

Benchmarking Is Not Verification: Mathematical, Computational, and Systems Foundations for Edge AI Qualification

Sebastián A. Cruz Romero¹

¹Capicú Technologies

The deployment of compressed machine learning models on heterogeneous edge hardware introduces a constellation of unresolved scientific problems spanning model compression theory, formal verification, runtime systems, empirical evaluation methodology, control theory, and software engineering. Despite significant advances in each of these domains individually, no unified framework exists to verify, validate, and characterize the behavior of compressed models and the systems that execute them at the edge – a capability we term a "qualification layer" for Edge AI. This survey examines six foundational research threads that converge on this problem: (1) the mathematical characterization of accuracy degradation under quantization, pruning, and knowledge distillation; (2) the state of neural network verification and certified robustness, including recent extensions to quantized network semantics; (3) empirical evidence for cross-platform behavioral divergence arising from runtime and hardware heterogeneity; (4) the science of benchmarking and metric design for resource-constrained inference; (5) control-theoretic frameworks for monitoring deployed models under distribution shift; and (6) the widening gap between cloud-centric MLOps tooling and the requirements of deployment-stage qualification at the edge. We synthesize these threads to argue that a qualification layer for Edge AI constitutes a novel and necessary research contribution, identify the key open mathematical and computational problems, and situate this research direction within the broader landscape of efficient and reliable machine learning systems.

Publication Date: May 1, 2026

Correspondence: scruzromero@capicu.ai

1 Introduction

Machine learning models increasingly execute not in data centers but at the network edge—on microcontrollers, mobile system-on-chips, field-programmable gate arrays, and neural processing units operating under severe constraints of power, memory, and latency. Getting models to the edge requires aggressive compression: quantization reduces precision from 32-bit floating-point to 8-bit or 4-bit integers, pruning removes redundant parameters, and knowledge distillation transfers capabilities from large teacher models to compact students. Each transformation alters the model's mathematical structure, its decision boundaries, and its robustness properties in ways that are only partially understood.

The problem compounds at the system level. *The same compressed model, executed on different inference runtimes and hardware backends, can produce numerically divergent outputs*—a phenomenon documented empirically across 75 distinct platforms by Schlögl et al.

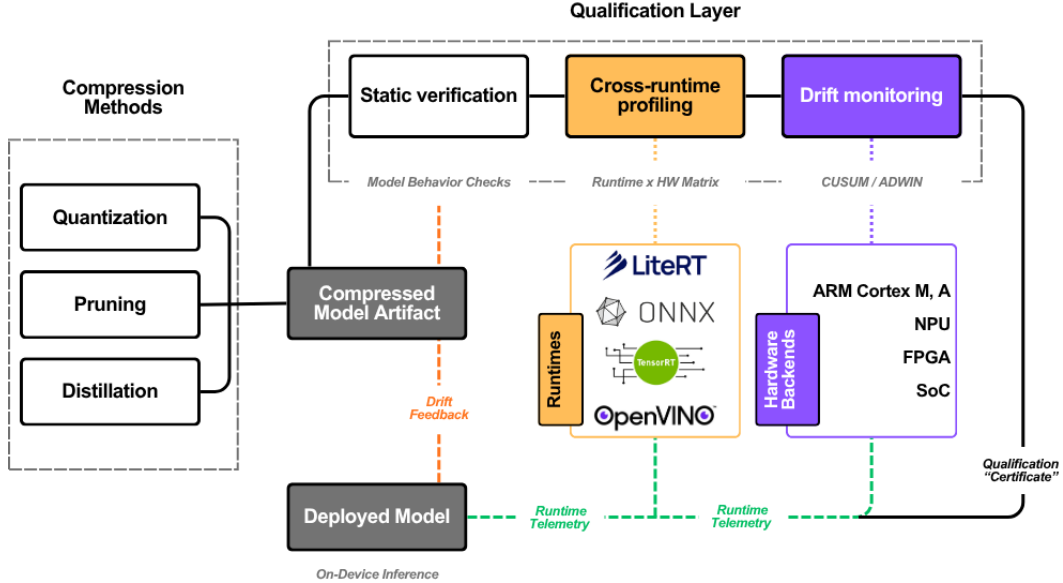


Figure 1: The qualification layer for Edge AI. A runtime-agnostic, device-agnostic system interposed between the compression pipeline and the heterogeneous deployment target space. The qualification layer performs three functions: static behavioral verification of the compressed model artifact, cross-runtime characterization across the runtime $\times$ hardware matrix, and continuous runtime monitoring via statistical drift detection.

[49]. Runtimes such as TensorFlow Lite, ONNX Runtime, TensorRT, and OpenVINO each implement operators differently, apply distinct kernel dispatch strategies, and exploit hardware-specific optimizations that silently alter inference results. Model format conversion between frameworks introduces additional semantic mismatches, with Davis et al. [11] finding that 33% of converter failures manifest as behavioral inconsistencies rather than crashes.

Yet no systematic, runtime-agnostic, device-agnostic methodology exists to qualify—that is, to *verify, validate, and characterize*—the behavior of a compressed model and its execution environment before and during edge deployment. The closest analogues are formal verification tools designed for real-valued networks, benchmarking suites like MLPerf that standardize performance measurement but not behavioral equivalence, and cloud-centric MLOps platforms that terminate their concern at the boundary of a centralized serving endpoint.

This survey argues that the convergence of six research threads—model compression theory, neural network verification, runtime heterogeneity, evaluation science, control-theoretic monitoring, and deployment-stage MLOps—creates both the scientific foundation and the urgent need for what we call a *qualification layer for Edge AI* (see Figure 1). We examine each thread in depth, identify open problems at their intersections, and argue that the qualification layer represents a distinct and tractable research contribution.

2 Model Compression Transforms Behavior in Mathematically Characterizable Ways

Model compression is the prerequisite for edge deployment, and its effects on model behavior constitute the first axis along which qualification must operate. Three families of techniques dominate: quantization, pruning, and knowledge distillation. Each introduces specific, mathematically describable distortions.

