RESEARCH CASE STUDY · PRE-HARDWARE PHASE

Edge-inference energy benchmark

TINYML AND EMBEDDED INFERENCE · INDEPENDENT RESEARCH · 2026–PRESENT

O.J. Akojede[1] and T.E. Hamzat[2]
[1] Independent Researcher, Lagos, Nigeria
[2] Energhx Research Group, Department of Mechanical Engineering, University of Lagos, Nigeria

Project phase: Pre-hardware study design, model preparation, static analysis and Raspberry Pi deployment validation completed. Physical energy measurement is the next phase.

Abstract

Operation counts are widely used to estimate the efficiency of neural networks before deployment. They describe arithmetic workload, but the energy consumed during inference also depends on the movement of model parameters and intermediate tensors through a hardware system. This distinction becomes important at the edge, where the same model may execute on devices with different memory hierarchies and runtimes.

This case study documents the completed pre-hardware phase of a controlled experiment designed to compare operation-based metrics with board-level inference energy. A corpus of 43 neural networks was trained across four edge-inference tasks. Each selected checkpoint was converted into FP32 and full-integer INT8 TensorFlow Lite artifacts, producing 86 Raspberry Pi deployment conditions. Static graph analysis was used to calculate arithmetic and memory-facing descriptors before energy results were inspected.

The pre-hardware results establish a controlled test set in which models with nearly equal multiply-accumulate counts have markedly different estimated activation demand. In the closest CIFAR-10 and keyword-spotting pairs, MAC counts differ by 1.074% and 0.203%, while the corresponding depthwise models produce approximately four times as many intermediate activation bytes. All 86 Raspberry Pi artifacts completed deployment smoke testing, and the fixed measurement banks completed 10,320 validation invocations. These results provide the experimental basis for the hardware phase, in which latency and board-input energy will be measured on a Raspberry Pi and an ESP32-S3.

Keywords: edge inference; energy measurement; operation counts; TinyML; TensorFlow Lite; embedded machine learning

1. Research problem

Inference is increasingly performed on microcontrollers and single-board computers rather than exclusively in cloud infrastructure. Local execution can reduce communication latency and

dependence on a network, but it places model selection within a limited energy and memory budget. Efficiency therefore becomes part of whether a model can be deployed, rather than merely a desirable optimization.

Floating-point operations and multiply-accumulate operations are commonly used to compare model complexity because they can be computed without access to the final device. Parameter count and artifact size are used for the same reason. These quantities are useful descriptions of a model, but they do not directly measure energy.

The physical cost of inference is produced by the interaction between a deployed graph and its execution system. Arithmetic contributes to that cost, as do memory access, activation lifetime, operator implementation and runtime scheduling [1,2]. A depthwise-separable network may require fewer MACs than a standard-convolution network while creating a different sequence of intermediate tensors. The difference may be insignificant on one processor and consequential on another.

Prior studies reach different conclusions about the usefulness of operation count. FLOPs have shown weak correspondence with measured energy in some machine-learning settings, while tightly constrained microcontroller search spaces have shown stronger relationships between operation count, latency and energy [3,4]. Mobile system-on-chip measurements likewise indicate that hardware-aware predictors can outperform FLOPs alone [5]. These results suggest that the validity of an operation-count proxy depends on the models being compared and the platform executing them.

The study therefore asks:

1. How strongly do FLOPs and MACs correspond with measured inference energy on representative edge platforms?
2. Does that relationship change between a microcontroller and a Linux-based single-board computer?
3. Do memory-facing metrics or measured latency explain energy more effectively than operation count alone?
4. When MAC counts are closely matched, do differences in graph structure expose failures in arithmetic-only estimates?

---

2. Experimental design

The experiment is divided into a predictor phase and a measurement phase. The predictor phase fixes the data partitions, model corpus, task-quality criteria, deployment artifacts, static metrics and controlled comparisons before physical energy results are inspected. The measurement phase executes those frozen artifacts without retraining or graph modification.

