-3}, weight decay of 10^{-5}, and cosine annealing [34]. The batch size is 256. We employ cross-entropy loss with label smoothing ($\varepsilon = 0.1$). Training uses mixed-precision (FP16/FP32) arithmetic on GPU for efficiency. For $\alpha = 1.0$ configurations, ImageNet-pretrained backbone weights are loaded and fine-tuned end-to-end; for all other width multipliers, the backbone is trained from scratch alongside the KAN head. Each experiment is executed on a single NVIDIA GPU; the fastest configurations ($\alpha = 0.5$, 2-layer KAN) converge in approximately one hour, while the largest ($\alpha = 1.0$, 4-layer KAN) require roughly five hours.

C. QUANTIZATION STRATEGIES

We implement and compare three quantization pipelines:

1) FP32 (BASELINE)

The model is serialised in its native PyTorch format. This pathway yields the largest model size but serves as the accuracy reference. All 16 FP32 configurations serve as baselines against which quantized variants are compared.

2) HYBRID INT8

We apply symmetric INT8 static quantization to the MobileNetV3-Small backbone and the linear projector using PyTorch's `torch.ao.quantization` API with a calibration set of 1 000 training images. The KAN classifier is kept in FP32 because PyTorch's quantisation observers do not support the B-spline operations. Inference consists of an INT8 forward pass through the backbone, a dequantise step, and an FP32 pass through the KAN layers. This mixed-precision strategy exploits the fact that the backbone accounts for >95% of total parameters in most configurations, so quantizing it alone yields substantial compression.

3) TFLite INT8

We export the full FP32 model through the chain PyTorch → ONNX → TensorFlow SavedModel → TFLite, applying representative-dataset-calibrated full-integer quantization at the final stage. This pipeline presented several engineering challenges:

- The B-spline grid computation produces NaN values for certain input ranges due to a `nan_to_num` guard in `spline.py` that must be bypassed for ONNX tracing.
- A custom patch reconstructs the `torch.Tensor.expand` operation, which is not natively supported by the ONNX-to-TF converter.
- Despite requesting full INT8 quantization, KAN spline parameters are stored as INT64 tensors inside the TFLite flatbuffer (Section V-D).
- Models with $\alpha = 1.0$ pass conversion but fail at runtime with an XNNPACK delegate error (Section V-E).

To address the INT64 storage anomaly, we additionally implement an *operator decomposition* pipeline that applies targeted ONNX graph surgery prior to TFLite conversion: `HardSigmoid` nodes are decomposed into `Add` → `Mul` → `ReLU` → `Min` sequences, and `Gather` nodes (from B-spline grid indexing) are replaced with LUT-based matrix operations. With enforced full-integer quantization, this produces TFLite models containing only INT8 and INT32 tensors (Section V-F).

D. EVALUATION METRICS

For each model we report:

- **Accuracy** (%): top-1 classification accuracy on a balanced evaluation subset (6 000 images, 3 000 per class).
- **Model size** (MB): the size of the serialised model file (PyTorch `.pt`, or TFLite `.tflite`).
- **Inference latency** (ms): mean wall-clock time for a single-image forward pass, measured on an Intel Core i7 CPU for FP32 and hybrid models, and via the TFLite interpreter for TFLite models.

FIGURE 1. High-level architecture of the proposed hybrid CNN-KAN model. The MobileNetV3-Small backbone and linear projector admit INT8 quantization, while KAN layers require FP32 precision due to the B-spline computation.

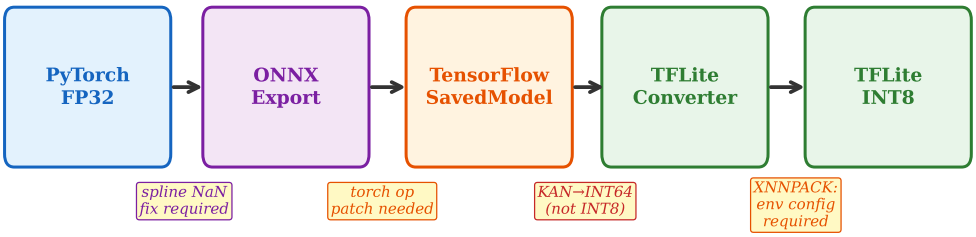

FIGURE 2. TFLite conversion pipeline with engineering challenges annotated at each stage. Green boxes indicate standard operations; orange boxes highlight KAN-specific issues encountered.

- **Total parameters**: the count of all learnable and non-learnable parameters, providing a hardware-independent measure of model capacity.
- **Peak activation memory** (KB): the maximum memory required to hold intermediate tensors during a single forward pass, as defined in Section II-B. For FP32 models, this is computed by the PyTorch profiler; for TFLite models, it is extracted from the flatbuffer metadata. This metric determines whether a model can execute within the SRAM budget of a given MCU.
- **Precision, Recall, F1**: macro-averaged over both classes (person, non-person), providing a more nuanced view than accuracy alone, especially for class-imbalanced scenarios.

Because we perform CPU-based benchmarking rather than on-device measurement, reported latencies should be interpreted as relative comparisons rather than absolute deployment figures. We discuss this limitation further in Section VI-F.

V. RESULTS

This section presents the complete experimental results for all 46 model configurations: 16 FP32 baselines, 10 hybrid INT8 variants, and 20 TFLite INT8 variants.

A. FP32 BASELINE PERFORMANCE

Table 1 presents accuracy, model size, inference latency, parameter count, and peak activation memory for all 16 FP32 configurations. Accuracy ranges from 86.93% ([32 → 16], $\alpha = 0.5$) to **88.92%** ([16 → 4], $\alpha = 1.0$).

Several important observations emerge. First, the width multiplier α is the dominant determinant of accuracy. Increasing α from 0.5 to 1.0 consistently yields 1.5–2.0 percentage points of improvement, regardless of KAN depth. For example, in the [16 → 4] family the gain is 1.55 pp (87.37% → 88.92%), and in [32 → 16] it is 1.54 pp (86.93% → 88.47%).

Second, deepening the KAN classifier provides diminishing returns. The 4-layer [64 → 32 → 24 → 16] topology achieves at most 88.77% ($\alpha = 1.0$), marginally below the 2-layer [16 → 4] at 88.92%—while requiring 3.6× the latency (182.86 ms vs. 51.00 ms) and 8.5% more parameters (1 017 704 vs. 937 688).

Third, the ultra-compact [8 → x] topologies at $\alpha = 1.0$ perform remarkably well: [8 → 4] reaches 88.80%, only 0.12 pp below the best overall result, with latency comparable to [16 → 4] (52.67 ms vs. 51.00 ms). Even [8 → 2], with only 2 hidden neurons, achieves 88.30%. The similar latencies across 2-layer topologies confirm that the backbone dominates inference time at this resolution.

Fourth, peak activation memory is dominated by the backbone. At $\alpha = 1.0$, all topologies show peak activations near 23.9 MB (FP32), far exceeding the SRAM budgets of typical MCUs. At $\alpha = 0.5$, the peak drops to ∼12.5 MB—still too large for bare-metal deployment, confirming that quantized pathways are essential.

B. HYBRID INT8 QUANTIZATION

Table 2 reports results for the 10 hybrid INT8 configurations. Quantizing the backbone and projector to INT8 while retaining the KAN head in FP32 compresses models by 2.43–3.30× with remarkably small accuracy penalties: the maximum observed loss is 0.25 pp ([64 → 32 → 24 → 16], $\alpha = 0.75$), and several configurations show marginal accuracy *improvements* (e.g., [16 → 4]$\alpha = 0.668$ gains +0.12 pp and [32 → 16]$\alpha = 1.0$ gains +0.10 pp), likely due to a mild regularisation effect of quantization noise.

The compression ratio increases with α because higher width multipliers produce larger backbone weight tensors that benefit proportionally more from INT8 compression. At $\alpha = 1.0$, the [16 → 4] configuration achieves 3.30×

TABLE 1. FP32 baseline results on the VWW validation set. Bold indicates best accuracy.