2.1 Quantization Theory and Error Characterization

Quantization maps continuous-valued weights and activations to a discrete set of representable values. The seminal framework of Jacob et al. [26] established the affine quantization mapping:

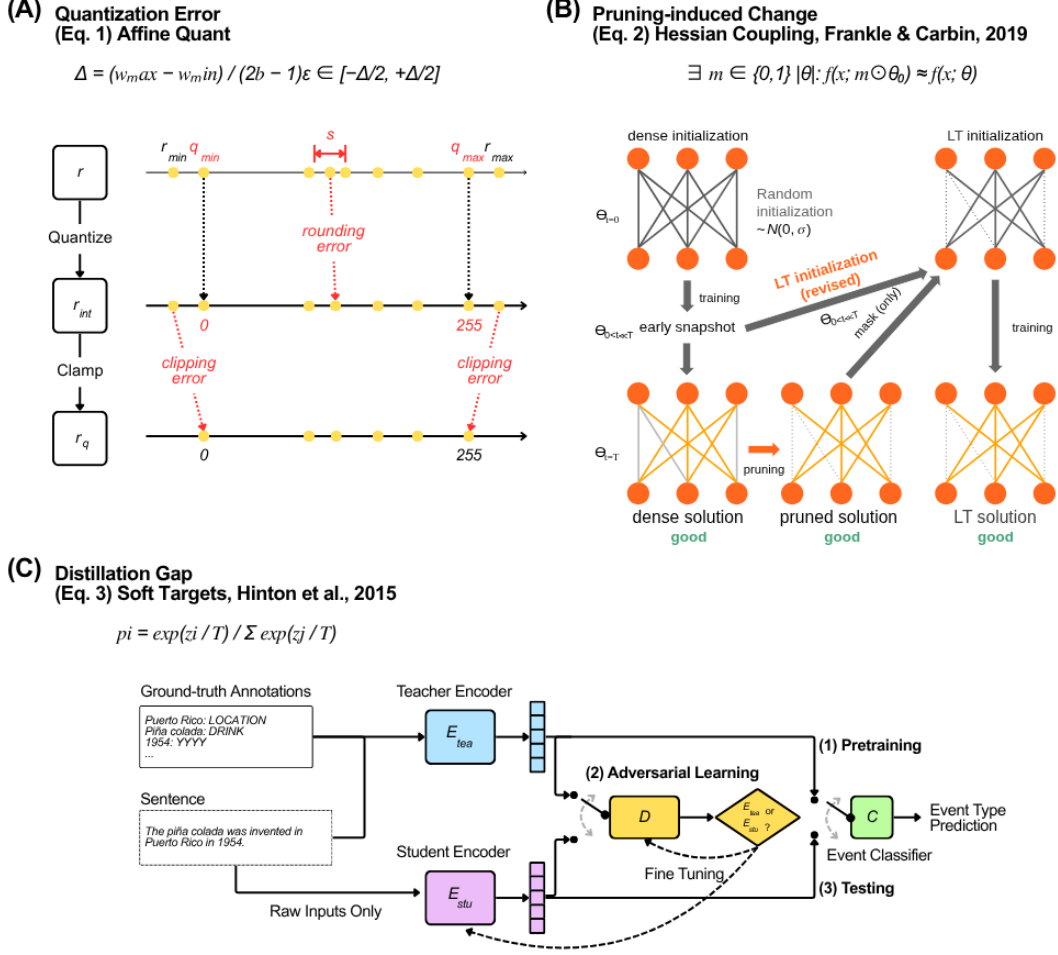


Figure 2: Compression-induced behavioral changes across three technique families. Each panel illustrates the primary error source introduced by one compression family, together with its governing equation. (A) *Quantization error* (affine mapping, Eq. 1): the real-valued weight range is clipped to a representable interval and then rounded to a discrete integer grid, introducing clipping error at the tails and rounding error $\varepsilon \in [-\Delta/2, +\Delta/2]$ throughout, where $\Delta = (w_{\max} - w_{\min}) / (2^b - 1)$ is the quantization step size. (B) *Pruning-induced change* (Hessian coupling, Eq. 2): the Lottery Ticket Hypothesis [15] posits the existence of a sparse mask $\mathbf{m}$ such that $f(\mathbf{x}; \mathbf{m} \odot \theta_0)$ matches full-network accuracy; the diagram contrasts dense initialisation, pruned solution, and lottery-ticket (LT) solution along the training trajectory. (C) *Distillation gap* (soft targets, Eq. 3): a student encoder is trained against a teacher encoder via temperature-scaled softmax outputs $p_i = \exp(z_i / T) / \sum_j \exp(z_j / T)$ [25]; the adversarial learning stage and event classifier illustrate the downstream task fidelity gap between teacher and student representations. A qualification layer must characterise all three error sources and their nonlinear interactions in multi-technique pipelines [e.g., 22].

$$Q(r) = S(q - Z), \quad (1)$$

where r is the real value, S is a scale factor, q is the quantized integer, and Z is the zero-point offset. This scheme enabled integer-only arithmetic inference on mobile processors, achieving 4× memory reduction with 8-bit quantization while maintaining acceptable accuracy on ImageNet through quantization-aware training (QAT).

The mathematical structure of quantization error has been progressively refined. For uniform b -bit quantization, the error $\varepsilon = Q(w) - w$ behaves as approximately uniform noise in $[-\Delta/2, \Delta/2]$ where $\Delta = (w_{\max} - w_{\min}) / (2^b - 1)$. The survey by Gholami et al. [18] connects this to rate-distortion theory: quantization is fundamentally a problem of optimal

discrete representation under distortion constraints. The choice of clipping range involves a tradeoff between *clipping error* (values outside the representable range) and *rounding error* (discretization within it), for which Banner et al. [3] derived analytical solutions under Gaussian and Laplacian assumptions on activation distributions, enabling practical 4-bit post-training quantization (PTQ) without retraining.

A critical insight for qualification emerged from Nagel et al. [39], who proved that the conventional rounding-to-nearest strategy is *provably suboptimal* for neural network quantization. Their AdaRound method frames the rounding problem via second-order Taylor approximation of the task loss:

$$\Delta\mathcal{L} \approx \frac{1}{2} \Delta\mathbf{w}^\top \mathbf{H} \Delta\mathbf{w}, \quad (2)$$

where $\mathbf{H}$ is the Hessian. Because off-diagonal Hessian entries couple individual rounding decisions, optimal rounding requires solving a layer-wise optimization problem rather than treating each weight independently. This mathematical insight—that quantization error is *non-separable* across parameters—has direct implications for qualification: verifying quantized model behavior cannot be reduced to verifying individual parameter perturbations.

For QAT, Esser et al. [13] proposed learning the quantizer step size via backpropagation through a scaled straight-through estimator (STE), achieving near-full-precision accuracy at 3-bit precision on ImageNet. The STE, however, lacks formal convergence guarantees—a theoretical gap that remains open and directly relevant to qualification, since QAT models carry implicit assumptions about gradient approximation quality that are not verified post-training.

Data-free approaches provide another qualification-relevant dimension. Nagel et al. [38] exploited the scale-equivariance of ReLU activations to perform cross-layer weight equalization and bias correction using only batch normalization statistics, requiring no calibration data—precisely the scenario where qualification tooling would operate on deployment pipelines with proprietary training data.

