Deploying TinyML on Commercial Smart Glasses

Patrick Krumpl[◇], Francesco Corti[◇], Dong Wang[◇], Olga Saukh[◇†]
[◇]Graz University of Technology, [†]Complexity Science Hub Vienna, Austria
{patrick.krumpl@student., francesco.corti@, dong.wang@, saukh@}tugraz.at

Abstract

Smart glasses are increasingly marketed as AI-enabled wearables, yet most current devices offload inference to a paired smartphone or the cloud, limiting privacy, latency, and offline operation. This paper investigates whether useful computer-vision inference can run directly on commercially available, resource-constrained smart glasses. We extend the open-source Brilliant Labs Frame firmware with a TensorFlow Lite Micro inference path accelerated by CMSIS-NN, and deploy two representative workloads: Visual Wake Words classification and FOMO-style centroid-based object detection. Our evaluation shows that on-device vision inference is feasible on an nRF52840-based wearable platform: the Visual Wake Words model reaches 81.6% accuracy on MS COCO minival, exceeding the MLPerf Tiny quality target while using 43% less flash than the quantized reference implementation. CMSIS-NN further reduces inference latency by 13.8× for Visual Wake Words and 9.5× for FOMO compared to reference kernels. Our results demonstrate both the potential and the current limits of TinyML on commercial smart glasses, and provide an open-source firmware basis for future wearable edge-AI research. Our code is available online[1].

CCS Concepts

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Machine learning**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

Keywords

TinyML, smart glasses, wearable computing, on-device inference, Edge AI, TensorFlow Lite Micro, CMSIS-NN

1 Introduction

Smart glasses integrate sensing, computation, and display into everyday eyewear, and recent advances in AI, particularly in computer vision (CV), enable functionalities such as contextual assistance, live translation, and hands-free interaction. However, despite being marketed as AI-enabled devices, most commercially available smart glasses still operate primarily as peripherals that offload computation to a paired smartphone or cloud service due to their limited computational resources and memory budgets [1, 3, 14].

[1]https://github.com/pkrumpl/frame-codebase

Figure 1: Brilliant Labs Frame smart glasses [3] used as the target deployment platform.

Performing inference externally introduces several drawbacks: captured sensor data must leave the device, increasing privacy and security concerns [19], while network communication adds latency and prevents reliable offline operation. On-device inference can mitigate these limitations by moving computation closer to where data is generated, an approach commonly associated with Edge AI and Tiny Machine Learning (TinyML). TinyML focuses on deploying machine learning models directly on ultra-low-power microcontrollers with memory footprints in the kB range and power consumption in the mW range [7, 18]. However, deploying CV workloads on commercial smart glasses remains challenging because most modern vision models exceed the available flash and RAM budgets even after standard compression techniques such as quantization or pruning [15].

Prior work has shown the feasibility of TinyML inference on embedded devices and wearable platforms. Lightweight architectures such as MobileNetV2 [17] and SqueezeNet [10] enable efficient vision inference on mobile and edge devices, while MCUNet demonstrated image classification directly on commercial microcontrollers through system-algorithm co-design [12]. In the context of smart glasses, TinyMM explored multimodal inference on microcontroller-based wearables [6], and TinyissimoYOLO demonstrated real-time object detection on a smart-glasses prototype equipped with a GAP9 RISC-V processor and dedicated neural hardware [16]. Nevertheless, most existing smart glasses either rely on cloud-based inference or employ substantially more capable hardware than typical consumer-grade wearable microcontrollers. As a result, the feasibility and practical limitations

of deploying on-device inference on commercially available low-power smart glasses remain insufficiently explored.