The experimental unit is a model–platform pair: one artifact executed under one documented hardware and runtime configuration. The principal cross-platform condition uses the same full-integer INT8 TensorFlow Lite artifact on both platforms whenever deployment succeeds. FP32 measurements on the Raspberry Pi provide a within-platform precision comparison rather than a second common cross-platform condition.

Repeated measurement windows will quantify technical variation within each condition. They will not be treated as additional independent models. Trial order is randomized within each complete repetition so that one architecture or precision is not consistently measured earlier or at a lower device temperature.

---

3. Model corpus

The canonical corpus contains 43 trained neural networks. Four models act as application-oriented reference-style workloads: a ResNet-8-style image classifier, a DS-CNN keyword-spotting network, a width-reduced MobileNetV2 visual-wake-word model and a fully connected ToyADMOS autoencoder. These workloads correspond to task categories represented in MLPerf Tiny, although they are described as benchmark-aligned unless an official architecture and protocol are reproduced exactly [4].

The remaining 39 models were built as structural probes. Thirty-eight form eight replicated width families; one dense keyword-spotting bottleneck acts as a singleton control. The families vary convolution type, activation resolution, channel width, residual structure and parameter concentration while retaining a valid inference task.

Model family	Task	Models	Experimental role
Standard-convolution CNN	CIFAR-10	7	Arithmetic-dense width series
Depthwise-separable CNN	CIFAR-10	7	Primary contrast with standard convolution
High-resolution/low-channel CNN	CIFAR-10	4	Sustained spatial-activation probe
Early-downsampling CNN	CIFAR-10	4	Activation-lifetime contrast
Multilayer perceptron	CIFAR-10	4	Parameter-heavy, non-convolutional probe
Bottleneck-residual CNN	CIFAR-10	4	Skip-connection and tensor-liveness probe
Standard-convolution KWS CNN	Speech Commands	4	Audio-domain standard-convolution series
Depthwise-separable KWS CNN	Speech Commands	4	Audio-domain depthwise comparison
Dense KWS bottleneck	Speech Commands	1	Parameter-concentration control
Reference-style workloads	Four tasks	4	Application-oriented anchors

