LiteML-Edge: A Contract-Centered Framework and Methodology for Replay-Driven 1:1 Python–Firmware Validation in TinyML

Charles Pereira dos Santos, Israel Gondres Torné, *Senior Member, IEEE*, Carlos Maurício Seródio Figueiredo, and Fábio de Sousa Cardoso

***Abstract*—Tiny Machine Learning (TinyML) enables data-driven inference on resource-constrained microcontrollers; however, offline evaluation and firmware execution are often treated as loosely coupled stages, leaving reproducibility gaps between reference experiments and on-device behavior. This paper presents LiteML-Edge, a contract-centered TinyML framework and methodology for replay-driven 1:1 Python–Firmware Validation. LiteML-Edge integrates training, optimization, quantization, artifact export, firmware deployment, and per-stage validation while exporting replay artifacts, preprocessing rules, tensor I/O metadata, output mapping, post-processing, and the Rolling-24 metric policy as a shared validation contract. The framework supports fully quantized, fully floating-point, and hybrid tensor interfaces within a single workflow. The evaluation covers one environmental sensing scenario, a fixed $W = 24$ policy, and the TensorFlow-to-TFLM deployment path. On a WEMOS LOLIN32 (ESP32) and a NUCLEO-F411RE (ARM Cortex-M4), case studies based on a multilayer perceptron (MLP), a long short-term memory network (LSTM), and a compact one-dimensional convolutional model (Conv1D Tiny) showed how the framework localizes agreement or mismatch across preprocessing inputs, model payloads, raw outputs, final predictions, and Rolling-24 metrics. Under replay on both targets, MLP and Conv1D Tiny achieved contract agreement across the critical path, whereas LSTM preserved input-stage agreement but showed its first reproducible divergence at the raw-output stage. Field execution is retained as operational evidence collected under the same contract, complementing replay without replacing it for equivalence claims. The resulting workflow makes validation failures more attributable, reproducible, and auditable before deployment decisions are made, while distinguishing contract conformance across platforms from hardware-dependent deployment costs.**

Link to graphical and video abstracts, and to code:
https://latamt.ieeer9.org/index.php/transactions/article/view/10820

***Index Terms*—Contract-Based Validation, Python–Firmware Validation, Reproducibility, TinyML.**

I. Introduction

TINY MACHINE LEARNING (TinyML) enables neural inference directly on microcontrollers under strict constraints on computation, memory, and energy [3], [23]. Survey and MCU-feasibility studies further show that deployment on these targets is attractive but tightly coupled to latency, memory, and power trade-offs [2], [4]. Its relevance spans embedded and low-power IoT applications that benefit from local, resource-aware inference with reduced communication dependence [9], [11].

Recent TinyML studies have emphasized quantization and deployment practices, as well as hardware-aware robustness concerns [7], [8]. In parallel, toolchains for benchmarking and on-device evaluation automation have improved runtime visibility and experimental reproducibility [6], [15], [18]. Yet these advances typically report offline results and device-side measurements as adjacent stages, not as two executions bound by the same explicit contract. As a result, preprocessing rules, tensor metadata, rolling-window semantics, and output decoding may remain implicit implementation assumptions instead of explicit, auditable comparison points. This is particularly critical in rolling-window time-series inference, where small deviations in alignment, numerical handling, or output reconstruction can distort on-device predictive metrics.

LiteML-Edge addresses this gap through a TinyML framework organized around an explicit validation contract that binds preprocessing, rolling-window policy, tensor decoding/mapping, post-processing, and Rolling-24 aggregation across offline and firmware executions. The workflow integrates training, optimization, quantization, artifact export, deployment, validation, and on-device cost reporting under the same contract while remaining model-aware when tensor I/O, output semantics, or post-processing rules differ. It is also compatible with deployment-aware extensions such as accelerator-in-the-loop experimentation and on-device semantic management [12], [20].

The contributions of this study are fourfold: (i) a TinyML framework organized around an explicit validation contract that integrates training, optimization, quantization, artifact export, deployment, validation, and on-device reporting, with traceable training and metric artifacts for cross-run traceability; (ii) a replay-driven 1:1 validation methodology in which Replay is the primary conformance mode and Field provides complementary operational evidence under the same contract; (iii) contract export and firmware execution governed by that contract, supporting fully quantized, fully floating-point, and hybrid tensor interfaces; and (iv) per-stage empirical evidence across immediate model I/O, final predictions, Rolling-24 metrics, and on-device cost measurements on two MCU architectures.

Sec. II situates LiteML-Edge relative to benchmarking and

The associate editor coordinating the review of this manuscript and approving it for publication was Javier Vázquez Castillo (*Corresponding author: Charles Santos*).

Charles Santos, I. G. Torné, C. M. S. Figueiredo, and F. de S. Cardoso are with the Postgraduate Program in Electrical Engineering, State University of Amazonas, Manaus, Brazil (e-mails: cpds.mee24@uea.edu.br, itorne@uea.edu.br, cfigueiredo@uea.edu.br, and fcardoso@uea.edu.br).

profiling toolchains. Sec. III, especially Sec. III-B, details the contract-centered architecture and artifact export mechanisms. Sec. IV presents stage-wise Replay validation, prediction and metric agreement, and secondary on-device cost evidence under Rolling-24. Sec. V discusses the main implications and limitations, and Sec. VI concludes the paper.

II. Related Work

Prior TinyML research can be grouped into three complementary directions: (i) embedded constraints and system trade-offs, (ii) benchmarking and evaluation toolchains, and (iii) deployment-time optimization strategies.

Embedded constraints and system trade-offs: Surveys and MCU-feasibility studies frame TinyML around microcontroller resource limits and recurring trade-offs among accuracy, latency, memory footprint, and energy consumption [2]–[4], [23]. Adaptive embedded inference work further highlights resource-aware model-selection decisions on constrained targets [10]. These works motivate measurement-aware deployment, but they generally stop short of defining an explicit offline-to-firmware *semantic equivalence protocol* (i.e., a verified rule set for preprocessing, windowing, tensor decoding, and aggregation).