KAN Topology	α	Acc. (%)	Size (MB)	Lat. (ms)	Params	Peak (KB)
$[8 \to 2]$	1.0	88.30	3.602	50.51	932 096	23 901.5
$[8 \to 4]$	1.0	88.80	3.604	52.67	932 448	23 901.5
$[8 \to 8]$	1.0	88.63	3.606	55.91	933 152	23 901.5
$[16 \to 4]$	1.0	**88.92**	3.624	51.00	937 688	23 901.7
	0.75	88.32	2.175	48.33	560 752	19 580.4
	0.668	87.58	1.707	39.34	439 128	17 629.2
	0.5	87.37	1.025	38.97	262 368	12 490.2
$[24 \to 16]$	1.0	88.58	3.663	64.51	948 112	23 901.9
$[32 \to 16]$	1.0	88.47	3.689	79.08	954 888	23 902.1
	0.75	87.58	2.232	80.34	575 648	19 580.9
	0.668	87.53	1.761	69.28	453 256	17 629.6
	0.5	86.93	1.073	63.47	274 960	12 490.6
$[32 \to 24 \to 16]$	1.0	88.67	3.730	107.38	965 512	23 902.2
	0.75	87.85	2.272	109.00	586 272	19 581.0
$[64 \to 32 \to 24 \to 16]$	1.0	88.77	3.929	182.86	1 017 704	23 903.1
	0.75	87.60	2.454	179.49	633 856	19 581.8

TABLE 2. Hybrid INT8 quantization results. Δacc denotes accuracy change vs. FP32. CR = compression ratio.

KAN Topology	α	Acc. (%)	Δacc (pp)	Size (MB)	Lat. (ms)	CR
$[16 \to 4]$	1.0	**88.83**	−0.09	1.098	51.84	3.30×
	0.75	88.32	0.00	0.719	46.04	3.03×
	0.668	87.70	+0.12	0.599	41.27	2.85×
	0.5	87.38	+0.01	0.417	43.13	2.46×
$[24 \to 16]$	1.0	88.55	−0.03	1.138	70.69	3.22×
$[32 \to 16]$	1.0	88.57	+0.10	1.164	79.61	3.17×
	0.75	87.55	−0.03	0.777	79.59	2.87×
$[32 \to 24 \to 16]$	1.0	88.72	+0.05	1.208	106.81	3.09×
$[64 \to 32 \to 24 \to 16]$	1.0	88.60	−0.17	1.412	187.95	2.78×
	0.75	87.35	−0.25	1.008	180.05	2.43×

compression (3.624 MB → 1.098 MB), while at $\alpha = 0.5$ the same topology compresses only 2.46×.

C. TFLite INT8 RESULTS

Table 3 presents results for the 20 TFLite INT8 configurations—the largest set, including additional width multipliers ($\alpha = 0.25, 0.334$) not trained as separate FP32 baselines. All models achieve latencies between 1.27 and 3.72 ms, representing a 14–49× speedup over FP32.

Accuracy losses relative to FP32 (where available) range from 0.11 pp ($[16 \to 4]$, $\alpha = 0.668$) to 2.66 pp ($[32 \to 16]$, $\alpha = 0.5$). Shallower topologies quantize better than deeper ones, consistent with the expectation that cascaded spline operations accumulate quantization error. Interestingly, the quantization behaviour is not monotonic with α: the $[16 \to 4]$ topology shows a 0.60 pp drop at $\alpha = 1.0$ but only 0.11 pp at $\alpha = 0.668$, suggesting complex interactions between backbone feature quality and KAN quantization sensitivity.

The smallest working model, $[24 \to 16]\alpha = 0.25$ at 0.309 MB with only 98 104 parameters, achieves 83.77% accuracy with 1.27 ms latency. This configuration fits within the Flash budget of most Cortex-M4/M7 MCUs and represents the most deployment-friendly option for bare-metal nodes. The TFLite tensor arena (peak activation memory) ranges from 196 to 392 KB across all configurations—dramatically lower than FP32 models (12.5–23.9 MB). The smallest TFLite peak activation (196 KB for $[24 \to 16]\alpha = 0.25$) fits within the SRAM budget of many MCUs, including STM32H7 (1 MB SRAM) and ESP32-S3 (512 KB SRAM + 8 MB PSRAM), making bare-metal deployment feasible without patch-based inference strategies.

D. INT64 STORAGE OBSERVATION

Inspection of TFLite flatbuffers reveals that KAN spline coefficients and grid tensors are stored as INT64 rather than INT8. The TFLite converter treats B-spline operations as custom ops that bypass the standard INT8 quantization pathway. As a consequence, TFLite models are only 1.94–2.90× smaller than FP32 (mean: 2.51×), rather than the ∼4× expected from true INT8 quantization. We address this limitation through operator decomposition in Section V-F.

E. XNNPACK DELEGATE BEHAVIOUR

During initial evaluation, all five TFLite models with $\alpha = 1.0$ produced correct output through the standard TFLite interpreter but crashed with a segmentation fault when the XNNPACK delegate was enabled. This behaviour was initially attributed to an architectural incompatibility between

TABLE 3. TFLite INT8 results. Models with $\alpha = 1.0$ marked † initially failed under XNNPACK but work correctly after environment configuration (Section V-E). Δacc is vs. FP32; "n/a" indicates no matching FP32 baseline.

KAN	α	Acc. (%)	Δacc	Size (MB)	Lat. (ms)	Params	Peak (KB)
$[16 \to 4]$	$1.0^{\dagger}$	88.32	−0.60	1.250	3.37	936 701	392
	0.75	86.67	−1.65	0.837	2.89	560 967	392
	0.668	87.47	−0.11	0.701	2.65	439 015	392
	0.5	86.48	−0.89	0.496	1.87	263 191	196
$[24 \to 16]$	$1.0^{\dagger}$	88.03	−0.55	1.292	3.51	954 945	392
	0.75	87.60	n/a	0.877	2.85	577 373	392
	0.5	86.20	n/a	0.535	1.98	278 391	196
	0.334	84.22	n/a	0.367	1.49	141 485	196
	0.25	83.77	n/a	0.309	1.27	98 104	196
$[32 \to 16]$	$1.0^{\dagger}$	87.95	−0.52	1.311	3.48	959 401	392
	0.75	86.83	−0.75	0.896	3.04	583 978	392
	0.668	87.10	−0.43	0.759	2.63	462 860	392
	0.5	84.27	−2.66	0.552	1.93	282 236	196
$[32 \to 24 \to 16]$	$1.0^{\dagger}$	87.77	−0.90	1.380	3.62	977 964	392
	0.75	87.65	−0.20	0.965	3.08	597 697	392
	0.5	86.32	n/a	0.622	2.01	300 572	196
	0.334	84.08	n/a	0.453	1.56	175 771	196
	0.25	84.28	n/a	0.395	1.38	119 459	196
$[64 \to 32 \to 24 \to 16]$	$1.0^{\dagger}$	87.65	−1.12	1.553	3.72	1 042 751	392
	0.75	85.95	−1.65	1.133	3.23	661 888	392

the KAN operators and the XNNPACK dispatch logic. However, further investigation revealed that the failures stemmed from a Python library version mismatch on the evaluation platform (Windows): the installed `tflite-runtime` version was incompatible with the XNNPACK delegate bundled in the TFLite distribution. After updating the Python environment to ensure matched library versions, all five $\alpha = 1.0$ models executed correctly under XNNPACK delegation, achieving accuracies of 88.32%, 88.03%, 87.95%, 87.77%, and 87.65% for the $[16 \to 4]$, $[24 \to 16]$, $[32 \to 16]$, $[32 \to 24 \to 16]$, and $[64 \to 32 \to 24 \to 16]$ topologies respectively.

This finding is worth documenting for two reasons. First, it demonstrates that KAN-based TFLite models are not inherently incompatible with XNNPACK acceleration, which is encouraging for deployment on ARM-based devices. Second, it highlights that environment configuration issues in the TFLite/XNNPACK ecosystem can masquerade as architectural failures, underscoring the importance of careful toolchain version management in TinyML workflows.

F. OPERATOR DECOMPOSITION FOR FULL INT8 DEPLOYMENT

The INT64 storage anomaly reported in Section V-D stems from two specific operations in the ONNX graph that lack native INT8 kernels in TFLite: the `HardSigmoid` activation (used internally by MobileNetV3's squeeze-and-excitation blocks) and the `Gather` operation (generated by the B-spline grid indexing in KAN layers). We resolve both through targeted ONNX graph surgery applied before the TFLite conversion stage.

For `HardSigmoid`, we decompose HardSigmoid(x) = min(max(0, $x/6 + 0.5$), 1) into four primitive operations: `Add`$(x, 3.0) \to$ `Mul`$(\cdot, 1/6) \to$ `ReLU`$(\cdot) \to$ `Min`$(\cdot, 1.0)$.

TABLE 4. Effect of operator decomposition on TFLite model composition ($[16 \to 4]$, $\alpha = 0.334$, 96×96).