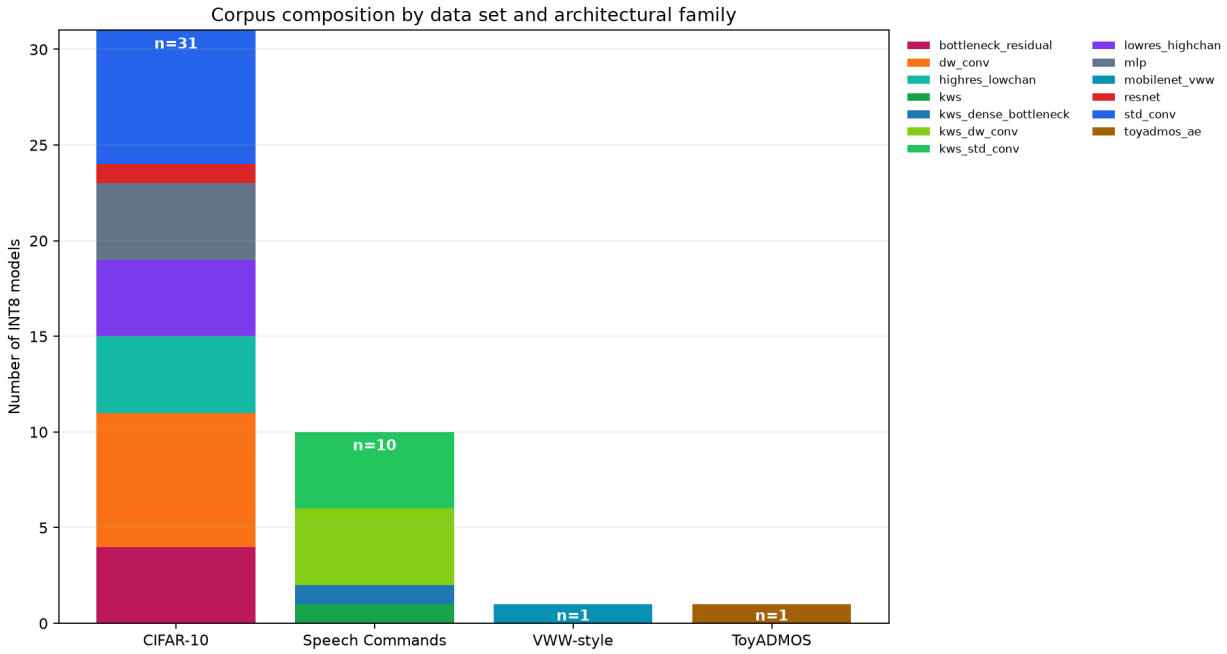


Figure 1. Composition of the 43-model INT8 corpus. CIFAR-10 and Speech Commands provide the replicated structural comparisons; the VWW-style and ToyADMOS models broaden the task and operator coverage.

The corpus is intentionally structured rather than balanced by data set. CIFAR-10 supports most controlled families because its fixed input and task make architectural variation easier to interpret. Speech Commands provides an independent domain for the standard-versus-depthwise comparison. VWW-style person detection and ToyADMOS anomaly detection broaden the represented inputs but are not used to infer a general data-set effect.

---

4. Data preparation and model deployment

All stochastic split and selection procedures use recorded seeds, and the resulting indices are retained. Normalization statistics are estimated from model-fitting data only. Held-out test data do not contribute to training, early stopping or post-training quantization calibration.

4.1 Inference tasks

CIFAR-10. The official training set was divided into 45,000 fitting and 5,000 validation images by seeded permutation. Images were scaled and standardized using fitting-partition statistics. The official 10,000-image test split was retained for final evaluation [6].

Speech Commands. Version 0.02 was partitioned with its published validation and test lists. One-second, 16 kHz recordings were represented as 49 × 10 mel-frequency cepstral coefficient matrices. The fitting, validation and test sets contain 36,923, 4,443 and 4,888 examples across 12 classes [7].

Visual Wake Words. A balanced person-detection subset was constructed from COCO 2014. Images were resized to 96 × 96 pixels. The 16,000-image fitting subset and disjoint 2,000-image

validation and test subsets retain person-size and difficult-negative strata. Results are described as VWW-style rather than official benchmark scores [8,9].

ToyADMOS. ToyCar recordings were transformed into normalized 640-element spectral vectors. The autoencoder was trained on normal examples and evaluated by reconstruction error, AUC and partial AUC [10].

Task metrics establish that each network performs meaningful inference; they are not pooled into a single performance scale. Classification accuracy and anomaly-detection AUC answer different questions and are retained within their respective tasks.

4.2 Frozen deployment artifacts

Every selected checkpoint was converted into an FP32 TensorFlow Lite artifact and a full-integer INT8 artifact. INT8 conversion was restricted to integer TensorFlow Lite operators and required signed INT8 input and output tensors. This provides a common numerical representation for TensorFlow Lite Micro on the ESP32-S3 and LiteRT on the Raspberry Pi [11,12].

Quantization calibration used a deterministic, training-only representative set of no more than 300 examples per model. The FP32 and INT8 conditions use the same held-out examples in the same order. Input conversion is performed before a measurement window using the scale and zero point stored in each artifact.

The converted graph is treated as the deployment identity. Static metrics are calculated from the deployed operators and tensor shapes wherever possible rather than inferred only from the training framework.

---

5. Static predictors

Deployed-graph MAC count is the primary arithmetic predictor. One multiplication followed by one accumulation is counted as one MAC; the same work is represented as two operations when reported as FLOPs. The graph profiler also records the quantities below.