Benchmarking and evaluation toolchains: A first subset of toolchains focuses on repeatable *profiling* and target retargeting, enabling standardized measurements across devices and backends [1], [5], [6]. These infrastructures improve comparability of performance results, but they mainly treat the model workflow as the object to benchmark, not as a contract to be checked stage by stage across offline and firmware execution.

A second subset emphasizes *on-device evaluation automation*, combining predictive metrics with device-side latency, energy, or operational monitoring [15], [18], [22]. Related evaluation studies also report feasibility limits or embedded performance observations under concrete system conditions [16], [17], [21]. This strengthens runtime reporting and experiment management, yet consistency of preprocessing, rolling semantics, and tensor decoding is usually assumed at deployment time, not enforced by an explicit conformance check.

Deployment-time optimization strategies: Another research direction targets efficiency and robustness through quantization practices and deployment reliability assessment [7], [8]. Related work also addresses hardware–model co-design and coordinated TinyML across interconnected devices [13], [14]. These contributions improve deployment flexibility and stability of embedded inference, but they are orthogonal to a contract-first methodology whose primary objective is to *make explicit and verify* preprocessing semantics, rolling-window policy, tensor decoding/mapping rules, post-processing order, and metric-update rules before interpreting accuracy or cost results.

Despite advances in benchmarking, profiling, and deployment optimization, the surveyed works generally do not make offline-to-firmware semantic equivalence explicit as a primary experimental requirement. This motivates a contract-first methodology in which equivalence is an explicit, auditable prerequisite, not an implicit assumption.

TABLE I
Validation Elements in TinyML Studies

Papers	Replay-driven	Offline ↔ firmware	Stage-wise	Exported artifacts	On-device reporting
[5]					✓
[6]					✓
[7]					✓
[8]					✓
[15]					✓
[16]					✓
[18]					✓
[22]					✓
This Paper	✓	✓	✓	✓	✓

Table I summarizes the reported validation elements across representative TinyML studies. A check mark indicates that the corresponding element is explicitly described in the cited paper, whereas a blank cell means that the element was not explicitly identified in the reported scope and should not be interpreted as a negative quality judgment. The matrix is intended as a descriptive scope comparison, not a score-based ranking. Within this conservative summary, LiteML-Edge is distinguished by formalizing offline-to-firmware validation as an explicit, contract-centered and replay-driven methodology.

III. Methodology

A. Framework Design and Contract-Bounded Iteration

LiteML-Edge is a contract-centered framework operationalized through a replay-driven pipeline that explicitly binds TensorFlow-side training, optimization, quantization, and artifact export to firmware-side deployment and validation through a shared experimental contract. The contract is defined at the artifact and policy level, not around any single code implementation, while the empirical validation reported here is limited to the TensorFlow reference/TFLM runtime pair on the two evaluated MCU architectures. The core requirement is that contract artifacts (schema, scalers, windowing, tensor metadata, and metric policy) remain explicit and stable and are consumed consistently by the validated runtime.

Fig. 1 summarizes the contract-centered workflow for deterministic 1:1 validation between Python reference and firmware execution. A Replay artifact is processed by a shared data pipeline, executed as a Python reference, exported through a contract-binding step, and then reproduced on-device for per-stage validation. The lower block represents the shared validation contract anchoring reproducibility across both environments.

The workflow is organized into six methodological blocks:

- **Replay Artifact**: a validated dataset is serialized in deterministic order so that the same input sequence can be consumed offline and on-device.
- **Shared Data Pipeline**: filtering, rolling-window construction ($W{=}24$), and scaling/normalization are defined once and treated as shared contract elements, not environment-specific implementations.

- **Python Reference Execution**: the shared pipeline, inference, and post-processing are executed offline in float32 as the reference path for conformance validation.
- **Contract Export**: compile-time artifacts are generated to bind Python and firmware execution, including exported headers, parameter binding, tensor metadata, and output-mapping rules.
- **Firmware Execution**: the embedded implementation reproduces the same contractual semantics under TensorFlow Lite Micro, including model-aware tensor handling and shared post-processing.
- **1:1 Validation**: equivalence is assessed across internal stages and aggregated outputs, including the preprocessed input, tensor payload, raw output, post-processed prediction (y^*), and Rolling-24 metrics.

For the experimental campaign reported here, the same validation workflow was used across three model families: a multilayer perceptron (MLP), a long short-term memory network (LSTM), and a compact one-dimensional convolutional model (Conv1D Tiny). LiteML-Edge nevertheless allows model-level contract instantiation whenever tensor I/O, output semantics, post-processing rules, or comparison checks must be specialized. In practice, model-aware checks specialize tensor-payload interpretation, raw-output decoding, and output mapping according to each model's I/O contract while preserving the same experiment-level workflow.

B. Artifact Export Contract and Reproducibility Checks

Traceable artifacts are generated and consumed directly by embedded firmware, preventing implicit or duplicated transformations between offline evaluations and on-device executions. For each experiment, preprocessing and post-processing definitions are exported as C/C++ headers, including feature ordering and tensor shapes, normalization or scaling parameters, rolling-window configuration (e.g., $W = 24$), and tensor I/O metadata (type and, when quantized, scale and zero-point). LiteML-Edge supports fully quantized, fully floating-point, and hybrid tensor configurations and treats tensor decoding as part of the execution contract by exporting the required metadata per I/O tensor and enforcing consistent decoding and output mapping. The same contract also links exported artifacts to traceable training and metric artifacts, supporting cross-run traceability. In the present framework, serializing these artifacts avoids runtime parsing overhead and enforces identical data handling across environments.