2.2 Pruning and the Structure of Compressible Subnetworks

Pruning removes parameters deemed redundant. Han et al. [21] established the foundational three-step pipeline of train–prune–retrain, achieving 13× compression of VGG-16 without accuracy loss. Li et al. [32] shifted to structured pruning—removing entire convolutional filters based on ℓ_1 -norm—which produces hardware-realizable speedups without requiring sparse computation libraries.

The Lottery Ticket Hypothesis of Frankle and Carlin [15] reframed the mathematical understanding of pruning by conjecturing that dense, randomly initialized networks contain sparse subnetworks (“winning tickets”) that, when trained in isolation from their original initialization, match the full network’s accuracy. Formally: for a network $f(\mathbf{x}; \boldsymbol{\theta}_0)$, there exists a mask $\mathbf{m} \in \{0, 1\}^{|\boldsymbol{\theta}|}$ such that $f(\mathbf{x}; \mathbf{m} \odot \boldsymbol{\theta}_0)$ achieves comparable test accuracy at 10–20% of original network size.

However, Liu et al. [33] challenged a key assumption by demonstrating that for structured pruning, training a pruned architecture from scratch with random initialization achieves comparable or better accuracy than fine-tuning inherited weights. This implies that *pruning’s value lies in architecture discovery rather than weight preservation*—a finding with direct qualification implications, since it suggests that the pruned architecture’s properties, not the specific weight values, determine post-compression behavior.

2.3 Knowledge Distillation and Its Theoretical Guarantees

Knowledge distillation [25] transfers information from a large teacher to a compact student by training the student to match temperature-scaled softmax outputs:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}, \quad (3)$$

where T is the temperature parameter. The soft targets carry “dark knowledge”—inter-class similarity structure encoded in the relative probabilities assigned to incorrect classes.

Phuong and Lampert [43] provided the first rigorous theoretical analysis, proving a generalization bound for distillation-trained linear classifiers that establishes fast convergence of expected risk. Their analysis identified three factors determining distillation success: data

geometry, optimization bias, and strong monotonicity (guaranteed risk decrease with training set size). For qualification, these results provide *formal conditions under which distilled models are provably well-behaved*—though the extension to deep nonlinear networks remains open.

The combination of compression techniques—exemplified by Han et al. [22], who achieved 35–49× compression through pruning, quantization, and Huffman coding—establishes the multi-technique pipeline paradigm. Each successive transformation modifies the model’s behavior in ways that interact nonlinearly, making end-to-end qualification of the compressed artifact essential.

3 Formal Verification and Certified Robustness Provide Partial but Insufficient Qualification Tools

If compression changes model behavior, verification asks whether the changed behavior still satisfies required properties. The neural network verification literature provides the mathematical machinery closest to what a qualification layer would require, but significant gaps remain.

3.1 The Verification Landscape for Real-Valued Networks

Katz et al. [29] introduced Reluplex, the first practical SMT-based verifier for deep networks with ReLU activations, extending the simplex method to handle piecewise-linear ReLU constraints. The underlying verification problem was proven NP-complete. The successor framework Marabou [30] extended this with support for multiple activation functions and integration with mixed-integer linear programming.

Abstract interpretation approaches pursued a complementary strategy: Gehr et al. [17] introduced AI2, using zonotope abstract domains to over-approximate reachable output sets, trading completeness for scalability. Singh et al. [53] developed DeepPoly, a more precise abstract domain combining floating-point polyhedra with intervals. Müller et al. [37] further advanced precision with PRIMA, a multi-neuron convex relaxation framework, verifying 20–34% more images than prior methods.

On the optimization side, Tjeng et al. [54] encoded piecewise-linear networks as mixed-integer programs with tight “big-M” formulations. Wang et al. [58] introduced symbolic interval propagation in ReluVal, subsequently improving with gradient-guided ReLU splitting in Neurify [57], achieving 5,000× speedup over Reluplex on ACAS Xu.

3.2 Certified Robustness as a Transferable Guarantee

Certified robustness methods provide formal guarantees that a classifier’s predictions are stable within specified input perturbation bounds. Cohen et al. [8] demonstrated that randomized smoothing—constructing a smoothed classifier $g(\mathbf{x}) = \arg \max_c \Pr(f(\mathbf{x} + \varepsilon) = c)$ where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ —yields a tight certified radius:

$$R = \frac{\sigma}{2} (\Phi^{-1}(p_A) - \Phi^{-1}(p_B)), \quad (4)$$

where p_A is the probability of the top class and Φ^{-1} is the inverse Gaussian CDF. This approach is *architecture-agnostic* and scales to full-resolution ImageNet, making it potentially applicable to certifying compressed models without modification.

Gowal et al. [20] showed that interval bound propagation (IBP), despite being a simple bounding technique, can produce large provably robust networks when combined with appropriate training schedules.

For qualification, Lipschitz analysis provides a particularly natural tool. Fazlyab et al. [14] developed LipSDP, computing guaranteed upper bounds on Lipschitz constants via semidefinite programming by interpreting activation functions as gradients of convex potentials satisfying incremental quadratic constraints. Virmaux and Scaman [56] proved that exact Lipschitz computation is NP-hard even for two-layer networks. If compression changes the network weights by $\delta \mathbf{W}$, and the output-to-weight Lipschitz constant L is bounded, then the output deviation satisfies:

$$\|f_{\text{original}}(\mathbf{x}) - f_{\text{compressed}}(\mathbf{x})\| \leq L \cdot \|\delta \mathbf{W}\|, \quad (5)$$

providing a formal behavioral equivalence guarantee within tolerance.