Metric	Before	After
Unique operators	28	13
Total tensors	791	326
INT8 tensors	609	264
INT64 tensors	41	0
FP32 tensors	16	0
INT32 tensors	77	62
bool tensors	48	0

Each of these operations has a well-supported INT8 kernel in TFLite and XNNPACK.

For `Gather`—which arises from the B-spline basis selection and cannot be directly quantized—we replace it with a look-up table (LUT) implemented as matrix multiplications and element-wise operations that are natively quantizable. This approximation discretises the learned spline functions into a fixed set of pre-computed values, trading continuous B-spline evaluation for standard operator compatibility.

Applying both transformations with enforced full-integer quantization (`-use_lut -enforce_quantization`) produces models with a dramatically simplified operator set. Table 4 compares the tensor-level composition before and after decomposition for the $[16 \to 4]\alpha = 0.334$ topology at 96×96 input resolution.

The decomposed model contains *only* INT8 and INT32 tensors (the latter for bias terms and shape metadata, which is standard for TFLite INT8 models). All 41 INT64 tensors and all 16 FP32 tensors are eliminated entirely. The non-standard operators (`BATCH_MATMUL`, `CAST`, `GATHER`, `SELECT_V2`, `LOGISTIC`, `STRIDED_SLICE`) are replaced by compositions of standard operators (`ADD`,

TABLE 5. Accuracy impact of operator decomposition and quantization for the [16 → 4] topology.

Config	Pipeline	Acc. (%)	Δ (pp)
$\alpha = 1.0$, 224×224	TFLite FP32 (baseline)	88.62	—
	TFLite FP32 + LUT	85.60	−3.02
	TFLite INT8 + LUT	86.07	−2.55
$\alpha = 0.334$, 96×96	TFLite INT8 (baseline)	84.0	—
	TFLite INT8 + LUT	83.0	−1.0

TABLE 6. Preliminary on-device inference measurements. All models use the [16 → 4] topology with operator decomposition and LUT-based spline approximation. Values in the Chirale rows for $\alpha = 0.5$ and $\alpha = 1.0$ are from earlier measurements and await re-confirmation.

Device	Input	α	Size	RAM	Lat. (s)
XIAO ESP32S3 Sense — Chirale TFLite Micro runtime [35]					
	96×96	0.334	0.26 MB	SRAM	1.61
	96×96	0.334	0.26 MB	PSRAM	1.67
XIAO ESP32S3 Sense — TFLM_ESP32 reference runtime					
	96×96	0.334	0.26 MB	PSRAM	5.8
Raspberry Pi 3 Model B — standard TFLite runtime					
	224×224	1.0	1.30 MB	RAM	0.16
	224×224	1.0	1.20 MB	RAM	0.11

Note: The last two Raspberry Pi rows compare the *same* model ($\alpha = 1.0$, 224×224) without (1.30 MB, 160 ms) and with (1.20 MB, 110 ms) operator decomposition. On ESP32-S3, the SRAM vs. PSRAM latency difference is only 0.06 s (3.6%), indicating that computation, not memory access, is the bottleneck at this model scale.

`MUL`, `MINIMUM`, `ABS`, `SUM`). This enables full XNNPACK delegation for all operations.

Table 5 quantifies the accuracy impact of operator decomposition for the [16 → 4] topology at two representative scales. The LUT-based spline approximation incurs a penalty of 1–3 pp relative to the unmodified TFLite model, with the larger model ($\alpha = 1.0$, 224 × 224) exhibiting a greater loss due to the higher number of spline parameters being discretised. Notably, full INT8 quantization applied *after* LUT decomposition recovers a fraction of the lost accuracy (+0.47 pp for $\alpha = 1.0$), consistent with the mild regularisation effect observed for hybrid INT8 models in Section V-B.

G. PRELIMINARY ON-DEVICE VALIDATION

To validate deployment feasibility beyond desktop simulation, we measure inference latency on two representative IoT platforms: an XIAO ESP32S3 Sense (Xtensa LX7 dual-core at 240 MHz, 512 KB SRAM + 8 MB PSRAM) and a Raspberry Pi 3 Model B (ARM Cortex-A53 quad-core at 1.2 GHz). All on-device models use the operator-decomposed TFLite INT8 pipeline from Section V-F. Table 6 reports the results.

Three observations emerge from these preliminary measurements. First, the smallest configuration ($\alpha = 0.334$, 96 × 96 input, 0.26 MB) fits entirely within the ESP32-S3's internal SRAM, achieving an inference time of ∼1.6 s. Initial on-device accuracy over the full 6 000-image validation set reached 81.2%, approximately 1.8 pp below the desktop TFLite result (83.0%). Investigation revealed that this gap was caused by a per-channel vs. per-tensor quantization mismatch: the TFLite converter applies per-channel quantization to the fully connected projector layer, but the Chirale TFLite Micro runtime silently applies a single per-tensor scale factor, producing incorrect dequantized activations without raising any error. After patching the runtime's `fully_connected` kernel with the per-channel-aware implementation from Espressif's ESP-NN library, on-device accuracy reaches **83.13%**—within 0.07 pp of the desktop result (83.07%), confirming that the deployment pipeline introduces no meaningful accuracy loss. Alternatively, the TFLite converter flag `_experimental_disable_per_channel_quantization_for_dense_layers` can be used to force per-tensor quantization at export, avoiding the runtime mismatch entirely with negligible accuracy impact (83.12% per-tensor vs. 83.07% per-channel on desktop).

Second, the choice of TFLite Micro runtime implementation has a dramatic impact on inference speed. On identical hardware and model, the Chirale TFLite Micro runtime [35] achieves 1.67 s compared with 5.8 s for the reference TFLM_ESP32 runtime—a 3.5× difference attributable to operator-level optimisations in the runtime library. This underscores that runtime selection is a first-order deployment consideration, comparable in impact to model architecture choices.

Third, operator decomposition provides a measurable speedup even on Linux-class devices: the Raspberry Pi 3 achieves 110 ms per frame with the decomposed model compared with 160 ms without, a 1.5× improvement from enabling full XNNPACK delegation over all operators.

H. TRAINING CONVERGENCE

Fig. 3 shows representative training curves. All models converge within the first 30 of 50 training epochs, with most accuracy gain in the first 10–15 epochs. Ultra-compact topologies converge as quickly as deeper variants, suggesting that spline flexibility compensates for reduced capacity.

I. CONFUSION MATRIX ANALYSIS

Fig. 4 presents confusion matrices for the best FP32 model ([16 → 4], $\alpha = 1.0$) and its TFLite INT8 counterpart, computed on the balanced evaluation subset (6 000 images in total: 3 000 person, 3 000 non-person).

For the FP32 model: TN = 2 758, FP = 242, FN = 423, TP = 2 577, yielding a specificity (true negative rate) of 91.93%, a sensitivity (true positive rate) of 85.90%, macro-averaged precision of 89.06%, recall of 88.92%, and F1 of 88.91%.

For the TFLite INT8 model: TN = 2 752, FP = 248, FN = 453, TP = 2 547. Compared with FP32, quantization primarily increases false negatives (+30) rather than false positives (+6), reducing sensitivity from 85.90% to 84.90% while leaving specificity nearly unchanged (91.73% vs. 91.93%). This means quantization makes the model slightly

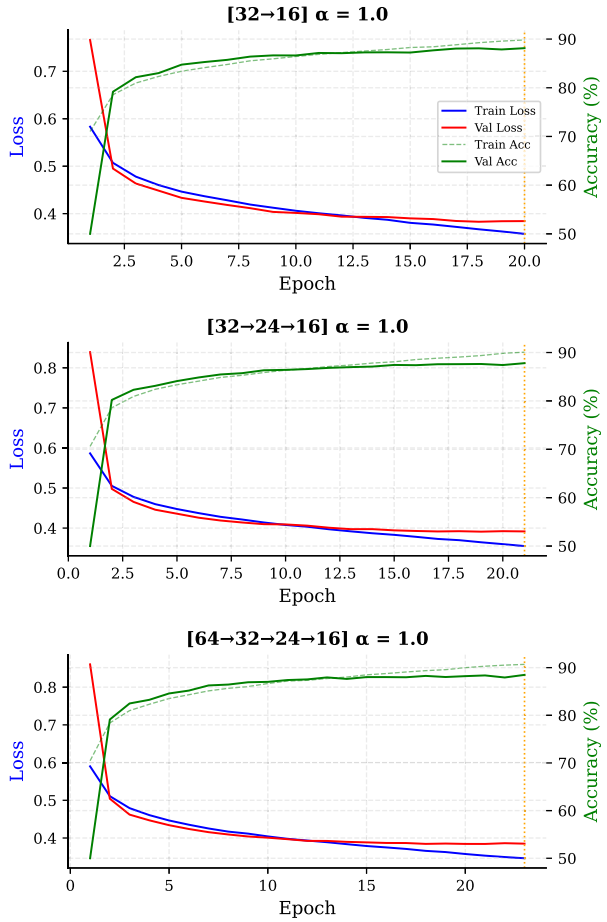

FIGURE 3. Training curves for representative configurations. All models converge within the first 30 of 50 training epochs.

more conservative—more likely to miss a person than to hallucinate one.