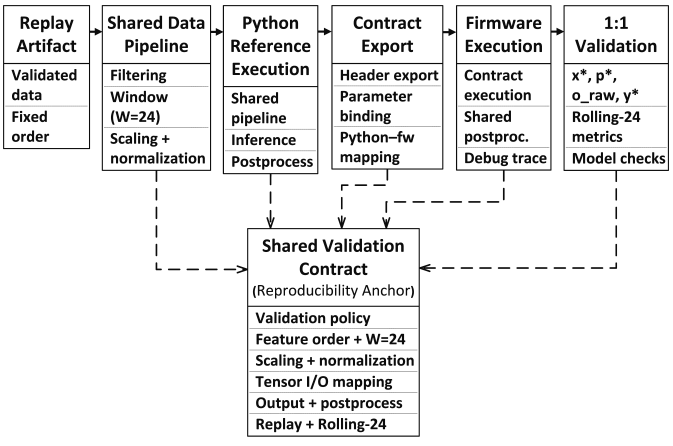


Fig. 1. LiteML-Edge workflow for replay-driven 1:1 Python–Firmware Validation.

Formal contract definition: LiteML-Edge makes the experimental semantics explicit through a contract $\mathcal{C}$ that must be satisfied by both the offline evaluator and the firmware runtime:

$$\mathcal{C} \triangleq (\mathcal{S},\ \mathcal{K},\ \mathcal{W},\ \mathcal{Q},\ \mathcal{P},\ \mathcal{M}), \tag{1}$$

where $\mathcal{S}$ is the feature schema (order, shapes, and time-feature policy), $\mathcal{K}$ is the scaler definition (forward/inverse parameters and application stage), $\mathcal{W}$ is the rolling policy (window size, stride, and warm-up exclusion), $\mathcal{Q}$ is the tensor I/O specification (type and, when quantized, scale/zero-point, with explicit output mapping for multi-output or concatenated outputs), $\mathcal{P}$ is the post-processing order (e.g., residual decoding and unit reconstruction), and $\mathcal{M}$ is the metric acceptance/aggregation policy (valid-event definition and update rules). Equivalence is assessed only for runs in which both sides satisfy $\mathcal{C}$; deviations are treated as contract violations, not model effects.

For per-stage validation, LiteML-Edge distinguishes four operational artifacts: x^*, the preprocessed input produced by the shared data pipeline; p^*, the effective tensor payload delivered to the model; o_{raw}, the immediate raw output obtained from the model/runtime path; and y^*, the post-processed prediction used directly for final reporting and Rolling-24 metric aggregation. This separation allows the framework to localize mismatches to the correct execution stage instead of collapsing all discrepancies into final metrics alone.

Contract invariants cover rolling semantics, feature schema and scaler parity, tensor decoding metadata (including output mapping), post-processing order, and metric policy. For regression models predicting residuals, the post-processing order is fixed by the exported contract (residual decoding and unit reconstruction), preventing order-of-operations divergence that would otherwise affect numerical stability and reported metrics.

Replay Mode is the deterministic conformance mechanism: offline scripts export a deterministic sample sequence, from which the firmware reconstructs the same windows in the same order. This isolates contract mismatches from variability introduced by live sensing in *Field Mode*. Numeric precision is treated as a controlled variable: offline evaluation uses float32 arithmetic to match firmware behavior, avoiding hidden float64 paths that can affect rolling aggregation. All offline arrays and metric accumulators are explicitly cast to float32. On-device, metric computation and post-processing are performed in float32; quantized tensors (when present) are decoded using their exported scale and zero-point.

Logging is standardized to enable direct comparison between offline and on-device results. Each inference record includes a window index, decoded predictions, optional ground truth (Replay Mode), and rolling metric updates. Under the explicit Rolling-24 policy, the nine final metric components comprise MAE, RMSE, and R^2, each computed at the aggregate level and separately for temperature and humidity. A *valid inference event* requires a full window ($W{=}24$), successful

TABLE II
CONV1D TINY CONTRACT INSTANTIATION

Element	Instantiated contract
$\mathcal{S}$	Before lag construction, H_{in} is processed by the contract-defined causal EMA. Fixed feature order: `T_out`, `H_out`, `T_in_lag1`, `H_in_lag1`, `T_out_lag1`, `H_out_lag1`, `T_in_lag2`, `H_in_lag2`, `sin_hour`, `cos_hour`, `weekday`, and `month`; identical order and lag semantics in Python and firmware.
$\mathcal{K}$	Training-fitted MinMax bounds for inputs and residual targets (`X_MIN`, `X_MAX`, `DY_MIN`, and `DY_MAX`); inputs are clamped to their training bounds, and all parameters remain fixed during validation and deployment.
$\mathcal{W}$	A 24-step sliding window with stride 1; inference starts only after warm-up, with the same ring-buffer update in Replay and Field.
$\mathcal{Q}$	INT8 input tensor $[1, 24, 12]$ in fixed chronological order, with scale and zero-point read from the input tensor; two float32 outputs for the scaled residuals ΔT_{in} and ΔH_{in}.
$\mathcal{P}$	Map outputs 0 and 1 to ΔT_{in} and ΔH_{in}, inverse-scale them, and add them to the corresponding lag-1 values to recover T_{in} and H_{in}.
$\mathcal{M}$	MAE, RMSE, and R^2 over valid post-warm-up predictions, with checks at the normalized input x^*, tensor payload p^*, raw output o_{raw}, reconstructed prediction y^*, and final metrics.

TABLE III
STAGE-WISE CONTRACT CONFORMANCE CHECKS

Check	Offline–On-Device Consistency Requirement
Replay artifact	Identical sample order, epoch alignment, and window indexing
Shared data pipeline	Same filtering, windowing, normalization/scaling, and warm-up policy
Critical input (x^*, p^*)	Contract-consistent preprocessed input and effective tensor payload under the exported schema
Immediate output (o_{raw})	Contract-consistent raw tensor bytes/metadata and decoded raw outputs under model-aware mapping
Post-processing (y^*)	Contract-consistent output mapping, inverse scaling, and reconstruction order
Predictions and metrics	Contract-consistent final predictions, valid-event rules, and Rolling-24 aggregation
Field operation	Same post-acquisition contract; treated as operational evidence, not primary proof of equivalence

invocation, and decoded outputs that satisfy the exported tensor contract, including the tensor type, quantization metadata when applicable, and output mapping. Metrics are accumulated only over valid post-warm-up inference events; in this study we report n=24 valid events.