In this paper, we investigate whether commercially available smart glasses with tight resource budgets can serve as a viable platform for on-device vision inference. We extend the firmware of Brilliant Labs' open-source Frame smart glasses with a TensorFlow Lite Micro (TFLM) inference stack accelerated by CMSIS-NN. We deploy and evaluate two representative CV workloads: a Visual Wake Words (VWW) classifier based on MobileNetV1 and a centroid-based object detector derived from the FOMO approach [8, 9]. This paper makes the following contributions:

- We present a firmware-level integration of TFLM with CMSIS-NN on commercially available open-source smart glasses, enabling on-device TinyML inference on an nRF52840-based wearable platform.
- We evaluate two representative vision workloads under strict wearable resource constraints and analyze their memory, latency, and accuracy trade-offs.
- We demonstrate that practical on-device vision inference is feasible on resource-constrained smart glasses, achieving 81.6% VWW accuracy on MS COCO minival while reducing flash usage by approximately 43% compared to the quantized reference implementation.

2 Platform and System Design

2.1 Commercial Smart Glasses as a TinyML Platform

In addition to machine learning inference, consumer wearable devices must simultaneously support communication, sensing, display management, and interactive runtime services within extremely constrained resource budgets.

We target Brilliant Labs' open-source *Frame* smart glasses, shown in Fig. 1, as a representative low-power wearable platform based on the Nordic Semiconductor nRF52840 Bluetooth SoC. Although the MCU provides 1 MB flash and 256 kB SRAM, a large fraction is already occupied by the existing firmware stack, including Bluetooth communication, camera handling, display drivers, the FPGA runtime, and the embedded Lua environment. As a result, the effective TinyML deployment budget shrinks to approximately 270 kB of flash and 200 kB of SRAM, making memory rather than raw compute throughput the dominant bottleneck for on-device inference [2, 12].

2.2 Firmware-Level TinyML Integration

To enable fully on-device inference, we extend the existing Frame firmware with TensorFlow Lite for Microcontrollers (TFLM) [5] and CMSIS-NN [11]. Unlike standalone TinyML demonstrations, the integration must coexist with a production-oriented wearable firmware stack already present on the device.

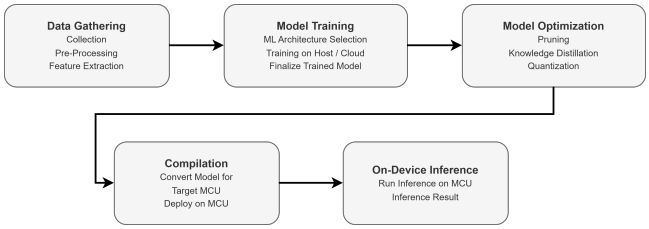

Figure 2: Integrated on-device inference pipeline.

The integration required adapting the firmware build system to support mixed C/C++ compilation, registering only the subset of TFLM operators required by each deployed model, and statically allocating the tensor arena inside the limited SRAM budget. Additional SRAM was reclaimed by disabling the unused microphone driver, recovering approximately 64 kB previously held by audio buffers. Since TFLM avoids dynamic memory allocation, deployment feasibility depends not only on parameter count but also on the peak size of intermediate activation tensors during inference.

The Frame glasses are updated exclusively through Bluetooth DFU because the sealed enclosure prevents physical debugger access; to avoid unrecoverable firmware states during development, we additionally implement a boot-failure recovery mechanism that automatically forces the device back into DFU mode after repeated unsuccessful boots.

A central contribution of this work is demonstrating that fully on-device inference can coexist with a realistic wearable firmware stack without cloud offloading or external compute support. The resulting pipeline (along with the model training, optimization and compilation) performs image acquisition, preprocessing, inference, and display update entirely on-device (Fig. 2).

2.3 Image Acquisition and Preprocessing

The camera pipeline introduces additional system-level constraints beyond neural network inference itself. Images are captured by the FPGA and transferred to the MCU as JPEG-compressed byte streams over SPI. Since the MCU is not directly connected to the image sensor, preprocessing must be implemented entirely in software on the nRF52840.