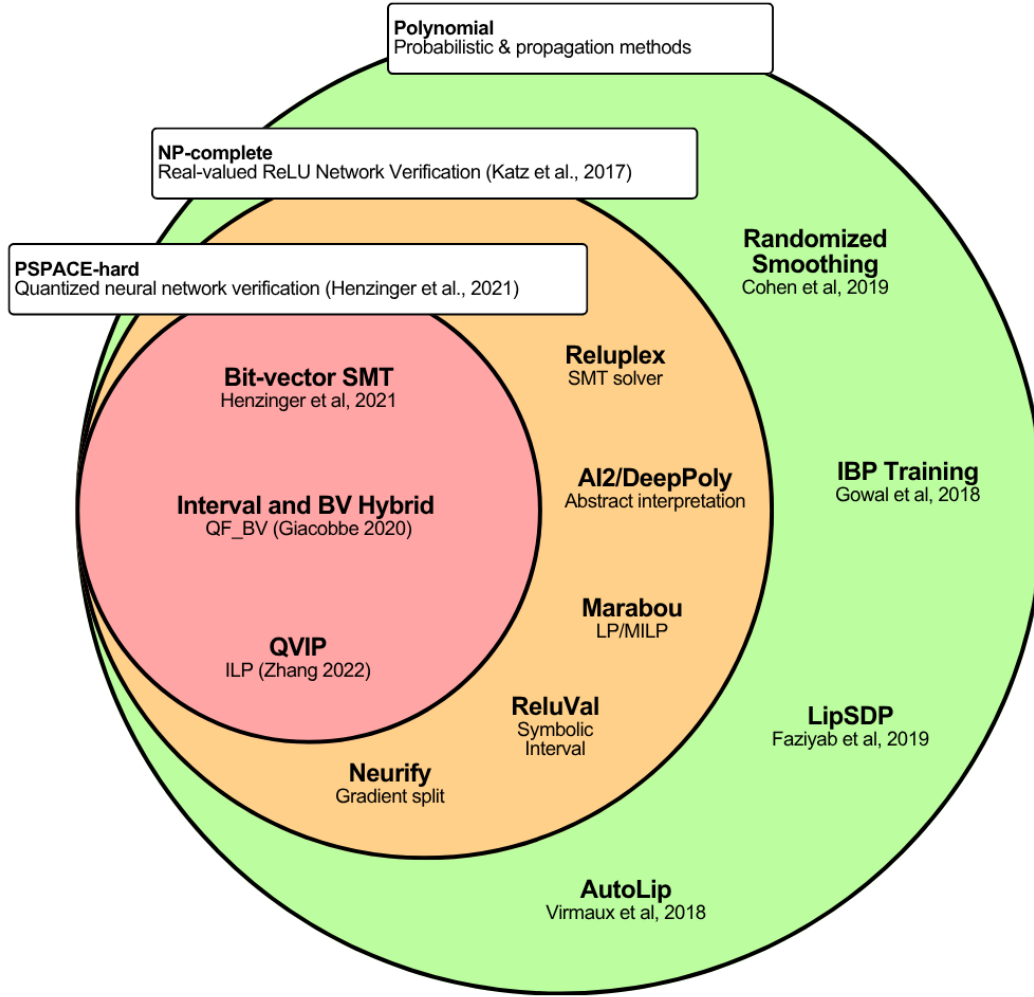


Figure 3: Computational complexity hierarchy for neural network verification. Verification of real-valued networks is NP-complete; verification of quantized networks is PSPACE-hard, strictly harder. Practical methods sacrifice either completeness (abstract interpretation, randomized smoothing) or scalability (exact SMT/MILP solvers). A qualification layer must navigate this landscape across the compressed, quantized models that actually execute at the edge.

3.3 The Verification Gap for Compressed Models

A critical finding for the qualification problem is that verification methods designed for real-valued networks are not sound for quantized networks. Giacobbe et al. [19] demonstrated this definitively, showing that neither robustness nor non-robustness is monotonic with the number of quantization bits. They introduced SMT solving over bit-vectors (QF_BV) to account for exact bit-precise quantized semantics.

Henzinger et al. [24] proved that quantized neural network verification is PSPACE-hard—a strictly harder complexity class than the NP-completeness of real-valued verification. Zhang et al. [60] advanced the field with QVIP, reducing quantized verification to integer linear programming and achieving two orders of magnitude speedup over SMT-based approaches. Jia and Rinard [27] further sharpened this concern by demonstrating that floating-point numerical error in verifiers themselves can be systematically exploited to find inputs violating claimed robustness guarantees—revealing that the gap between mathematical abstraction and physical computation is not merely theoretical.

4 Runtime and Hardware Heterogeneity Create a Behavioral Characterization Crisis

Even if a compressed model could be formally verified in isolation, the behavior observed at deployment depends critically on the runtime and hardware executing it.

4.1 Numerical Deviations Are Systematic and Consequential

Schlögl et al. [49] conducted the first systematic study across 75 distinct platforms, documenting that inference results are not consistent across platforms and are sometimes not even deterministic on the same platform (See Table 1). On CPUs, the primary cause is differences in SIMD instruction use (SSE vs. AVX vs. AVX-512); on GPUs, the dominant factor is runtime convolution algorithm selection (cuDNN auto-tuning). Deviations propagate through layers with high multiplication counts, amplifying through the network. “Boundary samples”—inputs near decision boundaries—can produce different classification results on different hardware, meaning that cross-platform numerical divergence can *flip predictions* for non-negligible regions of the input space.

4.2 Compilers Introduce Further Behavioral Variation

Chen et al. [7] introduced TVM, an end-to-end optimizing compiler providing performance portability across CPUs, GPUs, and FPGAs through operator fusion, hardware-specific scheduling, and learning-based cost models (AutoTVM). The fundamental insight of algorithm/schedule separation—inherited from Halide [44]—means that the same mathematical operation can have radically different physical implementations on different targets, with different numerical execution orders and thus different floating-point accumulation patterns. Rotem et al. [48] described Glow’s two-phase IR approach. Shen et al. [51] quantified compiler-induced divergence concretely in the first systematic study of deep learning compiler bugs, analyzing 603 bugs across TVM, Glow, and nGraph. They found that 21.44% of bugs occur in model loading alone, and that 29.63% of input-related bugs arise from ONNX format handling.

4.3 Model Conversion as a Source of Behavioral Mismatch

Davis et al. [11] surveyed 92 software engineers about deep learning interoperability and analyzed 200 failure issues in ONNX converters. Node conversion accounts for approximately 75% of defects, and 33% of failures produce semantically incorrect models—models that execute without error but produce behaviorally different outputs.

4.4 The Hardware Spectrum Amplifies Divergence

Jouppi et al. [28] documented the fundamental arithmetic differences between TPUs (8-bit integer matrix multiply), GPUs (mixed-precision floating-point), and CPUs (scalar/SIMD floating-point), with TPU achieving 15–30× speedup and 30–80× better TOPS/W over GPU/CPU. At the other extreme, David et al. [10] described TensorFlow Lite Micro operating in a few kilobytes of memory on Cortex-M microcontrollers with static memory allocation and platform-specific kernel optimizations (CMSIS-NN for ARM). The MLPerf Mobile benchmark [46] revealed that no single hardware unit dominates all ML tasks on mobile SoCs. This fragmentation means that behavioral characterization cannot target a single canonical execution environment.

5 The Science of Benchmarking Edge ML Remains Immature

Qualifying edge ML systems requires rigorous measurement methodology, but the benchmarking infrastructure for edge ML is still developing, and fundamental methodological questions remain open.