Table II provides a worked instantiation of $\mathcal{C}$ for Conv1D Tiny. Dataset preparation precedes Replay generation and remains outside the inference contract; the validated x^* path begins with the transformations shared by Python and firmware.

To reduce common sources of error in TinyML deployment, explicit equivalence checks are defined. Table III summarizes the conformance checks to be applied to each model variant and firmware build before interpreting cost or accuracy metrics.

Two execution contexts are supported: *Replay* is the primary conformance mode because it provides artifact-driven, auditable alignment, whereas *Field* preserves the same post-acquisition contract as operational evidence.

IV. EXPERIMENTS AND RESULTS

A. Scenario, Deployment, and Rolling-Window Protocol

To exercise the contract mechanisms under controlled conditions, a deterministic time-series artifact derived from a tropical-climate indoor/outdoor temperature–humidity study [24] was adopted. Starting from synchronized Singapore temperature and relative-humidity records, the artifact was built by inner-joining both variables on timestamp, mapping non-suffixed columns to outdoor temperature and humidity, averaging the indoor sensors per timestamp to obtain T_{in} and H_{in}, dropping incomplete timestamps, and rounding the final variables $(T_{\text{out}}, T_{\text{in}}, H_{\text{out}}, H_{\text{in}})$ to two decimals for deterministic regeneration. This produced a four-variable reference series with 9,932 synchronized observations.

Because the original subset contains irregular sampling intervals, the resulting series is regularized hourly to satisfy Rolling-24 while preserving seasonality and bounded value ranges. Replay headers are then generated deterministically from that fixed artifact within the LiteML-Edge export flow. The artifact is used as a controlled validation artifact for framework behavior, not as a real-world predictive benchmark.

Heterogeneous reference models (MLP, LSTM, Conv1D) are used as validation vehicles to stress distinct tensor layouts and output mappings under a unified export and execution contract. All models are exported to TensorFlow Lite and integrated into matched TensorFlow Lite Micro projects for a WEMOS LOLIN32 (ESP-WROOM-32, Xtensa) and a NUCLEO-F411RE (STM32F411RE, ARM Cortex-M4), following embedded inference integration practices and quantization-related deployment considerations [7], [11]. Both builds share the contract headers, Replay sequence, tensor metadata, post-processing, logging, sensor roles, and event boundaries; only platform bindings differ. In Field Mode, two DHT22 (AM2302) temperature and humidity sensors are used to acquire the indoor and outdoor temperature and humidity variables, whereas Replay Mode reconstructs these variables from the exported header-based dataset artifact rather than from live sensing. LiteML-Edge exports the tensor type and, when quantized, the scale and zero-point for the corresponding input/output tensors. In this study, the MLP model operates with INT8 input and INT8 output, the LSTM model operates in float32 input/output, and the Conv1D model operates in a hybrid configuration with INT8 input and float32 outputs. Deterministic decoding and output mapping rules prevent mismatches arising from implicit quantization assumptions or runtime-side ambiguity [8].

Evaluation employs MAE, RMSE, and coefficient of determination (R^2) under a rolling-window regime with size $W = 24$ and unit stride; warm-up and incomplete-window events are excluded from aggregation in both Replay and Field. The choice $W = 24$ is treated as a fixed deployment-facing contract parameter, not as a tuned hyperparameter.

To keep the protocol auditable while enabling multi-model experimentation, device evaluation uses a time-compressed acquisition schedule in which each logged step corresponds to one hourly sample of the artifact (i.e., 1 min of runtime represents 1 h of the dataset timeline). This preserves the ordering and cardinality required by Rolling-24 without impractical 24-hour

wall-clock runs. The conformance horizon is fixed to $n = 24$ valid post-warm-up inference events; for stride-1 Rolling-24, this structurally requires $W + (n-1) = 47$ sequential samples.

In Replay Mode, these samples are exported deterministically as header-based artifacts, reconstructing time features from a fixed epoch reference. Firmware logs confirmed 1:1 window sequencing and stable platform-specific heap headroom, as summarized in Table VI.

In Field Mode, live sensing is enabled and time features are reconstructed from device time. Field preserves the same post-acquisition contract used in Replay, but is interpreted as operational evidence under live acquisition, not as the primary basis for deterministic equivalence claims.

TABLE IV
REPLAY STAGE-WISE CONFORMANCE ON BOTH TARGETS UNDER ROLLING-24 (n=24 FOR EACH MODEL–TARGET PAIR)

Model	Shared pipe.	Crit. input (x^*, p^*)	Raw dump	Dec. o_{raw}	Post. y^*	Pred. (T/H)	Metric checks
MLP	✓	✓	✓	✓	✓	T 24/24 H 24/24	8/9
LSTM	✓	✓	×	×	×	T 21/24 H 23/24	1/9
Conv1D Tiny	✓	✓	✓	✓	✓	T 24/24 H 24/24	9/9

✓ = MATCH; × = MISMATCH

B. Stage-Wise Conformance Evidence

Conformance evidence is produced under Replay and interpreted hierarchically rather than from aggregate metric deltas alone. LiteML-Edge first checks the shared data pipeline and critical input path, then the immediate output path, followed by the post-processed prediction path (y^*), and finally the Rolling-24 metrics. This ordering prevents downstream agreement from masking earlier contract violations. Table IV summarizes the resulting stage-wise conformance evidence under Rolling-24 ($n = 24$).