We use the lightweight TJpgDec decoder [4] to decompress JPEG images directly into reduced-resolution tensors suitable for TinyML inference. Full-resolution RGB frames would require approximately 1.5 MB of RAM, far exceeding the available memory budget. Instead, we exploit TJpgDec's scaled decoding functionality to decode images directly into a reduced intermediate representation before resizing them

to the 96 × 96 model input resolution. The preprocessing pipeline additionally introduces substantial SRAM overhead due to the required JPEG input and output buffers. To avoid heap fragmentation and runtime allocation failures, all image buffers are statically allocated in the `.bss` section together with the TFLM tensor arena, allowing the linker to validate SRAM feasibility at compile time and providing deterministic memory behavior during runtime.

2.4 Workloads

We evaluate the proposed system using two representative computer vision workloads.

Visual Wake Words (VWW). The Visual Wake Words classifier is based on MobileNetV1 [2, 9]. The task performs binary classification of person presence on 96 × 96 RGB inputs and serves as the image classification classification workload evaluated in this work.

FOMO Object Detection. This workload is a centroid-based detector derived from Edge Impulse's FOMO approach [8]. Instead of predicting bounding boxes, FOMO estimates object centroids on a low-resolution feature map, reducing computational complexity and making lightweight object detection feasible on resource-constrained MCUs.

The deployed workloads build on MobileNet-based architectures [9, 17] due to their widespread adoption, available tooling support, and compatibility with TFLM and CMSIS-NN acceleration.

3 Evaluation

3.1 Experimental Setup

We evaluate the TinyML deployment along three dimensions: memory footprint, inference latency, and model accuracy.

Memory. The memory analysis is carried out on the host machine using TFLM's Generic Benchmarking Tool [5]. The tool is executed on the exported `.tflite` model files and reports both the flash footprint of the model and the tensor arena size required during inference.

Latency. Per-model inference latency is measured on the Nordic Semiconductor nRF52840 development kit, which uses the same MCU as the Frame smart glasses while avoiding additional overhead from camera, FPGA, and display peripherals. The end-to-end pipeline latency is measured directly on the Frame glasses.

Accuracy. Model accuracy is evaluated on the host machine. For the MobileNetV1-based VWW classifiers, we follow the MLPerf Tiny benchmarking procedure [2] on the MS COCO minival split [13, 20] and report top-1 accuracy, precision, recall, and F1-score relative to the MLPerf Tiny reference implementations. For the centroid-based FOMO detector [8], no standardized benchmark exists; therefore, we report the same metrics relative to Edge Impulse's reference implementation.

Table 1: Memory footprint of evaluated model variants.

Model	Model [kB]	Arena [kB]
VWW reference float32	846.0	321.0
VWW reference int8	325.5	105.1
VWW RGB int8 ($\alpha = 0.18$)	**186.0**	**99.6**
FOMO reference float32 (96×96)	82.1	899.8
FOMO reference int8 (96×96)	55.6	238.2
FOMO float32 (64×64)	46.5	411.8
FOMO int8 (64×64)	**39.0**	**114.7**

3.2 Memory and Deployment Constraints

Tab. 1 summarizes the memory footprint of the evaluated model variants. The results show that deployment feasibility depends not only on parameter storage but also critically on the tensor arena required during inference. The MLPerf Tiny VWW reference model [2] requires approximately 846 kB as float32, which exceeds the available flash budget. Although int8 quantization reduces the model size substantially, the quantized reference still requires 325.5 kB and therefore does not fit within the practical flash constraints of the platform. In contrast, the custom VWW RGB variant with $\alpha = 0.18$ reduces the model size by approximately 43 % relative to the quantized reference while maintaining competitive accuracy.

For FOMO, the deployment bottleneck is dominated by SRAM rather than flash usage. The original FOMO model with a 96 × 96 input resolution requires nearly 900 kB of SRAM as float32 and still exceeds the available memory budget after int8 quantization with a tensor arena size of 238.2 kB. Reducing the input resolution to 64 × 64 lowers the tensor arena requirement to 114.7 kB, enabling deployment within the available SRAM budget.