This asymmetry has practical implications for deployment. In always-on surveillance triggers, a missed detection can be compensated by the next frame (typically captured 100–500 ms later), whereas a false positive wastes energy by activating downstream processing. The observed false-negative rate of 7.07–8.50% across our models is therefore acceptable for most duty-cycled IoT applications.

Across all 46 models, we observe a consistent pattern: the false-negative rate (FN/(FN+TP)) exceeds the false-positive rate (FP/(FP+TN)) by 3–5 percentage points, indicating that the models are systematically biased toward the non-person class. This bias likely reflects the class distribution of VWW (∼60% non-person) and could be adjusted through threshold calibration if a specific operating point is required.

J. SCALING EFFECTS

Fig. 5 decomposes accuracy vs. width multiplier for each quantization pathway in three vertical panels. The consistent dominance of α over KAN topology is visually apparent: within each panel, curves for different KAN depths cluster tightly at each α value, with spread rarely exceeding 1 pp.

Fig. 6 illustrates latency scaling in two panels. Panel (a) shows FP32 latency as a function of α for each topology, revealing that deeper KAN heads incur dramatically higher latency: the 4-layer [64 → 32 → 24 → 16] requires 182.86 ms per frame, compared with 51.00 ms for [16 → 4]—a 3.6× penalty for at most 0.47 pp accuracy gain. Among 2-layer topologies, latency differences are modest (39–56 ms at $\alpha = 1.0$), confirming that the backbone dominates inference time. Panel (b) provides a grouped bar chart comparing FP32, hybrid INT8, and TFLite INT8 latencies at $\alpha = 1.0$ across all eight topologies. TFLite models achieve sub-4 ms latencies regardless of KAN depth—a 14–49× speedup over FP32—because the TFLite interpreter executes the dominant backbone computation in optimised INT8 kernels. Notably, hybrid INT8 models show negligible latency improvement over FP32 for deep KAN topologies, because the KAN head, which remains in FP32, dominates wall-clock time once the backbone is accelerated.

K. COMPRESSION ANALYSIS

Fig. 7 quantifies compression. Hybrid INT8 delivers mean 2.92× size reduction (range: 2.43–3.30×) with <0.25 pp accuracy loss. TFLite INT8 achieves mean 2.51× (range: 1.94–2.90×)—well below the expected 4×—due to INT64 storage.

L. PARETO ANALYSIS AND DEPLOYMENT MAPPING

Fig. 8 overlays all 46 variants on a size-vs-accuracy plot with IoT memory-budget zones. The Pareto frontier is dominated by hybrid INT8 models in the 0.4–1.5 MB range and by TFLite models below 0.5 MB.

VI. DISCUSSION

A. POSITIONING RELATIVE TO STATE OF THE ART

Table 8 places our results alongside published VWW benchmarks. Our best FP32 model (88.92%, 3.62 MB) is competitive with MobileNetV2 $\alpha = 1.0$ (89.9%, 2.30 MB) [1] and falls below the NAS-optimised MCUNetV2 (>90%, <1 MB) [4]. However, MCUNet results rely on 4-bit quantization and a custom inference engine (TinyEngine), which are not available in our pipeline.

At matched model sizes in the 0.4–1.0 MB range, our hybrid INT8 models achieve 87–89% accuracy, aligning with MCUNet's 87.4% at 0.74 MB. This suggests that replacing the MLP head with a KAN classifier neither dramatically helps nor hurts accuracy at this scale.

B. KAN VERSUS MLP: WHEN DOES THE SPLINE HELP?

The learnable B-spline activations provide a richer function space than fixed ReLU or hard-swish activations. To quantify this advantage on VWW, we conducted two complementary ablation experiments under identical training conditions (same backbone, AdamW optimiser, 50 epochs, cosine annealing, label smoothing $\varepsilon = 0.1$).

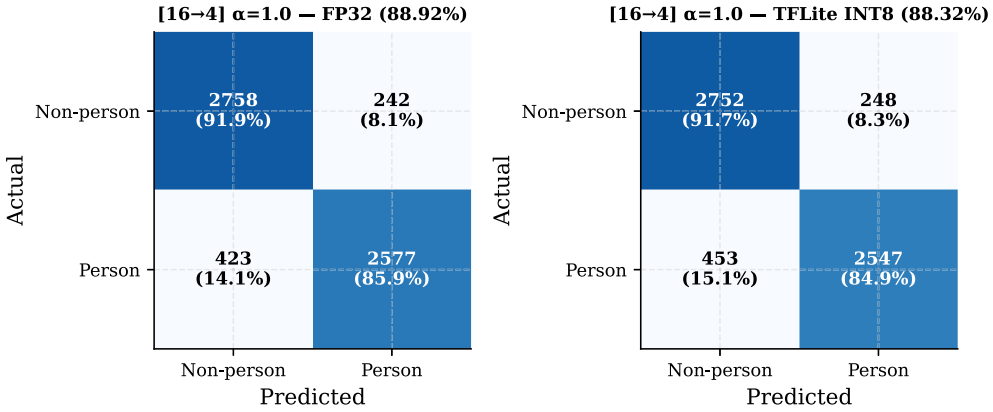

FIGURE 4. Confusion matrices for (a) the best FP32 model ([16→4], $\alpha=1.0$, 88.92%) and (b) its TFLite INT8 counterpart (88.32%). Values show raw counts and row-normalised percentages. Quantization primarily increases false negatives.

TABLE 7. Recommended configurations by IoT device class.

Budget	Target Device	Config	Size	Acc.	Quant.
<0.4 MB	XIAO, nRF52840	$[24\rightarrow16]$, $\alpha=0.25$	0.31 MB	83.77%	TFLite
0.4–0.7 MB	ESP32-S3 (low)	$[16\rightarrow4]$, $\alpha=0.5$	0.42 MB	87.38%	Hybrid
0.7–1.2 MB	ESP32-S3	$[16\rightarrow4]$, $\alpha=0.75$	0.72 MB	88.32%	Hybrid
1–2 MB	RPi Zero 2W	$[16\rightarrow4]$, $\alpha=1.0$	1.10 MB	88.83%	Hybrid
>2 MB	RPi 4, Jetson	$[16\rightarrow4]$, $\alpha=1.0$	3.62 MB	88.92%	FP32

TABLE 8. Comparison with published VWW benchmarks. Entries marked * from original publications.

Model	Size (MB)	Acc. (%)	Quant.	Source
MobileNetV2 $\alpha=0.25$*	0.24	85.9	INT8	[1]
MobileNetV2 $\alpha=1.0$*	2.30	89.9	INT8	[1]
MCUNet*	0.74	87.4	INT8	[3]
MCUNetV2*	0.73	90.0	INT4+8	[4]
Ours: Hybrid $[16\rightarrow4]$ $\alpha=0.5$	0.42	87.38	Hybrid INT8	This work
Ours: Hybrid $[16\rightarrow4]$ $\alpha=1.0$	1.10	88.83	Hybrid INT8	This work
Ours: TFLite $[24\rightarrow16]$ $\alpha=0.25$	0.31	83.77	INT8	This work
Ours: FP32 $[16\rightarrow4]$ $\alpha=1.0$	3.62	88.92	FP32	This work

TABLE 9. MLP ablation under identical training conditions. Upper block: parameter-matched (~800 vs. 792 params). Lower block: topology-matched (4 hidden neurons, ~78 MLP params).

Matching	α	KAN (%)	MLP (%)	Δ (pp)
Parameter-matched (FP32)				
	1.0	88.92	88.67	−0.25
	0.5	87.37	87.00	−0.37
Topology-matched, 4 hidden neurons (FP32)				
	1.0	88.92	87.00	−1.92
Topology-matched, 4 hidden neurons (INT8†)				
	1.0	86.07	87.02	+0.95
	0.334	83.0	83.3	+0.30

†KAN INT8 uses LUT decomposition (Section V-F).