Stage-specific checks use fixed scopes and acceptance criteria. Replay indices, integer payloads, tensor metadata, and memory order require exact agreement. Normalized float32 inputs and decoded raw outputs are compared using an absolute tolerance of 2×10^{-6}, while post-processed predictions (y^*) use 2×10^{-5}. Rounded prediction agreement is assessed at two decimal places, and final Rolling-24 metrics at four decimal places. The float32 tolerances were empirically calibrated from the numerical spread observed in replay-consistent executions and then applied unchanged to all models. Since they are empirical engineering criteria rather than universal thresholds, they carry a risk of threshold tuning. Even so, relaxing both tolerances by a factor of 100 leaves the LSTM stage-wise mismatch classification unchanged: the maximum differences at the decoded-output and post-processing stages still exceed the corresponding relaxed limits by factors of approximately 56 and 11. Aggregate MAE and RMSE values provide supporting evidence but do not override failures detected at earlier validation stages.

MLP and Conv1D Tiny achieved agreement across the critical validation path, matching the shared data pipeline, immediate model input, raw tensor dump, decoded raw output, and post-processed prediction stage (y^*). Conv1D Tiny matched all nine final Rolling-24 metric components at four decimal places, while MLP matched eight, with only a rounding-level difference in R^2_T (0.7038 offline vs. 0.7039 on-device). On each target, LSTM preserved agreement at the critical input stage but showed its first reproducible divergence immediately after inference at the raw-output stage, with 3 of the 24 valid events above tolerance (12.5% of the evaluated Replay trace), a maximum decoded-raw difference of 0.01127, and a maximum post-processed prediction difference of 0.02277. For the same traces, its prediction agreement was 21 of 24 for temperature and 23 of 24 for humidity, and one of nine final Rolling-24 metric components matched at four decimal places. Because stage-wise outcomes were identical across targets, conformance is reported once, whereas deployment costs remain platform-specific.

As complementary visual evidence for the final-prediction stage, Fig. 2 compares ground truth and post-processed temperature predictions for offline execution and firmware Replay over the valid-event block (n=24). The close alignment to the identity line for the replay-consistent case provides an intuitive view of final-prediction agreement, while Table IV remains the primary stage-wise evidence.

C. Secondary Deployment Cost Evidence under Rolling-24

Deployment cost is reported as secondary evidence. Under Rolling-24, LiteML-Edge separates *inference-only execution cost*, *deployment feasibility*, and baseline electrical behavior. Table V reports mean per-inference energy and latency for each model–hardware pair in Replay and Field using the protocol's valid-event and invocation-boundary semantics. Table VI summarizes serialized model size, tensor arena, and minimum heap headroom, whereas Table VII reports IDLE current and power separately from inference-only estimates.

Baseline vs. inference-only reporting: Baseline current and power are computed from IDLE samples outside inference windows. Inference-only cost uses per-inference energy ΔE and latency t from fixed invocation-boundary snapshots. Replay shows lower variability because execution is artifact-driven. Field exhibits slightly higher minimum free-heap values (Table VI), without changing the contract for windowing, tensor decoding, or metric acceptance.

Instrumentation and conditions: All cost results use the Rolling-24 contract (W=24), n=24 valid post-warm-up events per model–hardware pair, matched run lengths in Replay and Field, and fixed clocks of 240 MHz for the WEMOS LOLIN32 and 100 MHz for the NUCLEO-F411RE. An external INA219 measures the complete-board 5 V supply rail on both targets. To align the functional boundary, Wi-Fi, Bluetooth, and OLED display output are disabled on the WEMOS LOLIN32 because these functions are not part of the evaluated pipeline and have no corresponding hardware or workload in the evaluated NUCLEO-F411RE configuration; sensing, inference, serial logging, and electrical measurement remain

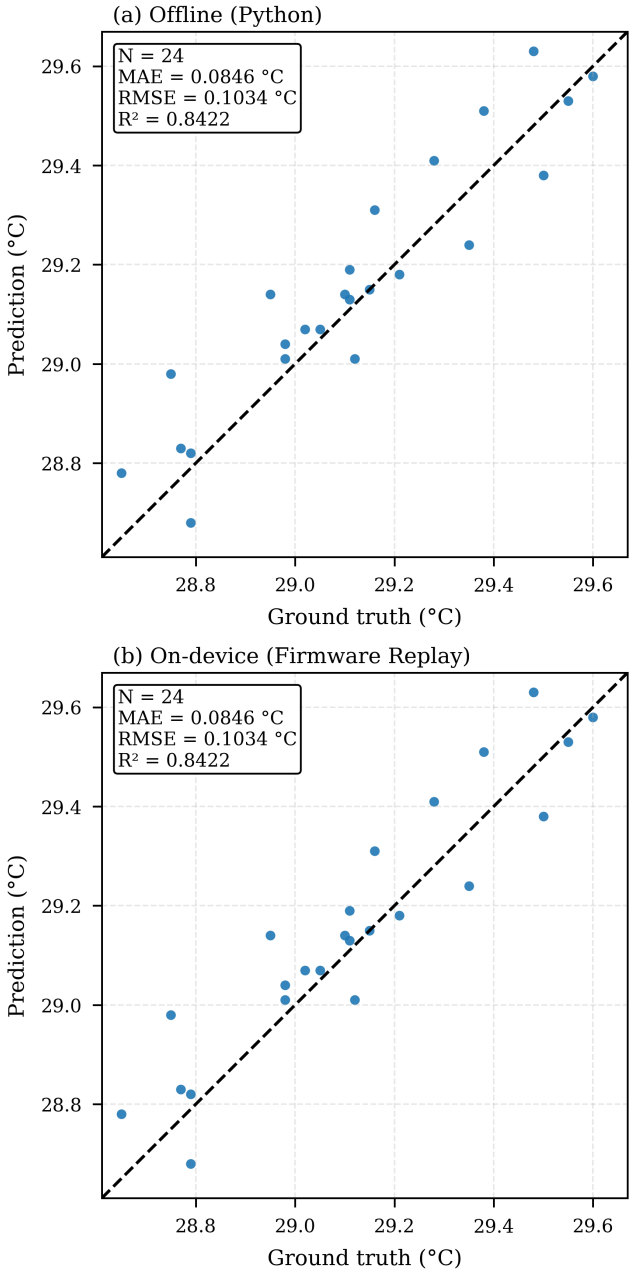


Fig. 2. Conv1D Tiny final-prediction agreement on both targets (n=24 per target); the two on-device prediction vectors overlap exactly.

active. This setup supports protocol-consistent comparisons under fixed boundaries rather than laboratory-grade absolute power metrology.