Static quantity	Definition	Interpretation
Deployed-graph MAC count	MACs summed across supported operators	Arithmetic workload
Static byte-access proxy	Input + output + twice the intermediate activation bytes + unique parameter bytes	Idealized data-access demand
Activation-output bytes	Intermediate operator-output tensor sizes	Intermediate activation volume
Parameter footprint	Unique weight and bias tensors	Stored model data
Peak live tensor estimate	Maximum estimated simultaneously live tensor storage	Working-memory pressure
TensorFlow Lite artifact size	Size of the deployed FlatBuffer	Deployment storage
Arithmetic intensity	MAC count divided by the static byte-access proxy	Estimated work per byte
Operator composition	Work distributed across operator types	Computational structure

Static quantity	Definition	Interpretation
BOPs	MACs × weight bits × activation bits	Precision-adjusted operation descriptor

The memory-facing proxy is calculated as:

```
static byte-access proxy = input bytes + output bytes
                         + (2 × intermediate activation-output bytes)
                         + unique parameter bytes
```

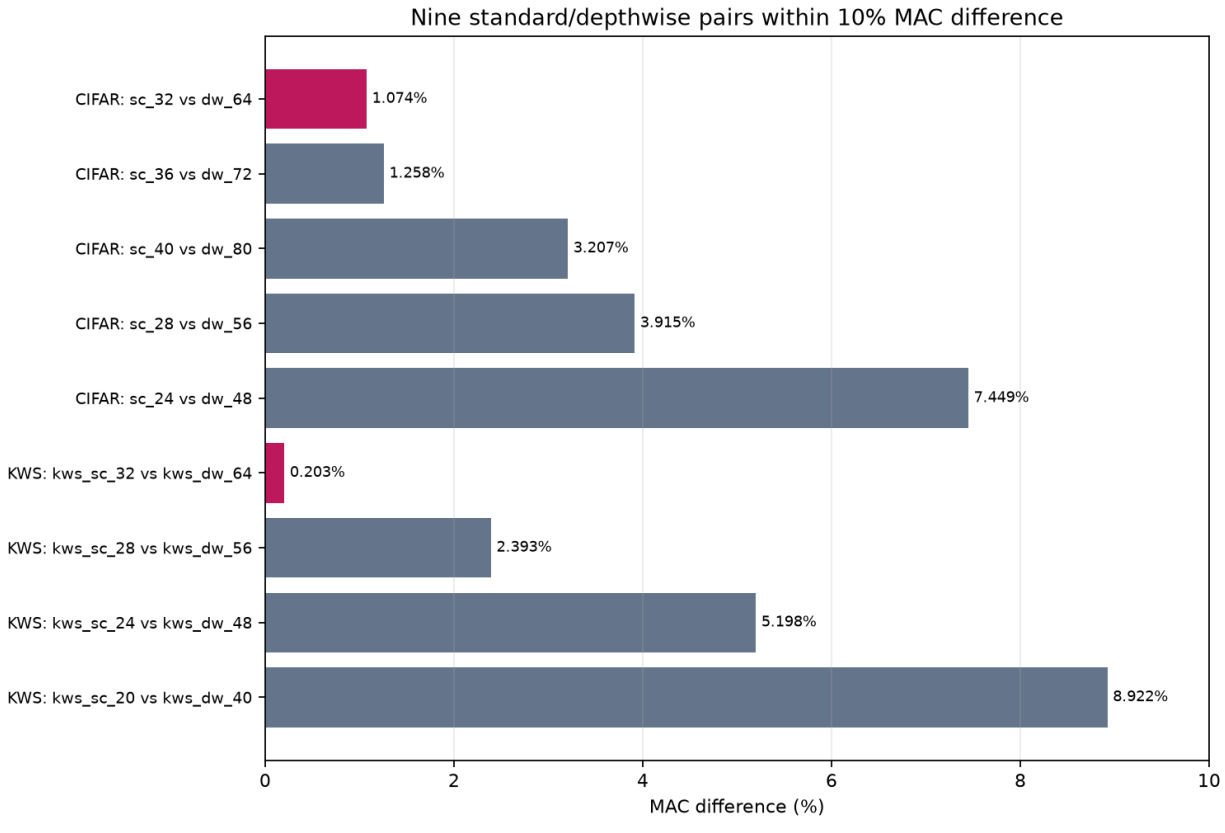
The factor of two represents one write and one later read for each intermediate output. This is a graph-derived lower-bound proxy, not a measurement of physical memory traffic. Cache reuse, tensor allocation and runtime-specific fusion will be addressed through the relationship between static predictors and the subsequent physical measurements.

Within a single precision, BOPs are a constant multiple of MAC count and cannot provide an independent model ranking. Their main analytical role is therefore the FP32–INT8 comparison.