Table 9 presents the results. In the *parameter-matched* comparison (MLP with ~800 parameters, matching the 792 parameters of the KAN [16 → 4] head), the KAN advantage is marginal: +0.25 pp at $\alpha = 1.0$ and +0.37 pp at $\alpha = 0.5$—differences within the expected run-to-run variability of a single training run. In the *topology-matched* comparison (MLP with 4 hidden neurons, matching the KAN [16 → 4] architecture but with only ~78 parameters), the KAN advantage is more pronounced in FP32 (+1.92 pp at $\alpha=1.0$), because the 11 learnable parameters per KAN edge provide substantially more representational capacity than the single weight per MLP connection at matched width. After INT8 quantization with LUT decomposition, however, both heads converge: 87.02% (MLP) vs. 86.07% (KAN) at $\alpha=1.0$, and 83.3% vs. 83.0% at $\alpha=0.334$.

These results lead to two conclusions. First, at matched parameter budgets, the B-spline activations do not yield a measurable advantage on VWW—the task is sufficiently

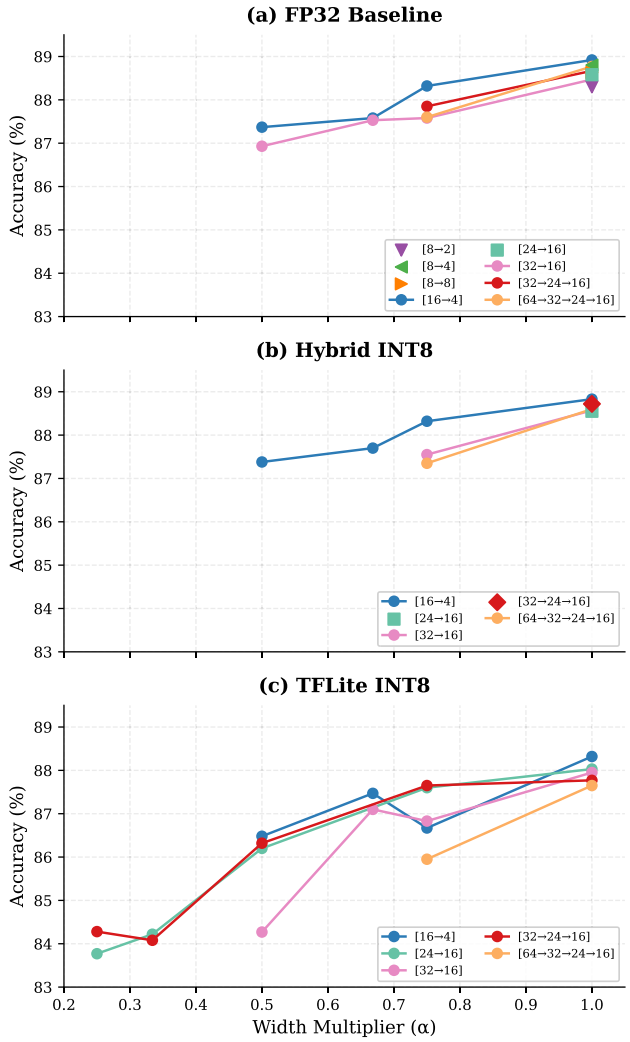

FIGURE 5. Classification accuracy vs. width multiplier for each quantization pathway. Each panel spans a separate row for clarity: (a) FP32 baselines, (b) Hybrid INT8, (c) TFLite INT8. The consistent dominance of α over KAN topology is visually apparent—within each panel, curves for different KAN architectures cluster tightly at each α value.

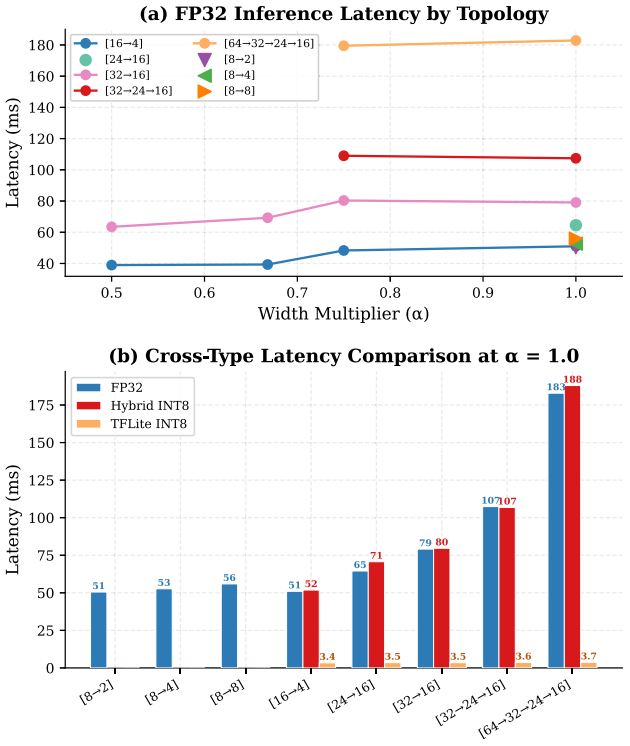

FIGURE 6. Inference latency analysis. (a) FP32 latency by KAN topology as a function of width multiplier α. Deeper topologies incur up to 3.6× latency penalties. (b) Grouped bar chart comparing FP32, hybrid INT8, and TFLite INT8 latencies at $\alpha = 1.0$ across all eight topologies. TFLite achieves sub-4 ms for all configurations. Values above bars indicate latency in milliseconds.

simple that a conventional MLP captures the decision boundary equally well. Second, after INT8 quantization the MLP *slightly exceeds* KAN accuracy because MLP quantizes losslessly, whereas KAN requires the lossy LUT approximation. This represents a genuine advantage of conventional classifiers for MCU deployment.

The ultra-compact results ([8 → 2]: 88.30%, [8 → 4]: 88.80%) reinforce this interpretation: even a 2-neuron bottleneck with KAN splines achieves within 0.62 pp of the best result, suggesting that the classification boundary is nearly linear in the backbone's feature space.

The INT64 storage anomaly documented in Section V-D further penalises KAN models, though the operator decomposition strategy in Section V-F eliminates this issue at the cost of 1–3 pp additional accuracy loss from LUT approximation. A key open question is whether KAN classifiers would show a clearer advantage on more challenging tasks—multi-class classification, fine-grained recognition, or datasets with complex decision boundaries where the backbone features alone are insufficient. Our results establish the VWW baseline and demonstrate that the hybrid CNN-KAN architecture is at least competitive with standard approaches, while the engineering challenges documented here provide a roadmap for future improvements.

C. THE HYBRID INT8 STRATEGY AS A PRACTICAL COMPROMISE

Our results identify hybrid INT8 quantization—compressing the CNN backbone to INT8 while keeping the KAN head in FP32—as the optimal strategy for current toolchains. This approach achieves 60–70% of the compression of full TFLite INT8 but avoids the KAN-specific quantization pitfalls entirely, with negligible accuracy loss (<0.25 pp in all cases and occasional marginal improvements due to quantization-induced regularisation).

The hybrid strategy exploits a structural property of our architecture: the MobileNetV3-Small backbone accounts for >95% of total parameters in most configurations, so quantizing it alone yields substantial compression while the compact KAN head contributes negligibly to overall model size. For deployment platforms that support mixed-precision inference

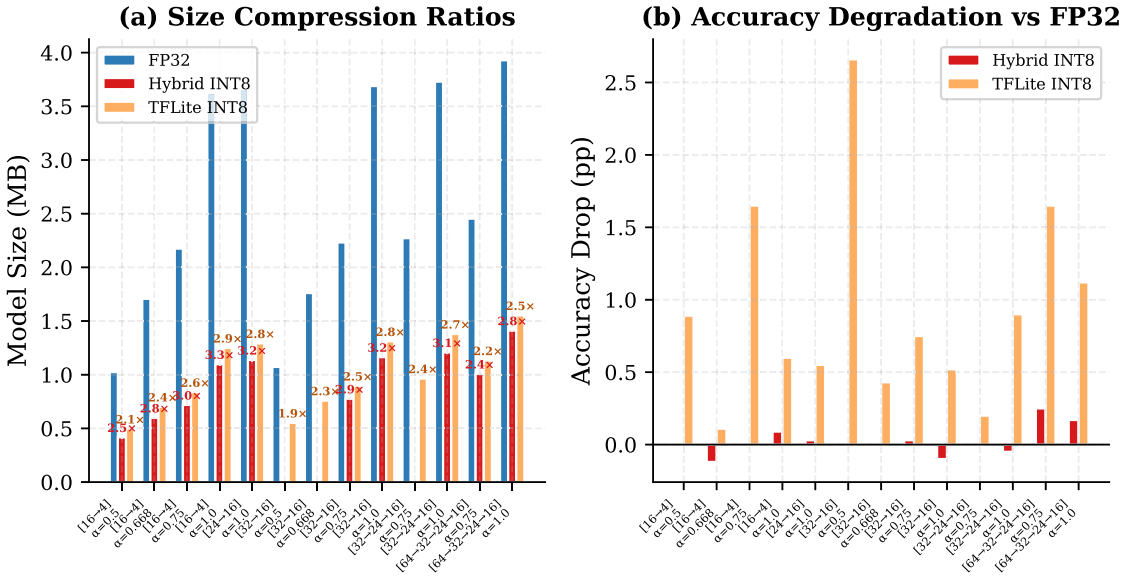

FIGURE 7. Compression analysis. (a) Size comparison with compression ratios. (b) Accuracy degradation.