Within each platform, baseline current and power remain comparable across models, consistent with the sensing/control loop dominating model-specific effects. The lower STM32F411RE baseline does not imply uniformly lower inference energy, particularly for recurrent execution.

Energy and latency measurement: Per-inference energy is obtained from the firmware cumulative counter as $\Delta E = E_{\text{invoke-post}} - E_{\text{invoke-pre}}$. The event-pre and event-post readings are retained as supporting telemetry for the complete inference pipeline. Per-inference latency is measured over that invocation interval. Mean energy and latency are calculated over $n = 24$ valid post-warm-up events for each model–hardware pair and reported separately for Replay and Field. MLP latency is similar across targets, whereas LSTM shows the largest difference.

V. Discussion

LiteML-Edge complements existing TinyML toolchains by treating stage-wise protocol equivalence as a primary experimental concern. Whereas benchmarking platforms emphasize cross-target profiling and reproducible performance measurement [5], [6], and deployment-oriented works focus on quantization robustness and integration stability [7], [11], the proposed workflow makes the shared data pipeline, tensor decoding/mapping, post-processing, and rolling-window policy explicit and checks them *before* metric interpretation. As summarized in Table I, representative TinyML studies report selected deployment and on-device evaluation elements, whereas LiteML-Edge combines replay-driven validation, offline-to-firmware equivalence, per-stage checking, and exported contract artifacts within a single framework. The replay results also show why aggregate agreement alone is insufficient: MLP and Conv1D Tiny remain consistent across the full validation path, whereas LSTM preserves input-stage agreement but exhibits its first reproducible mismatch at the raw-output stage. In the present evidence, this supports strong localization of divergence without claiming closed causal attribution. Because the preprocessed features, tensor shape, element order, scaling metadata, and input payload remain conformant, the evidence narrows the likely fault domain to the recurrent execution path or the converted operator representation. Plausible mechanisms include differences in hidden- and cell-state initialization or update, gate ordering and parameter packing, fused or decomposed kernels, sequence layout, and implementation-dependent numerical accumulation or activation evaluation. Recurrent feedback may then propagate and amplify small numerical or semantic differences over successive steps. Distinguishing among these possibilities would require traces of gate activations, cell and hidden states, and intermediate accumulations.

A methodological consequence is the explicit separation between the baseline system cost and the inference-only cost. By fixing artifact definitions and measurement semantics, ambiguity in latency and energy reporting is reduced, although such deployment-cost evidence remains secondary to the paper's primary validation contribution. This practical

TABLE V
PER-INFERENCE ENERGY AND LATENCY ($n = 24$) UNDER ROLLING-24

Model	Hardware	Replay		Field	
		$\overline{\Delta E}$ (µW h)	$\bar{t}$ (ms)	$\overline{\Delta E}$ (µW h)	$\bar{t}$ (ms)
MLP	WEMOS LOLIN32	0.165	1.08	0.146	1.06
	NUCLEO-F411RE	0.181	1.03	0.195	1.04
LSTM	WEMOS LOLIN32	1.376	14.04	1.309	13.84
	NUCLEO-F411RE	1.910	25.46	2.134	25.48
Conv1D Tiny	WEMOS LOLIN32	0.605	5.89	0.573	5.72
	NUCLEO-F411RE	0.598	6.55	0.628	6.56

TABLE VI
MODEL SIZE, TENSOR ARENA, AND HEAP HEADROOM

Model	Hardware	Model size (kB)	Arena (kB)	Min. free heap (kB)	
				Replay	Field
MLP	WEMOS LOLIN32	6.46	0.91	238.7	239.0
	NUCLEO-F411RE	6.46	0.89	102.2	103.3
LSTM	WEMOS LOLIN32	9.46	3.42	238.7	239.3
	NUCLEO-F411RE	9.46	3.41	102.2	103.3
Conv1D Tiny	WEMOS LOLIN32	7.12	2.59	238.7	239.0
	NUCLEO-F411RE	7.12	2.56	102.2	103.3

TABLE VII
BASELINE IDLE CURRENT AND POWER

Model	Hardware	Replay		Field	
		$\mathbf{I}_{\mathrm{mean}}$ (mA)	$\mathbf{P}_{\mathrm{mean}}$ (mW)	$\mathbf{I}_{\mathrm{mean}}$ (mA)	$\mathbf{P}_{\mathrm{mean}}$ (mW)
MLP	WEMOS LOLIN32	61.83	316.14	60.62	310.75
	NUCLEO-F411RE	55.12	270.29	58.12	286.43
LSTM	WEMOS LOLIN32	61.88	316.69	60.14	308.49
	NUCLEO-F411RE	51.58	254.47	57.95	285.77
Conv1D Tiny	WEMOS LOLIN32	60.75	311.07	59.90	307.23
	NUCLEO-F411RE	55.52	272.07	57.78	285.07

relevance is strengthened by the fact that the framework is demonstrated with compact, optimized, and deployable models under real ESP32 and STM32F411RE on-device constraints. Cross-platform conformance despite differing latency, memory, and energy supports separating semantic portability from platform-dependent deployment cost. Replay-based equivalence verification therefore serves as an interpretive safeguard for Field observations.