5.1 MLPerf Establishes a Foundation but Not a Qualification Framework

Mattson et al. [35] defined the MLPerf training benchmark with a time-to-quality metric and open/closed division structure. Reddi et al. [45] extended this to inference with four scenarios and rigorous statistical methodology using 99th-percentile latency with 99% confidence intervals; the first inference round received over 600 submissions spanning five orders of magnitude in performance.

(a) CPU platforms (64 platforms, TensorFlow 2.5.0, single-precision float32)

Vendor	Microarchitecture	Key SIMD	EQC count per model		
			Cifar10 small	Cifar10 R18	DeepWeeds R50v2
Intel	Sandy Bridge / Ivy Bridge	SSE4.2 (no AVX2)	1	1	1
Intel	Haswell / Broadwell	AVX2 (256-bit)	1	1	1
Intel	Coffee Lake	AVX2 (256-bit)	1	1	1
Intel	Skylake / Cascade Lake	AVX-512F (512-bit)	1	1	1–2
Intel	Ice Lake	AVX-512VBMI / IFMA	1	1	1
AMD	Rome / Milan	SSE4a, MisAlignSSE	1	1	1–2
<i>Total distinct EQCs across all 64 CPU platforms</i>			3	16	16

(b) GPU platforms (11 NVIDIA GPUs, same framework and models)

Generation	Device	Indeterm. [†]	EQC index (each GPU is unique)		
			Cifar10 small	Cifar10 R18	DeepWeeds R50v2
Kepler	K80	partial	1	1	1
Maxwell	GTX 970	partial	2	2	2
Maxwell	GTX 980	partial	3	3	3
Maxwell	M60	partial	3	3–4	4
Pascal	P100	all	4	4–5	5
Volta	V100	all	5	5–6	6
Turing	GTX 1650	none	6	6–7	7
Turing	RTX 2070	partial	7	7–8	8
Turing	T4 (AWS)	partial	7	8–9	9
Turing	T4 (GCP)	partial	7	8–9	9
Ampere	A100	partial	8	9–10	10
<i>Total distinct EQCs across all 11 GPU platforms</i>			8	10	10

(c) Aggregate EQCs across all 75 platforms (CPU + GPU combined)

Model	Architecture	Parameters	Max. total EQCs
Cifar10-small	Custom CNN (2 conv)	59,354	11
Cifar10-R18	ResNet-18	11,191,306	26
DeepWeeds-R50v2	ResNet-50v2	24,744,457	26



Table 1: Hardware-induced behavioral divergence in neural network inference (Schlög et al. 49). A single, fixed software stack — TensorFlow 2.5.0 running the same trained model on the same input — produces **up to 26 distinct output vectors** across 75 commodity hardware platforms. *Panel (a)* partitions 64 CPU platforms into equivalence classes (EQCs) of bit-identical inference outputs; the sole determinant is the SIMD instruction set enabled at runtime (SSE4.2, AVX2, AVX-512F, AMD-specific extensions). *Panel (b)* shows that every GPU generation forms its own EQC; 39 of 99 GPU inferences were indeterministic across repeated runs on the same hardware, with the P100 alone producing four distinct output vectors in 33 repeated runs due to runtime algorithm-selection noise. *Panel (c)* summarises the combined maximum EQC count. Crucially, these deviations arise from *hardware alone*; no runtime, compiler, or framework variation is present. A qualification layer must therefore characterise behavioral consistency not only across runtimes and frameworks, but across the underlying hardware execution substrate itself. [†]“Indeterministic” denotes platforms where repeated inference on identical inputs produced differing outputs within a single session, attributed to runtime convolution-algorithm selection under microarchitectural timing noise.

Banbury et al. [2] introduced MLPerf Tiny, the first benchmark for ultra-low-power TinyML systems operating at 10–250 MHz under 50 mW, measuring latency, energy, and accuracy simultaneously across four tasks using external power analyzers with precisely defined measurement windows. However, MLPerf measures *performance*—throughput, latency, energy—not *behavioral equivalence*. A qualification layer must go beyond performance benchmarking to systematic behavioral testing.

5.2 Methodological Fragilities Undermine Evaluation Claims

Dehghani et al. [12] introduced the concept of the “benchmark lottery,” demonstrating that relative algorithm rankings change significantly when different benchmark tasks are chosen. Bouthillier et al. [6] modeled the full benchmarking process and identified all sources of variation: data sampling, augmentation, parameter initialization, and hyperparameter choice. They showed that a biased estimator incorporating more sources of variation yields results closer to the ideal estimator at $51\times$ lower computational cost.

Henderson et al. [23] documented analogous problems in deep reinforcement learning, where different implementations of the same algorithm disagree on performance rankings—directly relevant to edge ML, where hardware and compiler variation compound software-level irreproducibility.

6 Control Theory Offers a Framework for Runtime Qualification

The control systems perspective recasts the qualification problem from a one-time verification task to a continuous monitoring challenge. We formalize this perspective as two interlocking algorithms. **Algorithm 1** (STATICQUALIFY) defines a pre-deployment qualification gate: a compressed model must pass an accuracy check, a Lipschitz bound test, and a cross-platform behavioral fingerprint before receiving a qualification certificate. **Algorithm 2** (RUNTIMEQUALIFY) defines the continuous deployment-stage loop: a CUSUM statistic and an ADWIN adaptive window jointly monitor inference loss, and a three-way controller (NOMINAL / WARN / REQUALIFY) determines whether corrective action is needed. The formal foundations for each component are developed in the subsections that follow.

6.1 Distribution Shift as the Fundamental Runtime Threat

Moreno-Torres et al. [36] unified scattered terminology by distinguishing covariate shift ($P_{\text{tr}}(\mathbf{X}) \neq P_{\text{test}}(\mathbf{X})$ with $P(Y|\mathbf{X})$ invariant), prior probability shift, and concept shift (change in $P(Y|\mathbf{X})$ itself). Shimodaira [52] provided the foundational correction mechanism via importance-weighted maximum likelihood and proved its asymptotic optimality. Lu et al. [34] established a systematic detection-understanding-adaptation framework, reviewing over 130 methods, categorizing drift as sudden, gradual, incremental, or recurring.

6.2 Statistical Process Control Maps Directly to Model Monitoring

The CUSUM control chart [40]:

$$S_n = \max(0, S_{n-1} + (x_n - \mu_0 - k)), \quad (6)$$

triggers an alarm when $S_n > h$. The ADWIN algorithm [5] adaptively adjusts its sliding window size based on observed rates of change, with rigorous false-positive and false-negative bounds derived from Hoeffding’s inequality. Gama et al. [16] introduced the Drift Detection Method (DDM), monitoring the error rate of online classifiers as a Bernoulli process.

These methods treat the model’s error rate, prediction confidence, or output distribution as a process signal—precisely the abstraction needed for a qualification layer that monitors edge model behavior against expected baselines.