Fig. 3 compares the memory footprint of the two deployed showcase models against the effective platform budget. A key observation of this work is that commercially usable wearable firmware significantly changes the effective TinyML deployment budget compared to bare-metal MCU benchmarks commonly used in prior TinyML literature [2, 12]. In particular, intermediate activation tensors become the primary SRAM bottleneck for computer vision workloads once realistic firmware overheads are taken into account.

The results further show that int8 quantization does not achieve the theoretical 4× reduction in model size because the FlatBuffer representation additionally stores network topology and metadata, which remain unchanged under

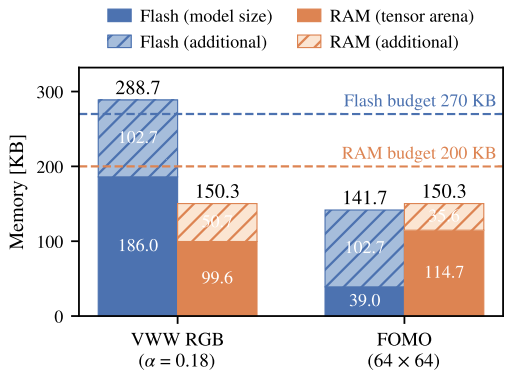

Figure 3: Memory footprint of the deployed VWW RGB and FOMO models against the available platform budgets. Additional flash and SRAM overheads originate from the TFLM runtime, CMSIS-NN backend, and image preprocessing pipeline.

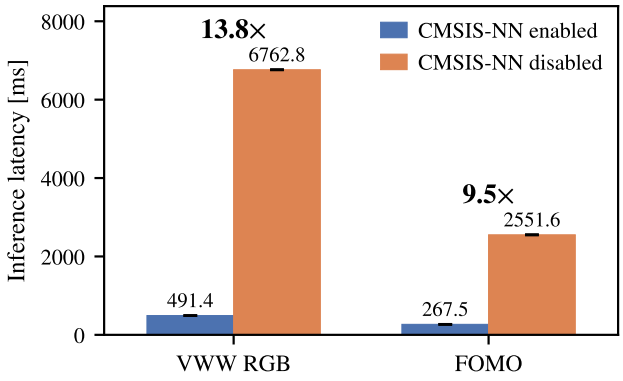

Figure 4: Inference latency for the deployed VWW RGB and FOMO models with and without CMSIS-NN.

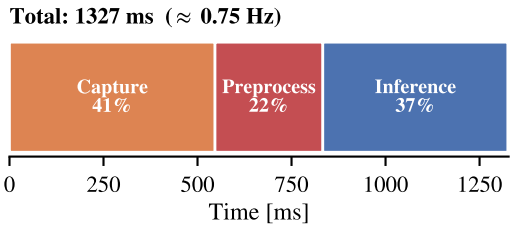

Figure 5: End-to-end pipeline latency breakdown for the deployed VWW RGB model on the Frame glasses.

quantization. Similarly, the tensor arena reduction is limited by TFLM interpreter state and operator metadata that scale with the network structure rather than weight precision.

The TFLM runtime and CMSIS-NN backend together contribute approximately 102 kB of constant flash overhead. Furthermore, the image preprocessing pipeline requires approximately 35–51 kB additional SRAM depending on the model input resolution.

Overall, the results reveal fundamentally different architectural bottlenecks for the two workloads. The VWW model, based on MobileNetV1 [9], is constrained primarily by flash usage caused by parameter-heavy pointwise convolutions, whereas the FOMO model, based on MobileNetV2 [17], is dominated by SRAM usage caused by expanded intermediate activation tensors within inverted residual blocks.