A. Threats to Validity

Although LiteML-Edge is defined as an end-to-end offline validation framework operationalized through a replay-driven pipeline, the empirical validation presented here is limited to the TensorFlow/TFLM instantiation in one environmental sensing scenario, $W = 24$, and two MCU architectures. Broader validity across runtime ecosystems, window policies, or modalities requires dedicated evaluation. Changing W affects warm-up, window count, buffering, and smoothing; the contract structure may remain applicable to other modalities, but each modality would require specific sampling, preprocessing, tensors, tolerances, and metrics. Formal statistical comparison of machine-learning results typically depends on appropriate metrics and sufficient repeated evaluation values [19]; in contrast, the evaluated sequence forms a bounded diagnostic trace for execution equivalence under a fixed experimental contract. The 3/24 fraction (12.5%) exactly describes the evaluated trace, not a population rate or evidence of statistical sufficiency. Consecutive Rolling-24 events share input history and are not treated as independent for population inference; extending the Replay could alter both the observed event fraction and the final Rolling-24 values, and hence the number of the nine metric components that agree at four decimal places, without invalidating event-level comparisons or stage localization. The one-of-nine result is therefore a bounded component count for the evaluated trace, not a statistical estimate. Accordingly, the LSTM case is interpreted as localization of the earliest reproducible mismatch stage, not as definitive proof of a unique runtime mechanism.

Field Mode measurements remain subject to device-specific timing, sensing noise, synchronization behavior, and seeding progression. Replay is therefore the primary conformance mode for 1:1 equivalence claims, with Field retained as operational evidence. Energy comparisons are limited by complete-board 5 V rail sensing and firmware event boundaries; these choices support internally consistent comparison across runs but do not replace dedicated power metrology. Matched functional builds reduce platform-specific confounding, but the results do not establish a definitive ranking of the two hardware platforms.

B. Limitations and Future Directions

LiteML-Edge does not aim to standardize model architectures or establish performance benchmarks. Instead, it formalizes auditable evaluation contracts that maintain consistent data, preprocessing, tensor, inference, post-processing, and metric definitions across reference and deployed implementations. Future work will extend validation beyond $n = 24$ to additional datasets, input modalities, window sizes, runtimes, hardware platforms, and automated contract-verification pipelines. Broader adoption across toolchains may enable community-level validation of experimental contracts as explicit methodological artifacts.

VI. CONCLUSION

This paper presented LiteML-Edge as a contract-centered TinyML framework for replay-driven 1:1 validation between Python reference execution and firmware execution on resource-constrained microcontrollers. LiteML-Edge integrates offline training, optimization, quantization, artifact export, deployment, and per-stage validation so that agreement and mismatch become easier to trace and audit through replay artifacts, shared data pipeline rules, and execution governed by the exported contract.

Experiments on the WEMOS LOLIN32 and NUCLEO-F411RE yielded the same critical-path agreement for MLP and Conv1D Tiny, while localizing the earliest reproducible LSTM mismatch to the raw-output stage, not to input packaging. Replay supported conformance claims, while Field characterized operation under live sensing. Within the evaluated scope of one environmental sensing scenario, a fixed $W = 24$ policy, and the TensorFlow-to-TFLM deployment path, LiteML-Edge supported reproducible and auditable comparison of embedded TinyML validation outcomes for the fully quantized, fully floating-point, and hybrid tensor configurations represented

by the evaluated models, without extending these findings to other datasets, window sizes, or runtimes. Its contribution is therefore methodological: a validation framework centered on an explicit contract, demonstrated with compact, optimized, and realistically deployable TinyML models.

REFERENCES

[1] F. M. Aymone and D. P. Pau, "Benchmarking In-Sensor Machine Learning Computing: An Extension to the MLCommons-Tiny Suite," *Information*, vol. 15, no. 11, Art. no. 674, 2024, doi: 10.3390/info15110674.

[2] A. Elsts and R. McConville, "Are Microcontrollers Ready for Deep Learning-Based Human Activity Recognition?," *Electronics*, vol. 10, no. 21, Art. no. 2640, 2021, doi: 10.3390/electronics10212640.

[3] R. Immonen, T. Hämäläinen, and K. E. J. Leppänen, "Tiny Machine Learning for Resource-Constrained Microcontrollers," *Journal of Sensors*, vol. 2022, Art. no. 7437023, 2022, doi: 10.1155/2022/7437023.

[4] N. Schizas, A. Karras, C. Karras, and S. Sioutas, "TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review," *Future Internet*, vol. 14, no. 12, Art. no. 363, 2022, doi: 10.3390/fi14120363.

[5] P. van Kempen, R. Stahl, D. Mueller-Gritschneder, and U. Schlichtmann, "MLonMCU: TinyML Benchmarking with Fast Retargeting," in *Proc. 2023 Workshop on Compilers, Deployment, and Tooling for Edge AI (CODAI '23)*, 2023, doi: 10.1145/3615338.3618128.

[6] V.-E. Baciu, J. Stiens, and B. da Silva, "MLino bench: A Comprehensive Benchmarking Tool for Evaluating ML Models on Edge Devices," *Journal of Systems Architecture*, vol. 155, Art. no. 103262, 2024, doi: 10.1016/j.sysarc.2024.103262.

[7] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and Deployment of Deep Neural Networks on Microcontrollers," *Sensors*, vol. 21, no. 9, Art. no. 2984, 2021, doi: 10.3390/s21092984.

[8] M. de Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, and N. Pazos, "Robustifying the Deployment of tinyML Models for Autonomous Mini-Vehicles," *Sensors*, vol. 21, no. 4, Art. no. 1339, 2021, doi: 10.3390/s21041339.

[9] C. M. Bhushan *et al.*, "Deploying TinyML for Energy-Efficient Object Detection and Communication in Low-Power Edge AI Systems," *Scientific Reports*, vol. 15, Art. no. 44299, 2025, doi: 10.1038/s41598-025-27818-9.

[10] B. Taylor, V. Sanz Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive Deep Learning Model Selection on Embedded Systems," in *Proc. 19th ACM SIGPLAN/SIGBED Int. Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES '18)*, New York, NY, USA: ACM, 2018, pp. 31–43, doi: 10.1145/3211332.3211336.

[11] H.-Y. Lin, "Embedded Artificial Intelligence: Intelligence on Devices," *Computer*, vol. 56, no. 9, pp. 90–93, 2023, doi: 10.1109/MC.2023.3280397.