6.3 Feedback Loops and Performative Effects

Sculley et al. [50] identified hidden feedback loops as a critical source of technical debt in ML systems. Perdomo et al. [42] introduced the framework of performative prediction, where model deployment itself changes the data distribution: $D(\theta)$ maps model parameters to induced distributions. They proved convergence conditions for repeated risk minimization—the ε -sensitivity condition:

$$\varepsilon < \frac{\gamma}{\beta}, \quad (7)$$

Algorithm 1: STATICQUALIFY($\hat{f}_\theta, \mathcal{D}_{\text{cal}}, \rho, \mathcal{H}, \tau_{\text{acc}}, \tau_L, \tau_\delta$) — pre-deployment qualification

Input : Compressed model $\hat{f}_\theta$;
calibration dataset $\mathcal{D}_{\text{cal}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$;
target runtime ρ and hardware backend $\mathcal{H}$;
tolerance thresholds $\tau_{\text{acc}}, \tau_L, \tau_\delta$
Output: qualification certificate $\mathcal{Q} \in \{\text{PASS}, \text{FAIL}\}$;
Lipschitz bound $\hat{L}$;
EQC fingerprint $\phi(\hat{f}_\theta, \rho, \mathcal{H})$

▷Step 1 - accuracy gate

1 $\hat{a} \leftarrow \frac{1}{N} \sum_{i=1}^N \mathbf{1}[\hat{f}_\theta(\mathbf{x}_i) = y_i]$

2 **if** $\hat{a} < \tau_{\text{acc}}$ **then return** (FAIL, $\emptyset, \emptyset$)

▷Step 2 - Lipschitz bound via LipSDP [14]

3 $\hat{L} \leftarrow \text{LIPSDP}(\hat{f}_\theta)$

4 **if** $\hat{L} > \tau_L$ **then return** (FAIL, $\hat{L}, \emptyset$)

▷Step 3 - cross-platform behavioral fingerprint

5 $\mathbf{o}^* \leftarrow \hat{f}_\theta(\mathbf{x}_{\text{ref}})$ ▷reference output on FP32 CPU

6 $\mathbf{o}^{(\rho, \mathcal{H})} \leftarrow \text{DEPLOY}(\hat{f}_\theta, \rho, \mathcal{H}, \mathbf{x}_{\text{ref}})$

7 $\delta \leftarrow \|\mathbf{o}^* - \mathbf{o}^{(\rho, \mathcal{H})}\|_\infty$

8 **if** $\delta > \tau_\delta$ **then return** (FAIL, $\hat{L}, \emptyset$)

▷Step 4 - equivalence class assignment [49]

9 $\phi \leftarrow \text{COMPUTE EQC}(\hat{f}_\theta, \rho, \mathcal{H}, \mathcal{D}_{\text{cal}})$

10 **return** (PASS, $\hat{L}, \phi$)

where γ is strong convexity and β is smoothness—providing formal convergence guarantees analogous to stability conditions in control theory.

7 MLOps Tooling Stops at the Cloud Boundary

The gap between existing MLOps infrastructure and the requirements of edge deployment qualification constitutes a practical motivation for the qualification layer research program.

7.1 Current Tooling Is Cloud- and Training-Centric

MLflow [59] provides experiment tracking, reproducible runs, and model packaging. ModelDB [55] introduced the first end-to-end model management system with branching history tracking. TFX [4] deployed a full pipeline at Google scale, identifying critical training-serving feature skew through automated data validation. Clipper [9], a prediction serving system with adaptive model selection and batching, requires Docker containers per model—infeasible for microcontrollers.

None of these platforms provides native support for edge device management, over-the-air updates, resource-constrained inference environments, or offline-first workflows.

7.2 The Training-Serving Gap Widens at the Edge

Amershi et al. [1] defined a nine-stage ML workflow based on studying Microsoft teams, but all nine stages are cloud-centric; edge deployment introduces additional stages—device qualification, model conversion and validation, on-device inference verification, and fleet-wide behavioral monitoring—that have no counterpart.

Paley et al. [41] confirmed that monitoring and updating are among the most challenging phases of ML deployment.

Algorithm 2: $\text{RUNTIMEQUALIFY}(\hat{f}_\theta, k, h, \delta_{\text{adwin}}, \tau_{\text{warn}}, \tau_{\text{req}})$ — continuous deployment-stage qualification loop

Input: Deployed model $\hat{f}_\theta$ with certificate $\mathcal{Q} = \text{PASS}$;
CUSUM sensitivity k , alarm threshold h ;
ADWIN confidence δ_{adwin} ;
qualification thresholds $\tau_{\text{warn}}, \tau_{\text{req}}$;
OTA update dispatcher ACTUATE

```

▷Initialise observer state
1  $S_0 \leftarrow 0$                                 ▷CUSUM statistic [40]
2  $\mu_0 \leftarrow \mathbb{E}_{\mathcal{D}_{\text{cal}}}[\ell(\hat{f}_\theta(\mathbf{x}), y)]$         ▷baseline loss
3  $W \leftarrow \mathcal{D}_{\text{cal}}$                                 ▷ADWIN adaptive window [5]
4  $t \leftarrow 0$ 

5 loop
    ▷Observe new inference event
6    $(\mathbf{x}_t, \hat{y}_t, \ell_t) \leftarrow \text{OBSERVE}()$ 
7    $t \leftarrow t + 1$ 

    ▷CUSUM update - sequential change detection
8    $S_t \leftarrow \max(0, S_{t-1} + (\ell_t - \mu_0 - k))$ 

    ▷ADWIN window update - adaptive drift detection
9    $W \leftarrow \text{ADWINUPDATE}(W, \ell_t, \delta_{\text{adwin}})$ 
10   $\hat{\mu}_t \leftarrow \text{MEAN}(W)$ 

    ▷Controller: threshold decision
11  if  $S_t > h$  or  $|\hat{\mu}_t - \mu_0| > \tau_{\text{req}}$  then
    |   ▷REQUALIFY - trigger full static re-qualification
    |    $\text{ACTUATE}(\text{REQUALIFY}, \hat{f}_\theta)$ 
    |    $S_t \leftarrow 0; \mu_0 \leftarrow \hat{\mu}_t; W \leftarrow \emptyset$ 
    |   ▷reset observer on corrective action
14  else if  $S_t > \tau_{\text{warn}}$  or  $|\hat{\mu}_t - \mu_0| > \tau_{\text{warn}}$  then
    |   ▷WARN - flag for human review, continue serving
    |    $\text{ACTUATE}(\text{WARN}, S_t, \hat{\mu}_t)$ 
16  else ▷NOMINAL - no action required
    |    $\text{ACTUATE}(\text{NOMINAL})$ 
17  end
18 end

```

7.3 Behavioral Testing Offers a Deployment-Stage Methodology

Ribeiro et al. [47] introduced CheckList, a task-agnostic behavioral testing methodology. In user studies, practitioners with CheckList created $2\times$ more tests and found $3\times$ more bugs in commercial models. This approach is directly applicable to edge deployment qualification—testing whether models satisfy behavioral requirements before and after deployment.