3.3 Inference Performance

Fig. 4 reports the latency of a single inference for all deployed models with and without CMSIS-NN acceleration. Tab. 2 reports the corresponding means and standard deviations. The VWW RGB and FOMO models achieve sub-second inference latency when CMSIS-NN is enabled, whereas both exceed two seconds when using TFLM's portable reference kernels. The VWW RGB model requires 491.4 ms per inference with CMSIS-NN enabled compared to 6762.8 ms without acceleration, corresponding to a 13.8× speedup. Similarly, the deployed FOMO model achieves a 9.5× speedup, reducing latency from 2551.6 ms to 267.5 ms. The measured speedups are larger than previously reported CMSIS-NN improvements [11] because our baseline uses TFLM's portable reference kernels instead of CMSIS-DSP convolution implementations. All inference latency measurements are obtained from the cycle-accurate Cortex-M4 Data Watchpoint and Trace (DWT) cycle counter after 10 untimed warmup inferences. Overall, the results demonstrate that optimized Cortex-M kernels are essential for practical TinyML deployment on wearable MCU platforms.

We additionally evaluate the complete end-to-end pipeline including image acquisition, preprocessing, inference, and display update. Fig. 5 shows the latency breakdown for the deployed VWW RGB workload. One complete pipeline iteration requires approximately 1.3 s, corresponding to a frame rate of approximately 0.75 Hz. The end-to-end pipeline is measured for the VWW workload only, since the capture, preprocessing, and display stages are shared across CV tasks. The measurement relies on the same DWT cycle counter and comprises 30 timed iterations preceded by 5 untimed warmup iterations, with the camera exposure fixed after an initial auto-exposure convergence phase. Image acquisition accounts for approximately 41 % of the total runtime due to blocking polling from the FPGA, while image preprocessing contributes another 22 % because of the FPGA-imposed JPEG conversion pipeline. Together, these stages consume nearly two thirds of the total latency, whereas the display update

Table 2: Latency of the deployed models and of the end-to-end on-device pipeline.

Model / pipeline stage	Mean [ms]	Std. dev. [ms]
Model inference (nRF52840 DK)		
VWW RGB ($\alpha = 0.18$), CMSIS-NN	491.36	0.18
VWW RGB ($\alpha = 0.18$), reference	6762.82	0.15
FOMO (64×64), CMSIS-NN	267.49	0.15
FOMO (64×64), reference	2551.62	0.23
End-to-end pipeline (Frame, VWW RGB, CMSIS-NN)		
Image acquisition	546.57	21.09
Preprocessing	286.67	18.13
Inference	492.39	0.11
Display update	1.63	0.02
Total	**1327.26**	**17.16**

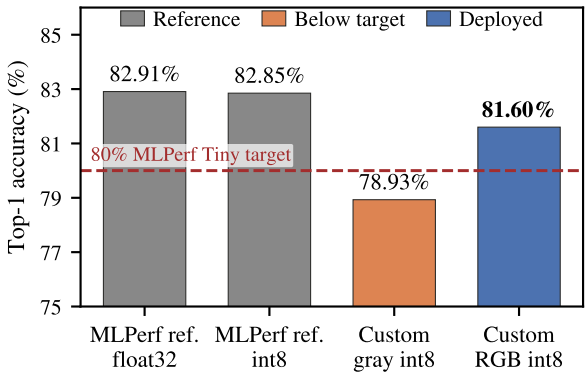

Figure 6: Top-1 accuracy of the MLPerf Tiny VWW reference implementations and custom variants.

contributes negligibly. We note that a ~1.3 s end-to-end latency is not sufficient for interactive real-time applications. We therefore position this work as an empirical feasibility study of on-device inference on commercial smart glasses, rather than as a deployment-ready consumer system. These results demonstrate that practical wearable TinyML deployment is constrained not only by neural network inference itself but also significantly by surrounding sensor and preprocessing pipelines.

3.4 Accuracy

Fig. 6 compares the top-1 accuracy of the MLPerf Tiny VWW reference implementations [2] against our custom variants. The deployed RGB model achieves 81.60 % top-1 accuracy, surpassing the 80 % MLPerf Tiny target while remaining only 1.25 % below the quantized reference implementation.