[12] H. Ren, D. Anicic, X. Li, and T. Runkler, "On-device Online Learning and Semantic Management of TinyML Systems," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 4, Art. no. 55, pp. 1–32, 2024, doi: 10.1145/3665278.

[13] M. Gulati, K. Zandberg, Z. Huang, G. Wunder, C. Adjih, and E. Baccelli, "TDMiL: Tiny Distributed Machine Learning for Microcontroller-Based Interconnected Devices," *IEEE Access*, vol. 12, pp. 167810–167826, 2024, doi: 10.1109/ACCESS.2024.3492921.

[14] E. Manor and S. Greenberg, "Custom Hardware Inference Accelerator for TensorFlow Lite for Microcontrollers," *IEEE Access*, vol. 10, pp. 73484–73493, 2022, doi: 10.1109/ACCESS.2022.3189776.

[15] Z. Huang, K. Zandberg, K. Schleiser, and E. Baccelli, "U-TOE: Universal TinyML On-Board Evaluation Toolkit for Low-Power IoT," in *Proc. 2023 12th IFIP/IEEE Int. Conf. on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, 2023, doi: 10.23919/PEMWN58813.2023.10304946.

[16] G. Delnevo, S. Mirri, C. Prandi, and P. Manzoni, "An Evaluation Methodology to Determine the Actual Limitations of a TinyML-Based Solution," *Internet of Things*, vol. 22, Art. no. 100729, 2023, doi: 10.1016/j.iot.2023.100729.

[17] M. Ersoy and U. Şansal, "Analyze Performance of Embedded Systems with Machine Learning Algorithms," in *Trends in Data Engineering Methods for Intelligent Systems*, J. Hemanth, T. Yigit, B. Patrut, and A. Angelopoulou, Eds., vol. 76, pp. 231–236, 2021, doi: 10.1007/978-3-030-79357-9_23.

[18] M. A. Hasanpour, M. Kirkegaard, and X. Fafoutis, "EdgeMark: An Automation and Benchmarking System for Embedded Artificial Intelligence Tools," *Journal of Systems Architecture*, vol. 167, Art. no. 103488, 2025, doi: 10.1016/j.sysarc.2025.103488.

[19] O. Rainio, J. Teuho, and R. Klén, "Evaluation Metrics and Statistical Tests for Machine Learning," *Scientific Reports*, vol. 14, Art. no. 6086, 2024, doi: 10.1038/s41598-024-56706-x.

[20] J. Kwon and D. Park, "MAIL: Micro-Accelerator-in-the-Loop Framework for MCU Integrated Accelerator Peripheral Fast Prototyping," *Applied Sciences*, vol. 15, no. 3, Art. no. 1056, 2025, doi: 10.3390/app15031056.

[21] P. Nandi, K. R. Anupama, A. Bajaj, S. Shukla, T. Musale, and S. Kachadiya, "Performance Evaluation of Machine Learning Algorithms on System on Chips in Wearables for Healthcare Monitoring," *Procedia Computer Science*, vol. 218, pp. 2755–2766, 2023, doi: 10.1016/j.procs.2023.01.247.

[22] Z. Huang, K. Zandberg, K. Schleiser, *et al.*, "RIOT-ML: Toolkit for Over-the-Air Secure Updates and Performance Evaluation of TinyML Models," *Annals of Telecommunications*, vol. 80, pp. 283–297, 2025, doi: 10.1007/s12243-024-01041-5.

[23] S. Heydari and Q. H. Mahmoud, "Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions," *Sensors*, vol. 25, no. 10, Art. no. 3191, 2025, doi: 10.3390/s25103191.

[24] J. Pan, J. Tang, M. Caniza, J.-M. Héraud, E. Koay, H. K. Lee, *et al.*, "Correlating Indoor and Outdoor Temperature and Humidity in a Sample of Buildings in Tropical Climates," *Indoor Air*, vol. 31, pp. 2281–2295, 2021, doi: 10.1111/ina.12876.



Charles Pereira dos Santos received his degrees in electronics technology from UTAM, Manaus, Brazil, and in technical electronics from FMM, Manaus, Brazil, as well as an MBA in project management from FUCAPI, Manaus, Brazil. He is currently pursuing the M.Sc. degree in electrical engineering at the State University of Amazonas (UEA), Manaus, Brazil. He has over 28 years of experience in embedded systems, industrial automation, IoT, and electronic systems architecture. His research interests include TinyML, reproducible machine learning pipelines, energy-aware embedded systems, and automation/control architectures.



Israel Gondres Torné (Senior Member, IEEE) received his B.S. and M.S. degrees in electrical engineering from the University of Camaguey, Cuba, in 1995 and 2008, respectively, and the Ph.D. degree in technical sciences in electrical engineering from the same institution in 2015. He is currently Adjunct Professor C and coordinator of the Professional Master's Degree in Electrical Engineering at the State University of Amazonas (UEA). His research interests include power systems, energy efficiency, embedded systems, and artificial intelligence.



Carlos Maurício Seródio Figueiredo is an Associate Professor at the School of Technology of the State University of Amazonas (EST/UEA). He received the B.S. degree in electrical engineering from the Federal University of Amazonas (UFAM) and the M.Sc. and Ph.D. degrees in computer science from the Federal University of Minas Gerais (UFMG). His research interests include intelligent and embedded systems.



Fábio de Sousa Cardoso received his B.Sc. degree in electrical engineering from the Federal University of Amazonas (UFAM), Manaus, Brazil, in 1995, the M.Sc. degree in biomedical engineering from the Federal University of Paraíba (UFPB), Brazil, in 2000, and the Ph.D. degree in mechanical engineering from the Federal University of Rio de Janeiro (UFRJ), Brazil, in 2015. He is currently an Associate Professor at the School of Technology of the State University of Amazonas (EST/UEA) and a researcher at the Instituto Inovathus de Tecnologia. His research interests include embedded systems, IoT, Industry 4.0, distributed robotics, multi-agent systems, and machine learning for cyber-physical and intelligent industrial environments.