Lavin et al. [31] proposed Machine Learning Technology Readiness Levels (MLTRL), adapting NASA’s spacecraft engineering methodology to ML systems, defining 10 readiness levels with “switchback mechanisms” allowing intentional regression to earlier stages. However, *no specific edge deployment guidance or tooling integration currently exists.*

8 The Qualification Layer as a Convergent Research Contribution

The six research threads surveyed above converge on a single gap: no existing tool, framework, or methodology provides systematic, runtime-agnostic, device-agnostic qualification of compressed ML models deployed at the edge.

8.1 The Gap at the Intersection of Compression and Verification

Compression changes model behavior. Verification can characterize behavior. But verification methods for real-valued networks are not sound for quantized networks [19], and quantized verification is PSPACE-hard [24]. The Lipschitz analysis framework [14, 56] offers

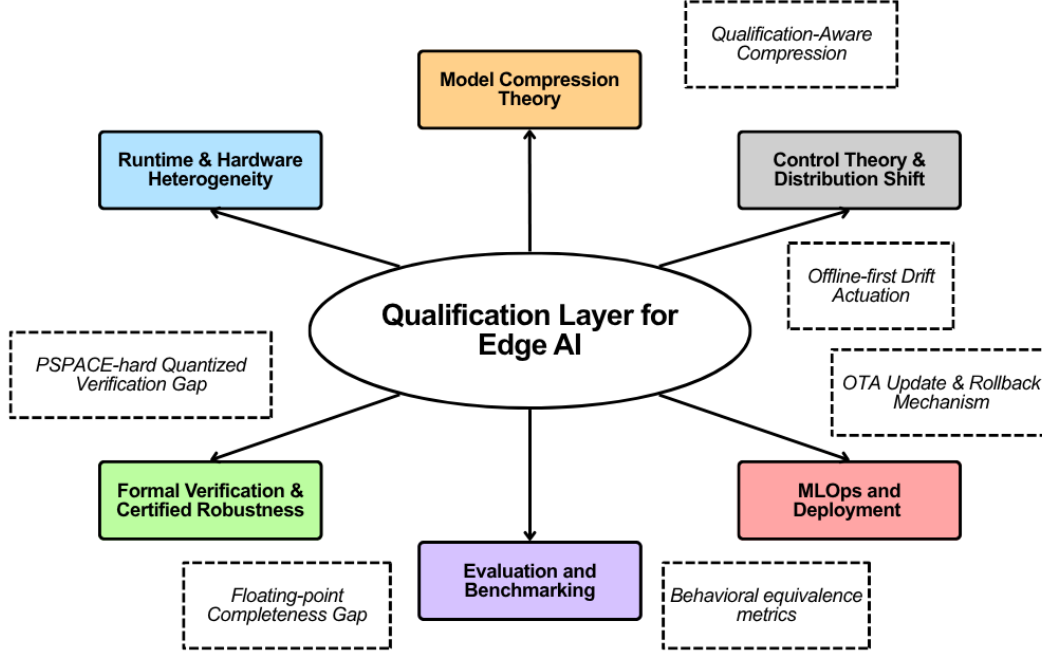


Figure 4: Convergence of six research threads on the qualification layer for Edge AI. Each thread contributes necessary but insufficient foundations. The qualification layer is the first research artifact targeting their intersection in a unified, runtime-agnostic, device-agnostic framework.

a promising bridge: bounding the change in Lipschitz constant under compression provides a global bound on output deviation via Equation (5), enabling transferable behavioral guarantees. Computing these bounds efficiently for the types of architectures and compression methods used in practice is an open mathematical problem.

Open Problem 8.1 Efficient behavioral equivalence testing for compressed models

Develop methods that are neither complete formal verification (intractable at scale) nor simple accuracy comparison (insufficient for safety-relevant properties), but occupy a principled middle ground with known coverage guarantees for the class of quantized, pruned, and distilled models used at the edge.

8.2 Cross-Runtime Characterization as a Missing Science

A qualification layer needs what might be called a *cross-runtime behavioral fingerprinting* capability: given a model, a set of target runtimes, and a set of hardware backends, characterize the expected output divergence across the matrix, identify inputs likely to produce divergent classifications, and provide guarantees about the fraction of the input space where behavior is consistent.

Open Problem 8.2 Cross-runtime divergence characterization

Build a mathematical framework for predicting and bounding output divergence across the runtime $\times$ hardware $\times$ precision matrix, integrating compiler analysis [7] with numerical deviation characterization [49] and operator-level failure analysis [11].

8.3 From Benchmarking to Qualification Metrics

MLPerf standardizes performance measurement; a qualification layer must standardize behavioral characterization. This means extending benchmarking methodology from scalar metrics to distributional and relational properties: output distribution similarity across

platforms, robustness certificate preservation under compression, decision boundary stability under numerical perturbation, and temporal behavioral consistency under deployment conditions.

Open Problem 8.3 Unified qualification metrics

Define a metric space over the behavioral properties of edge ML systems that integrates accuracy, robustness, cross-platform consistency, temporal stability, and resource consumption into a coherent qualification score, building on the statistical rigor framework of Bouthillier et al. [6].

8.4 A Control-Theoretic Architecture for Continuous Qualification

The static view of qualification—verify once before deployment—is insufficient because distribution shift [34] and feedback effects [42] cause deployed model behavior to evolve. The control-theoretic perspective provides the architecture that consists of: an *observer* monitoring behavioral metrics at the edge, a *controller* deciding when behavior has degraded below qualification thresholds, and an *actuator* triggering intervention.

Open Problem 8.4 Lightweight sequential monitoring for edge devices

Adapt sequential drift detection algorithms (CUSUM [40], ADWIN [5]) to operate within the memory and compute budgets of microcontrollers, potentially using compressed sufficient statistics and energy-proportional sampling strategies.

8.5 Qualification-Aware Compression

Open Problem 8.5 Qualification-aware compression

Design compression pipelines that optimize not only for accuracy-efficiency tradeoffs but for *qualifiability*—producing models whose behavioral properties are easier to verify and monitor post-deployment, potentially by co-optimizing Lipschitz constants and quantization schedules jointly.