Tab. 3 summarizes the detailed accuracy metrics together with the corresponding flash footprint. The deployed RGB variant achieves competitive accuracy while requiring about 43 % less flash memory than the quantized MLPerf Tiny reference implementation. Beyond top-1 accuracy, the models also differ in their precision-recall trade-offs. The MLPerf Tiny references prioritize precision over recall, whereas the deployed RGB variant achieves a more balanced operation, resulting in a slightly higher F1-score. The preferable trade-off depends on the target application: high precision reduces false activations, while balanced precision and recall reduce missed detections. At 186 kB, the deployed RGB variant is the smallest evaluated model that still exceeds the 80 % MLPerf Tiny target, demonstrating that the VWW task can be deployed on the Frame glasses at approximately 57 % of the flash footprint of the quantized reference implementation.

Table 3: Comparison of flash footprint and accuracy metrics for evaluated VWW models on the MS COCO minival split [20].

Model	Flash (kB)	Top-1 (%)	Precision (%)	Recall (%)	F1 (%)
MLPerf reference ($\alpha = 0.25$, RGB, float32)	846.0	82.91	91.91	71.58	80.48
MLPerf reference ($\alpha = 0.25$, RGB, int8)	325.5	82.85	91.62	71.71	80.45
Custom ($\alpha = 0.18$, gray, int8)	185.9	78.93	79.92	76.37	78.11
Custom ($\alpha = 0.18$, RGB, int8)	**186.0**	**81.60**	**81.49**	**81.01**	**81.25**

Table 4: Accuracy metrics of evaluated FOMO models.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
FOMO reference (96×96, gray, int8)	91.67	94.00	97.00	96.00
FOMO retrained (64×64, gray, int8)	**85.71**	**88.00**	**87.00**	**88.00**

Tab. 4 compares the deployed FOMO model against the original Edge Impulse demo model [8]; both models are evaluated using Edge Impulse Studio's integrated testing pipeline on the demo dataset. The retrained 64×64 FOMO model achieves nearly 6 % lower accuracy than the original demo model. However, it is the only evaluated variant that satisfies the memory constraints of the target platform while still enabling practical on-device object detection.

4 Discussion, Limitations, and Outlook

Our results demonstrate that fully on-device computer vision inference is feasible on commercially available smart glasses despite extremely constrained hardware resources.

At the same time, the evaluation reveals that practical deployment bottlenecks differ substantially from those typically emphasized in bare-metal TinyML benchmarks.

A key observation is that inference itself is not the only limiting factor for wearable TinyML systems. Although CMSIS-NN acceleration reduces inference latency by up to 13.8×, the end-to-end pipeline remains dominated by image acquisition and preprocessing overheads imposed by the JPEG-over-SPI camera pipeline. This suggests that wearable TinyML platforms may benefit at least as much from improved sensor interfaces, hardware image scaling, and more efficient memory hierarchies as from dedicated neural accelerators alone.

The results further highlight the importance of workload-specific optimization. The VWW classifier and the FOMO detector expose fundamentally different resource bottlenecks despite operating on the same hardware platform. While the MobileNetV1-based VWW workload is constrained primarily by flash usage, the MobileNetV2-based FOMO workload is dominated by SRAM consumption caused by expanded intermediate activation tensors. These findings indicate that model architecture choices for wearable TinyML systems must consider not only parameter count and compute complexity but also activation memory behaviour under realistic firmware constraints.

More broadly, consumer smart glasses are typically designed assuming computation happens elsewhere. Bringing ML on-device therefore requires firmware integration that was never part of the original product specification, making wearable on-device machine learning a systems-integration problem at least as much as a model-design one.

Limitations. First, the presented workloads do not achieve real-time performance, with the full inference pipeline operating at approximately 0.75 Hz. Second, the evaluation focuses on two representative vision workloads and does not cover additional sensing modalities such as audio or multimodal fusion. Third, we do not evaluate energy consumption, thermal behaviour, or long-term continuous operation, which are important considerations for practical wearable deployment. Finally, the experiments are performed on a single hardware platform and therefore may not directly generalize to smart glasses equipped with dedicated NPUs or more capable application processors.