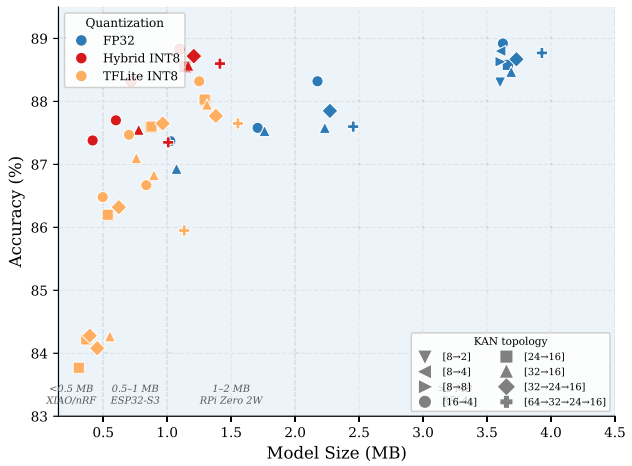

FIGURE 8. Pareto frontier across all quantization strategies with target device budgets.

(e.g., Linux-based boards with PyTorch or ONNX Runtime, or custom firmware with separate execution paths for INT8 and FP32 operators), this pathway is production-ready.

D. PRACTICAL RECOMMENDATIONS

1) **Prefer shallow KAN heads.** The 2-layer [16 → 4] offers the best accuracy–latency trade-off. The ultra-compact [8 → 4] is recommended when parameter count is critical (88.80% with only 932 448 parameters).
2) **Use hybrid INT8 for production.** It provides >3× compression with negligible accuracy loss.
3) **Apply operator decomposition for MCU targets.** Replace `HardSigmoid` and `Gather` with standard operator compositions before TFLite conversion to eliminate INT64 tensors and enable full XNNPACK delegation (Section V-F).
4) **Verify XNNPACK environment.** Ensure matched versions of `tflite-runtime` and XNNPACK before deployment. All α values work correctly when the toolchain is properly configured.
5) **Consider peak activation memory.** FP32 models require >12 MB activation memory even at $\alpha = 0.5$. TFLite INT8 models reduce the tensor arena to 196–392 KB, well within the SRAM budgets of MCUs such as STM32H7 and ESP32-S3.
6) **Prioritise width multiplier over KAN depth.** Increasing α from 0.5 to 0.75 yields 1–1.5 pp; increasing KAN depth from 2 to 4 layers yields <0.5 pp while multiplying latency by 3–8×.
7) **Select the TFLite Micro runtime carefully.** On-device measurements show up to 3.5× latency variation between runtime implementations on identical hardware and model (Section V-G).

E. NEGATIVE RESULTS, RESOLVED ISSUES, AND THEIR IMPLICATIONS

Two engineering obstacles initially appeared as fundamental limitations of the CNN-KAN deployment pipeline; both were ultimately resolved.

The INT64 storage anomaly (Section V-D) was caused by the TFLite converter's inability to quantize `HardSigmoid` and `Gather` operations to INT8. The operator decomposition strategy described in Section V-F fully eliminates all INT64 and FP32 tensors by replacing these operations with equivalent compositions of standard TFLite operators. The decomposed model on Raspberry Pi 3 is both smaller

(1.20 MB vs. 1.30 MB) and faster (110 ms vs. 160 ms) than the unmodified TFLite INT8 model, confirming that the decomposition not only resolves the storage anomaly but also enables more efficient execution through full XNNPACK delegation.

The XNNPACK delegate failures initially reported for $\alpha = 1.0$ models were ultimately traced to a Python runtime version mismatch rather than an architectural incompatibility (Section V-E). This resolution is significant because it means KAN-based models are compatible with XNNPACK acceleration, which is essential for production deployment on ARM processors. However, the diagnostic difficulty of this issue—which initially appeared to be a fundamental KAN incompatibility—illustrates the fragility of the TinyML toolchain for non-standard architectures.

The remaining open challenge is the accuracy impact of the LUT-based spline approximation used in operator decomposition. As shown in Table 5, the LUT penalty ranges from 1 pp ($\alpha = 0.334$, 96×96) to 3 pp ($\alpha = 1.0$, 224×224), with larger models losing more due to the greater number of spline parameters being discretised. An encouraging finding is that INT8 quantization applied after LUT decomposition partially recovers accuracy (+0.47 pp for $\alpha = 1.0$), yielding a net penalty of 2.55 pp for the full decomposition-plus-quantization pipeline. On-device evaluation initially revealed a 1.8 pp gap for the $\alpha = 0.334$ configuration (83.0% desktop $\rightarrow$ 81.2% on-device), which was traced to a per-channel vs. per-tensor quantization mismatch in the Chirale TFLite Micro runtime's `fully_connected` kernel. After patching the kernel with a per-channel-aware implementation from ESP-NN, the gap closed to 0.07 pp (83.13% on-device vs. 83.07% desktop). This finding has implications beyond KAN deployment: any TFLite Micro runtime that lacks per-channel support for dense layers will exhibit similar silent accuracy degradation. Reducing the LUT approximation penalty through finer discretisation or quantization-aware training of spline coefficients remains a promising direction for further improvement.

F. LIMITATIONS

1) **Per-channel quantization in TFLite Micro.** The desktop-to-device accuracy gap was traced to a per-channel vs. per-tensor quantization mismatch in the Chirale runtime and resolved by patching the `fully_connected` kernel (Section V-G). Other TFLite Micro runtimes may exhibit similar issues; developers should validate on-device accuracy against desktop baselines. Power consumption has not yet been measured. Desktop-derived latency figures reported in Sections V-A–V-C should be interpreted as relative comparisons rather than deployment predictions.
2) **Single dataset.** We evaluate exclusively on VWW. Tasks with more complex decision boundaries—multi-class classification, fine-grained recognition—may benefit more from KAN expressiveness.
3) **No QAT.** Quantization-aware training of KAN layers could narrow the accuracy gap between FP32 and TFLite INT8 models. This would require differentiable quantization of spline coefficients, which is a non-trivial extension.
4) **Peak activation measurement.** The peak activation values reported are derived from profiling tools rather than direct SRAM measurement on target hardware. Actual memory consumption may vary depending on the inference engine's memory allocation strategy and operator scheduling.
5) **No knowledge distillation.** Distillation from a larger teacher could improve accuracy without increasing model size, providing an additional compression pathway complementary to quantization.
6) **Fixed spline configuration.** All experiments use cubic B-splines with $G = 5$ grid intervals. Varying these hyperparameters could yield better accuracy–size trade-offs but would expand the search space significantly.

VII. COMPREHENSIVE EXPERIMENTAL OVERVIEW

Fig. 9 provides a six-panel summary of the complete experimental campaign, consolidating the key trade-offs across all 46 model configurations.

Panel (a) plots model size against accuracy, revealing a clear Pareto structure: FP32 models occupy the upper-right quadrant (high accuracy, large size), while TFLite INT8 variants cluster in the lower-left. Hybrid INT8 models achieve a favourable intermediate position, offering near-FP32 accuracy at roughly one-third the size.

Panel (b) shows the latency–accuracy trade-off on a logarithmic time axis. The two-order-of-magnitude latency gap between FP32 (∼50–180 ms) and TFLite INT8 (∼1.3–3.7 ms) underscores the practical necessity of quantization for real-time applications, while highlighting that hybrid INT8 models do not substantially improve latency because the FP32 KAN head remains the bottleneck.

Panel (c) compares mean compression ratios. Hybrid INT8 achieves 2.92× mean compression—closer to the theoretical 4× limit—while TFLite INT8 reaches only 2.47× due to the INT64 storage anomaly documented in Section V-D. The gap between these two bars visually summarises the cost of the TFLite converter's inability to properly quantize B-spline operations.