---

6. Controlled pre-hardware findings

Nine standard/depthwise pairs have MAC differences below 10%. These pairs provide a controlled test of whether similar arithmetic workload produces similar energy when estimated activation demand differs.



Magenta marks the closest prespecified pair in each task; grey pairs form the width-series sensitivity set.

Figure 2. Relative MAC difference for the prespecified standard/depthwise comparisons. The closest pair in each task is highlighted.

Standard/depthwise pairs within 10% MAC difference

Standard model	Depthwise model	Standard MACs	Depthwise MACs	MAC difference	Standard static bytes	Depthwise static bytes	Static-byte increase (%)	Activation-byte increase (%)
std_conv_32	dw_conv_64	7,963,264	8,049,280	1.074%	193,850	539,610	178.4%	288.9%
std_conv_36	dw_conv_72	9,954,000	9,829,584	1.258%	225,462	608,394	169.8%	288.9%
std_conv_40	dw_conv_80	12,165,920	11,781,920	3.207%	258,802	677,562	161.8%	288.9%
std_conv_28	dw_conv_56	6,193,712	6,441,008	3.915%	163,966	471,210	187.4%	288.9%
std_conv_24	dw_conv_48	4,645,344	5,004,768	7.449%	135,810	403,194	196.9%	288.8%
kws_std_conv_32	kws_dw_conv_64	3,238,464	3,245,056	0.203%	119,030	262,102	120.2%	297.5%
kws_std_conv_28	kws_dw_conv_56	2,494,968	2,555,392	2.393%	98,178	228,070	132.3%	297.4%
kws_std_conv_24	kws_dw_conv_48	1,848,240	1,946,880	5.198%	79,054	194,422	145.9%	297.4%
kws_std_conv_20	kws_dw_conv_40	1,298,280	1,419,520	8.922%	61,658	161,158	161.4%	297.3%

Figure 3. MAC count, static byte-access proxy and activation-output differences for the nine matched pairs.

The closest CIFAR-10 pair is `std_conv_32` versus `dw_conv_64`. Their MAC counts differ by 1.074%, but the depthwise model produces 288.9% more intermediate activation-output bytes and has a 178.4% larger static byte-access proxy.

The closest Speech Commands pair is `kws_std_conv_32` versus `kws_dw_conv_64`. Their MAC counts differ by 0.203%, while the depthwise model produces 297.5% more activation-output bytes and has a 120.2% larger static byte-access proxy.

These are not energy results. They establish controlled conditions in which an arithmetic-only metric ranks two models as nearly equal even though their deployed graphs have different estimated data-access demands. Whether those differences affect latency and energy is the question carried into hardware measurement.

Across the full corpus, activation-output bytes and total static bytes have a moderate positive association: Pearson $r = 0.504$ and Spearman $\rho = 0.546$. The relationship is incomplete because parameter-heavy networks can have large static-byte estimates without large intermediate activations.