Outlook. The presented firmware integration provides an open-source baseline for future research on wearable TinyML systems. Future work could explore more efficient image acquisition pipelines, event-based or low-resolution sensors, adaptive inference strategies, TinyViT-style architectures, and multimodal on-device assistants combining vision, audio, and inertial sensing. From a benchmarking perspective, our findings suggest that evaluating TinyML systems solely on MCU benchmarks risks overlooking critical system-level constraints that emerge in realistic commercial wearables.

Acknowledgments. This research was funded in part by the Austrian Science Fund (FWF) within the DENISE doctoral school (grant DOI: 10.55776/DFH5) and the FFG COMET K1 Center "Pro^{2}Future II" (Cognitive and Sustainable Products and Production Systems of the Future), Contract No. 911655.

References

[1] Apple Inc. 2025. Apple Vision Pro. https://www.apple.com/apple-vision-pro/ Accessed: 2026-02-17.

[2] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, et al. 2021. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597* (2021).

[3] Brilliant Labs. 2025. *Frame - Brilliant Documentation.* Brilliant Labs. https://docs.brilliant.xyz/frame/frame/

[4] ChaN. [n. d.]. *TJpgDec – Tiny JPEG Decompressor.* http://elm-chan.org/fsw/tjpgd/ Accessed: 2026-04-23.

[5] Robert David et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *ML and systems* 3 (2021), 800–811.

[6] Lokmane Demagh, Patrick Garda, Cedric Gilbert, and Khalil Hachicha. 2023. TinyMM: Multimodal-Multitask Machine Learning on Low-Power MCUs for Smart Glasses. In *2023 IEEE SENSORS*. IEEE, 1–4.

[7] Lachit Dutta and Swapna Bharali. 2021. Tinyml meets iot: A comprehensive survey. *Internet of Things* 16 (2021), 100461.

[8] Edge Impulse. 2026. FOMO – Edge Impulse Documentation. https://docs.edgeimpulse.com/studio/projects/learning-blocks/blocks/object-detection/fomo Accessed: 2026-03-16.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, et al. 2017. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* arXiv:1704.04861 [cs.CV]

[10] Forrest N Iandola, Song Han, Matthew W Moskewicz, et al. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv:1602.07360* (2016).

[11] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601* (2018).

[12] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, et al. 2020. MCUnet: Tiny deep learning on iot devices. *NeurIPS* 33 (2020), 11711–11722.

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, et al. 2014. Microsoft COCO: Common objects in context. In *ECCV*. Springer, 740–755.

[14] Meta. 2025. New Meta Ray-Ban AI-Powered Display Glasses. https://www.meta.com/ai-glasses/meta-ray-ban-display/ Product page.

[15] MLCommons. 2025. MLPerf Inference: Tiny Benchmark Suite. https://mlcommons.org/benchmarks/inference-tiny/ Accessed: 2026-03-14.

[16] Julian Moosmann, Pietro Bonazzi, Yawei Li, et al. 2025. Ultra-Efficient On-Device Object Detection on AI-Integrated Smart Glasses with TinyissimoYOLO. In *ECCV 2024 Workshops*. Cham, 262–280.

[17] Mark Sandler, Andrew Howard, et al. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE CVPR*. 4510–4520.

[18] Nikolaos Schizas, Aristeidis Karras, Christos Karras, and Spyros Sioutas. 2022. TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. *Future Internet* 14, 12 (2022), 363.

[19] Svenska Dagbladet. 2026. *Meta's AI Smart Glasses and Data Privacy Concerns: Workers Say "We See Everything".* Svenska Dagbladet. https://www.svd.se/a/K8nrV4/metas-ai-smart-glasses-and-data-privacy-concerns-workers-say-we-see-everything

[20] TensorFlow Authors. 2018. MSCOCO minival image IDs. https://github.com/tensorflow/models/blob/master/research/object_detection/data/mscoco_minival_ids.txt.