Panel (d) relates parameter count to accuracy. Models with fewer than 300 K parameters achieve 83–87% accuracy, while surpassing 900 K parameters yields diminishing returns (87–89%). This saturation aligns with our finding that backbone width (α) is a more effective accuracy lever than KAN depth.

Panel (e) presents the distribution of quantization-induced accuracy degradation. Hybrid INT8 models cluster tightly near zero degradation, while TFLite INT8 models show a

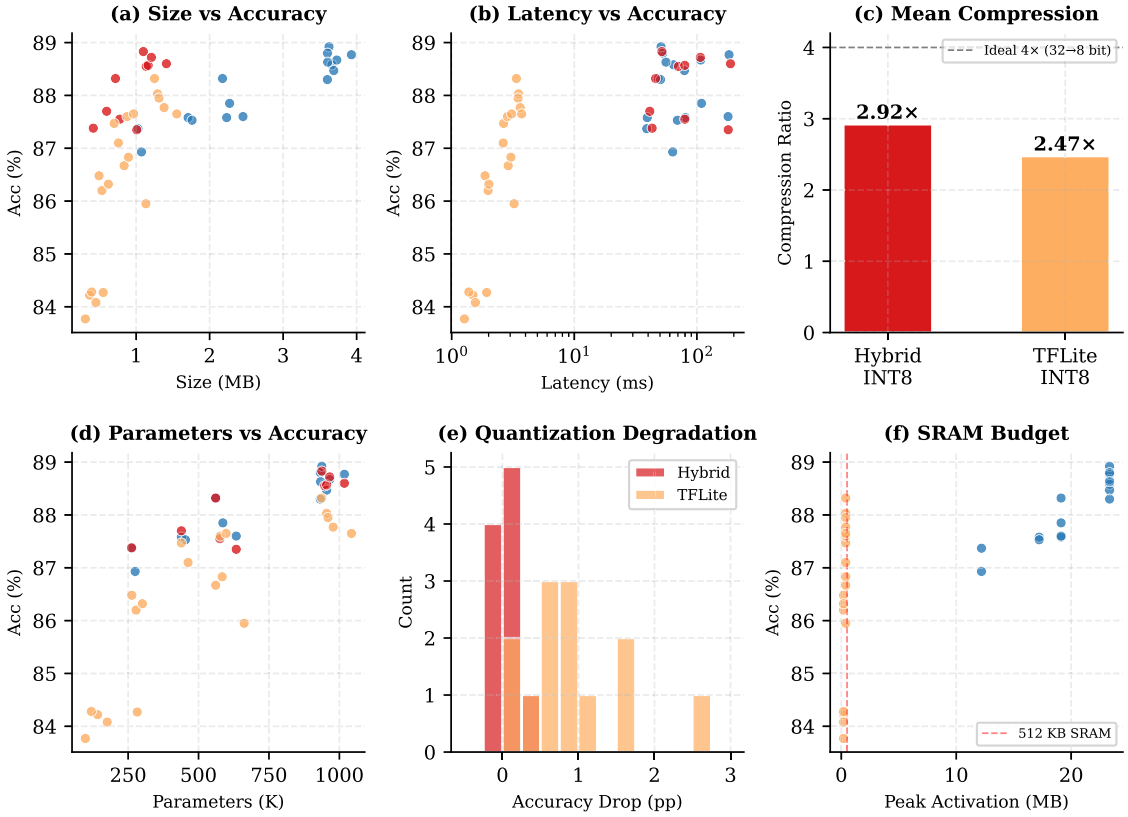

FIGURE 9. Comprehensive overview. (a) Size vs. accuracy. (b) Latency vs. accuracy. (c) Mean compression ratio. (d) Parameters vs. accuracy. (e) Degradation distribution. (f) Peak activation vs. accuracy.

wider spread extending beyond 2 pp—consistent with the error accumulation in cascaded quantized spline operations.

Panel (f) plots peak activation memory against accuracy, with a vertical reference line at 512 KB SRAM. All TFLite INT8 models fall within feasible SRAM budgets (196–392 KB), whereas FP32 models require 12.5–23.9 MB—two orders of magnitude above typical MCU SRAM capacities. This panel most directly addresses the deployment feasibility question: only quantized models are viable for bare-metal MCU execution.

VIII. CONCLUSION

We have presented a systematic evaluation of hybrid CNN-KAN architectures for person detection on IoT devices, spanning 46 model configurations across eight KAN topologies and three quantization strategies. Our main findings are as follows.

First, the backbone width multiplier α is a more effective lever for controlling the accuracy–size trade-off than KAN classifier depth. Shallow 2-layer KAN heads achieve within 0.47 pp of 4-layer variants while requiring up to 3.6× less inference time. The ultra-compact $[8 \rightarrow 4]$ topology achieves 88.80% with only 932 448 parameters.

Second, hybrid INT8 quantization delivers up to 3.30× compression with accuracy losses below 0.25 pp, making it the recommended pathway for mixed-precision platforms.

Third, TFLite INT8 yields sub-megabyte models (0.31–1.55 MB) but compression is limited to 1.94–2.90× due to INT64 storage of KAN parameters. We resolve this limitation through operator decomposition—replacing `HardSigmoid` and `Gather` with compositions of standard TFLite operators—which eliminates all INT64 and FP32 tensors from the converted model. The decomposed model achieves both smaller size (1.20 MB vs. 1.30 MB) and lower latency (110 ms vs. 160 ms on Raspberry Pi 3) through full XNNPACK delegation, at an accuracy cost of 1–3 pp depending on model configuration (86.07% vs. 88.62% for the best $\alpha = 1.0$ model).

Fourth, initial XNNPACK delegate failures for $\alpha = 1.0$ TFLite models were traced to a runtime environment configuration issue, not an architectural incompatibility. After resolution, all models execute correctly under XNNPACK delegation, confirming that KAN-based TFLite models are compatible with hardware-accelerated inference on ARM processors.

Fifth, KAN classifiers do not offer a measurable accuracy advantage over MLP heads on VWW, but the ultra-compact results ($[8 \rightarrow 4]$: 88.80% with only 932 448 parameters) demonstrate that KAN splines can maintain high accuracy even with extremely narrow bottlenecks, which may prove

valuable for applications requiring the smallest possible classifier head.

Sixth, peak activation memory for FP32 models ranges from 12.5–23.9 MB, far exceeding MCU SRAM budgets, while TFLite INT8 models reduce the tensor arena to 196–392 KB, making deployment feasible on MCUs with ≥512 KB of SRAM without requiring patch-based inference strategies.

Seventh, on-device evaluation on an XIAO ESP32S3 Sense confirms that the smallest operator-decomposed model ([16 → 4], $\alpha = 0.334$, 96 × 96 input, 0.26 MB) fits entirely within internal SRAM and achieves ∼1.6 s inference time, validating the tensor arena estimates from desktop profiling. After resolving a per-channel vs. per-tensor quantization mismatch in the Chirale TFLite Micro runtime, on-device accuracy reaches 83.13% over the full 6 000-image validation set—within 0.07 pp of the desktop result (83.07%)—demonstrating that end-to-end deployment of a hybrid CNN-KAN architecture on a microcontroller introduces no meaningful accuracy loss.

Future work will target: (i) quantization-aware training for B-spline layers, which could narrow the accuracy gap introduced by LUT-based spline approximation by allowing spline coefficients to adapt to discretisation noise during backpropagation; (ii) systematic on-device benchmarking with power consumption profiling and latency characterisation across a wider range of MCU targets (ARM Cortex-M7, RISC-V), with particular attention to per-channel quantization support across different TFLite Micro runtime implementations; (iii) multi-class and fine-grained classification tasks where the representational richness of KAN layers may confer a clearer advantage over MLP heads; and (iv) finer LUT discretisation strategies to reduce the 1–3 pp approximation penalty while maintaining full INT8 compatibility.

ACKNOWLEDGMENT

The trained model checkpoints, training scripts, and raw experimental data are available at https://github.com/KuznetsovKarazin/kan-image-iot (branch v2b)

REFERENCES

[1] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," 2019, *arXiv:1906.05721*.

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. ECCV*, 2014, pp. 740–755.

[3] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 11711–11722.

[4] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, p. 13.

[5] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[6] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Hou, and M. Tegmark, "KAN: Kolmogorov–Arnold networks," in *Proc. ICLR*, 2025, p. 47.

[7] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," *Dokl. Akad. Nauk SSSR*, vol. 114, pp. 953–956, 1957.