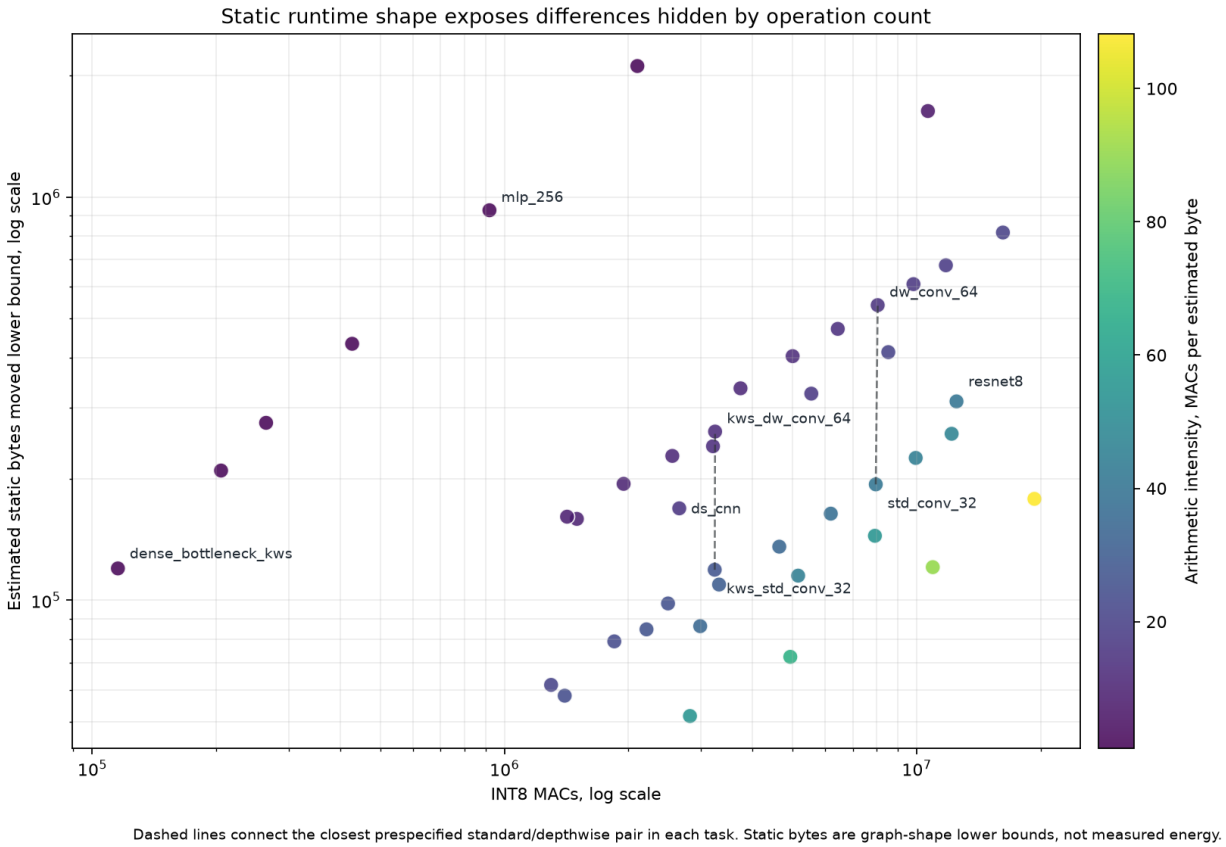


Figure 4. The 43 deployed INT8 graphs occupy different static runtime shapes. Dashed lines mark near-MAC examples; colour represents arithmetic intensity.

A broader descriptive screen identified 35 standard/depthwise combinations with MAC differences below 50%. The depthwise model had the larger static byte-access proxy in every combination, with increases from 63.3% to 271.0%. Thirty-two of the 35 combinations had at least twice the estimated static-byte demand. Because models recur across combinations, this screen describes the available comparison space and is not treated as 35 independent tests.

---

7. Hardware preparation

The hardware phase compares a Raspberry Pi 4 with an ESP32-S3 development board. The platforms are not forced to share an internal runtime because their architectural differences form part of the research question. The common condition instead fixes the INT8 artifact, input values and measurement principles.

7.1 Raspberry Pi execution condition

The Raspberry Pi uses a 64-bit Linux environment on the Arm Cortex-A72 processor. TensorFlow Lite artifacts execute through LiteRT 2.1.6 with the XNNPACK CPU backend and one inference thread. The inference and logging processes are assigned to separate CPU cores. The performance governor requests a fixed 1.8 GHz minimum and maximum frequency, with the resulting frequency

and throttling state read back during trials. An independently powered fan is used for cooling and remains outside the power-measurement boundary.

Board-input power is monitored by placing an INA226 module in the positive 5 V supply path:

```
5 V source → INA226 Current+ → 0.002 Ω shunt → INA226 Current− → Raspberry Pi 5 V input
```

The measurement boundary includes the processor, memory and on-board regulation. It excludes the external fan and upstream supply losses, so results will be reported as board-input energy rather than processor-core energy.

The INA226 is addressed through Raspberry Pi I²C bus 1. A 9 ms conversion interval provides approximately 111 result updates per second, while the logger polls the latest power result every 2 ms. Each observation receives a monotonic timestamp for numerical energy integration.