[8] V. I. Arnold, "On functions of three variables," *Dokl. Akad. Nauk SSSR*, vol. 114, pp. 679–681, 1957.

[9] A. D. Bodner, A. Santiago Tepsich, J. N. Spolski, and S. Pourteau, "Convolutional Kolmogorov–Arnold networks," 2024, *arXiv:2406.13155*.

[10] Z. Bozorgasl and H. Chen, "Wav-KAN: Wavelet Kolmogorov–Arnold networks," 2024, *arXiv:2405.12832*.

[11] Blealtan. (2024). *An Efficient Implementation of Kolmogorov–Arnold Network*. [Online]. Available: https://github.com/Blealtan/efficient-kan

[12] Z. Liu, P. Ma, Y. Wang, W. Matusik, and M. Tegmark, "KAN 2.0: Kolmogorov–Arnold networks meet science," 2024, *arXiv:2408.10205*.

[13] Y. Hou, T. Ji, D. Zhang, and A. Stefanidis, "Kolmogorov–Arnold networks: A critical assessment of claims, performance, and practical viability," 2024, *arXiv:2407.11075*.

[14] B. Azam and N. Akhtar, "Suitability of KANs for computer vision: A preliminary investigation," 2024, *arXiv:2406.09087*.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[17] T. J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-aware neural network adaptation for mobile devices," in *Proc. ECCV*, 2018, pp. 285–300.

[18] G. Wu, Y. Chen, and Y. Zhang, "MicroENet: An efficient network for MCUs with low model parameters and peak memory," in *Proc. 27th Int. Conf. Comput. Supported Cooperat. Work Des. (CSCWD)*, May 2024, pp. 790–795, doi: 10.1109/CSCWD61410.2024.10580468.

[19] A. M. Garavagno, D. Leonardis, and A. Frisoli, "ColabNAS: Obtaining lightweight task-specific convolutional neural networks following Occam's Razor," *Future Gener. Comput. Syst.*, vol. 152, pp. 152–159, Mar. 2024.

[20] C. Banbury, E. Njor, A. M. Garavagno, M. Mazumder, M. Stewart, P. Warden, M. Kudlur, N. Jeffries, X. Fafoutis, and V. J. Reddi, "Wake vision: A tailored dataset and benchmark suite for TinyML computer vision applications," 2024, *arXiv:2405.00892*.

[21] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. ICML*, 2019, pp. 6105–6114.

[22] C. Banbury et al., "MLPerf tiny benchmark," in *Proc. NeurIPS Datasets Benchmarks*, 2021, p. 15.

[23] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[24] M. Nagel, M. Fournarakis, R. Ali Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*.

[25] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *Proc. ICLR*, 2019, p. 12.

[26] Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, "HAWQ: Hessian AWare quantization of neural networks with mixed-precision," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 293–302.

[27] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," 2020, *arXiv:2004.09602*.

[28] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[29] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proc. ICCV*, Oct. 2019, pp. 4793–4801.

[30] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1135–1143.

[31] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. ICLR*, 2018, p. 42.

[32] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, and R. Rhodes, "TensorFlow lite micro: Embedded machine learning for TinyML systems," in *Proc. Mach. Learn. Syst.*, vol. 3, 2021, pp. 800–811.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2014, p. 13.
[34] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. ICLR*, 2016, p. 16.
[35] Chirale. (2024). *Chirale_TensorFlowLite: Optimized TensorFlow Lite Micro Library for ESP32*. [Online]. Available: https://github.com/spazi ochirale/Chirale_TensorFlowLite

DANIELE FAGGI received the B.Sc. degree in computer science. He is currently pursuing the M.Sc. degree in artificial intelligence with the Department of Theoretical and Applied Sciences, eCampus University, Novedrate, Italy, under the supervision of Prof. Oleksandr Kuznetsov. His research interests include deep learning for embedded systems, TinyML, the IoT applications, and efficient neural network architectures for resource-constrained devices. His current work focuses on the design and optimization of lightweight AI models for real-time inference on edge hardware, contributing to the research activities of the SMARTEST Research Center, eCampus University.

OLEKSANDR KUZNETSOV (Member, IEEE) received the Doctor of Sciences degree. He was a Senior Data Scientist with Proxima Labs, San Francisco, from 2023 to 2024, developing trustless computation platforms using Zero-Knowledge Proofs, and as a Chief Scientist with Grape Blockchain, London, from 2022 to 2023, pioneering DAG-based consensus mechanisms. He is currently an Associate Professor with the Department of Theoretical and Applied Sciences, eCampus University, Italy; and a Full Professor with the Department of Intelligent Software Systems and Technologies, V. N. Karazin Kharkiv National University, Ukraine. His distinguished career spans over two decades with leadership roles across academia, industry, and defense sectors. His research expertise encompasses applied cryptography, post-quantum security, blockchain and decentralized systems, artificial intelligence in cybersecurity, biometric authentication, steganography, and the IoT security. He is a Laureate of the prestigious Boris Paton National Prize of Ukraine, in 2021, the nation's highest academic distinction in recognition of his contributions to cybersecurity and cryptographic information protection systems.

STEFANO MEREU is currently pursuing the Ph.D. degree with the National Ph.D. Program in Robotics and Intelligent Machines (DRIM), jointly affiliated with the Department of Informatics, Bioengineering, Robotics and Systems Engineering (DIBRIS), University of Genoa, Italy, and the Istituto Italiano di Tecnologia (IIT), Genoa, Italy. His research focuses on Kolmogorov–Arnold Networks and advanced neural architectures, with particular emphasis on their mathematical foundations and methodological innovations, and their application to computer vision and time series analysis. His work also includes 3D vision, point cloud processing, and optimization methods for perception systems, including camera placement optimization and active vision, with applications in robotic inspection and maintenance for industrial and cultural heritage scenarios.

ALESSANDRO GALDELLI (Member, IEEE) received the Ph.D. degree in information engineering from Universitá Politecnica delle Marche, Italy, in 2021. He is currently an Assistant Professor with the Department of Information Engineering (DII), Universitá Politecnica delle Marche. His research focuses on artificial intelligence and computer vision, with particular emphasis on deep learning, machine learning, image processing, and remote sensing. His work addresses the development of intelligent visual systems and advanced neural network models for real-world applications, including edge computing and automated monitoring systems. He has contributed to several research activities in the field of visual data analysis and has authored scientific publications on deep learning–based detection and recognition systems.

EMANUELE FRONTONI (Member, IEEE) is currently a Full Professor of computer science with the University of Macerata and the Co-Director of the VRAI Vision Robotics and Artificial Intelligence Laboratory. He is the author of more than 230 international articles and collaborates with numerous national and international companies in technology transfer and innovation activities. His research interests include computer vision and artificial intelligence with applications in robotics, video analysis, human behavior analysis, extended reality, and digital humanities. He has been the Program Chair or the General Chair of various international conferences and summer schools, such as IEEE / ASME MESA Mechatronic Embedded System and Applications, in 2016 and 2017, IEEE ECMR European Conference on Mobile Robotics, in 2017, BigDat 2020, and DeepLearn 2021; and a Co-Organizer of many international workshops, such as DeepRetail @ ICPR 2020, D2CH @ CVPR 2021, and AI4DH @ ICIAP 2022.

MARCO ARNESANO (Member, IEEE) is currently a Full Professor of the Mechanical and Thermal Measurements (IMIS-01/A) and the Director of the Department of Theoretical and Applied Sciences, eCampus University, Italy. He is the author of more than 100 scientific publications in international journals, conference proceedings, and books, and co-inventor of two patents. He has led research groups within international (FP7, H2020, Horizon Europe) and national (POR, PRIN) funded projects, and provides scientific supervision for innovative companies and startups. His research interests include measurement systems and techniques applied to diagnostics, energy efficiency, environmental monitoring, building performance assessment, and human-environment interaction. His expertise spans IEQ (Indoor Environmental Quality) measurement systems, HVAC controls, physiological measurements, wearable devices, and signal processing and analysis methods. He co-founded two technology transfer startups, MasterBiga srls and LiS-Live Information Systems srls. Since 2021, he has been served as the Technical Program Chair for the international conference MetroLivEnv (Metrology for Living Environments), and has coordinated international focus groups and workshops on efficient systems and Smart Buildings (ForumPiscine 2017–2018, Beyond Energy Efficiency 2017, San Leandro, USA; Energy in Buildings 2017, Athens), contributing technical presentations at major venues including the ASHRAE Annual Conference 2017, Sustainable Places 2018, and the International Building Physics Conference 2018.