Electrical-zero behaviour was characterized with the Raspberry Pi load disconnected and an Arduino Nano acting as the I²C host. Four repeats of 4,000 samples produced a combined apparent-current mean of −0.070 mA, no read failures and a between-repeat standard deviation of 0.006 mA. This offset is small relative to Raspberry Pi operating current and is expected to cancel substantially through active-minus-idle subtraction.

7.2 Fixed measurement inputs

Four deterministic held-out measurement banks were prepared, one for each task. Each contains 120 examples. CIFAR-10 contributes 12 examples from every class; Speech Commands contributes 10 from each of 12 classes. The VWW-style bank is balanced between person and no-person images. ToyADMOS contributes 15 examples from every normal/anomalous and machine-ID combination.

These banks are measurement stimuli rather than training or calibration data. Every artifact for a task receives the same ordered floating-point values. INT8 conversion, file loading and reshaping occur before the measured window so that preprocessing does not become part of the reported inference energy.

7.3 Deployment validation

All 86 Raspberry Pi artifacts—43 FP32 and 43 INT8—completed deployment smoke testing. The fixed input banks completed 10,320 validation invocations without an invocation failure, non-finite output or complete INT8 input saturation. Artifact format, tensor metadata, allocation, input conversion and inference completion were checked before the formal energy runs.

The ESP32-S3 condition will use full-integer inference through TensorFlow Lite Micro. Its toolchain, kernel implementation, processor settings and tensor arena will be fixed before formal measurement. Successful deployment will define the common cross-platform set; any unsupported model will remain part of the platform-suitability record.

8. Next phase

The completed phase has produced the trained corpus, frozen deployment artifacts, static predictor table, prespecified comparisons, measurement inputs and Raspberry Pi runtime controls. Hardware measurement now begins with formal Raspberry Pi trials, followed by ESP32-S3 deployment and measurement.

Each experimental repetition will contain every planned condition once in randomized order. The formal trial register will retain the model, precision, execution order, inference count, phase duration, energy, latency, voltage, temperature, frequency and runtime-control fields. Active-window energy will be integrated from timestamped power samples and interpreted against the paired idle condition.

The final analysis will compare MACs and FLOPs with directly measured energy, then test whether latency or memory-facing descriptors better explain the observed variation. The near-MAC pairs provide the most direct evaluation: arithmetic workload is held approximately constant while the static runtime shape changes.

References

1. Horowitz, M. (2014). Computing's energy problem (and what we can do about it). *IEEE International Solid-State Circuits Conference*.
2. Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12), 2295–2329. https://doi.org/10.1109/JPROC.2017.2761740
3. Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020). Towards the systematic reporting of the energy and carbon footprints of machine learning. arXiv:2002.05651.
4. Banbury, C., Reddi, V. J., Torelli, P., et al. (2021). MLPerf Tiny Benchmark. *NeurIPS Datasets and Benchmarks*.
5. Tu, X., Mallik, A., Chen, D., et al. (2023). Unveiling energy efficiency in deep learning: Measurement, prediction, and scoring across edge devices. *ACM/IEEE Symposium on Edge Computing*, 80–93.
6. Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*.
7. Warden, P. (2018). Speech Commands: A dataset for limited-vocabulary speech recognition. arXiv:1804.03209.
8. Lin, T.-Y., Maire, M., Belongie, S., et al. (2014). Microsoft COCO: Common objects in context. *European Conference on Computer Vision*.
9. Chowdhery, A., Warden, P., Shlens, J., Howard, A., & Rhodes, R. (2019). Visual Wake Words Dataset. arXiv:1906.05721.
10. Koizumi, Y., Saito, S., Uematsu, H., Harada, N., & Imoto, K. (2019). ToyADMOS: A dataset of miniature-machine operating sounds for anomalous sound detection. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*.
11. Jacob, B., Kligys, S., Chen, B., et al. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
12. David, R., Duke, J., Jain, A., et al. (2020). TensorFlow Lite Micro: Embedded machine learning on TinyML systems. arXiv:2010.08678.