9 Conclusion

This survey has traced six research threads that independently address fragments of a single underlying problem: how to systematically qualify compressed ML models for edge deployment. The evidence is compelling. Compression provably changes model behavior in mathematically complex ways [19, 39]. Runtime and hardware heterogeneity introduces systematic, consequential numerical divergence [49]. Existing verification methods break down precisely where they are most needed—on the quantized, pruned models that actually execute at the edge [24]. Benchmarking methodology is developing but does not address behavioral equivalence [2]. MLOps infrastructure terminates at the cloud boundary [4, 59]. And the models themselves are nonstationary systems whose behavior drifts under distribution shift [34] and feedback effects [42].

No existing tool spans these concerns. The qualification layer for Edge AI—a runtime-agnostic, device-agnostic system for verifying, validating, and continuously characterizing compressed model behavior across heterogeneous deployment targets—represents a distinct research contribution at the intersection of these six threads. The mathematical foundations exist in pieces: quantization error theory, Lipschitz analysis, certified robustness, sequential change detection, and performative stability. The empirical motivations are documented in rigorous peer-reviewed studies. What remains is the synthesis—the unified mathematical framework, the computational architecture, and the practical open-source tooling that transforms these fragments into a coherent qualification methodology for the billions of ML models that will execute at the edge of the network.

References

- [1] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, 2019.
- [2] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau, et al. MLPerf tiny benchmark. In *Proceedings of Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.
- [3] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.
- [4] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. TFX: A TensorFlow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1387–1395, 2017.
- [5] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining (SDM)*, pages 443–448, 2007.
- [6] X. Bouthillier, P. Delaunay, M. Bronzi, A. Trofimov, B. Nichyporuk, J. Szeto, N. Mohammadi Sepahvand, E. Raff, K. Madan, M. Volkovs, et al. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 3, pages 747–769, 2021.
- [7] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 578–594, 2018.
- [8] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. Certified adversarial robustness via randomized smoothing. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320, 2019.
- [9] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 613–627, 2017.
- [10] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, et al. TensorFlow Lite Micro: Embedded machine learning for TinyML systems. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 3, pages 800–811, 2021.
- [11] C. Davis, N. Humbatova, G. Bavota, P. Tonella, and M. Bianchini. Interoperability in deep learning: A user survey and failure analysis of ONNX model converters. *arXiv preprint arXiv:2303.17708*, 2024.
- [12] M. Dehghani, Y. Tay, A. A. Gritsenko, Z. Zhao, N. Houlsby, F. Diaz, D. Metzler, and O. Vinyals. The benchmark lottery. *arXiv preprint arXiv:2107.07002*, 2021.
- [13] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha. Learned step size quantization. In *International Conference on Learning Representations (ICLR)*, 2020.
- [14] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas. Efficient and accurate estimation of Lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

- [15] J. Frankle and M. Carlin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.
- [16] J. a. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295. Springer, 2004.
- [17] T. Gehr, M. Mirman, D. Drashkevich, P. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.
- [18] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. *Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence*, pages 291–326, 2022. arXiv:2103.13630.
- [19] M. Giacobbe, T. A. Henzinger, and M. Lechner. How many bits does it take to quantize your neural network? In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 12079 of *Lecture Notes in Computer Science*, pages 79–97. Springer, 2020.
- [20] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. 2018.
- [21] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, 2015.
- [22] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *International Conference on Learning Representations (ICLR)*, 2016. arXiv:1510.00149.
- [23] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [24] T. A. Henzinger, M. Lechner, and D. Žikelić. Scalable verification of quantized neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3787–3795, 2021.
- [25] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [26] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, 2018.
- [27] K. Jia and M. Rinard. Exploiting verified neural networks via floating point numerical error. In *Static Analysis: 28th International Symposium (SAS)*, pages 235–255. Springer, 2021.
- [28] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017.
- [29] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, pages 97–117. Springer, 2017.

- [30] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. Barrett. The Marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, pages 443–452. Springer, 2019.
- [31] A. Lavin, C. M. Gilligan-Lee, A. Prolu, P. Navarro, I. Sharma, T. Tiwary, M. Ghassemi, and B. Woodall. Technology readiness levels for machine learning systems. *Nature Communications*, 13(1):6039, 2022.
- [32] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient ConvNets. In *International Conference on Learning Representations (ICLR)*, 2017.
- [33] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR)*, 2019.
- [34] J. Lu, A. Liu, F. Dong, F. Gu, J. a. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.
- [35] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, et al. MLPerf training benchmark. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 2, pages 336–349, 2020.
- [36] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, 2012.
- [37] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev. PRIMA: General and precise neural network certification via scalable convex hull approximations. In *Proceedings of the ACM on Programming Languages (POPL)*, volume 6, pages 1–33, 2022.
- [38] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1325–1334, 2019.
- [39] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort. Up or down? Adaptive rounding for post-training quantization. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 7197–7206, 2020.
- [40] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.
- [41] A. Paleyes, R.-G. Urma, and N. D. Lawrence. Challenges in deploying machine learning: A survey of case studies. *ACM Computing Surveys*, 55(6):1–29, 2022.
- [42] J. C. Perdomo, T. Zrnic, C. Mendler-Dünner, and M. Hardt. Performative prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 7599–7609, 2020.
- [43] M. Phuong and C. H. Lampert. Towards understanding knowledge distillation. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 5142–5151, 2019.
- [44] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 519–530, 2013.
- [45] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, et al. MLPerf inference benchmark. In *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459, 2020.

- [46] V. J. Reddi, D. Kanter, P. Mattson, J. Duke, T. Nguyen, R. Chukka, K. Shiring, K.-S. Tan, M. Charlebois, W. Chou, et al. MLPerf mobile inference benchmark. In *Proceedings of Machine Learning and Systems (MLSys)*, 2022.
- [47] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4902–4912, 2020.
- [48] N. Rotem, J. Fix, S. Abdulrasool, G. Deng, R. Dzhabarov, J. Hegeman, R. Hewitt, N. Kouvelas, B. Ohabama, A. Tulloch, et al. Glow: Graph lowering compiler techniques for neural networks. In *arXiv preprint arXiv:1805.00907*, 2018.
- [49] B. Schlögl, M. Hofer, and A. Böhme. Causes and effects of unanticipated numerical deviations in neural network inference frameworks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.
- [50] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, 2015.
- [51] Q. Shen, S. Ma, J. Chen, Y. Liu, S.-C. Cheung, and C.-Z. Xu. A comprehensive study of deep learning compiler bugs. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 968–980, 2021.
- [52] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.
- [53] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. In *Proceedings of the ACM on Programming Languages (POPL)*, volume 3, pages 41:1–41:30, 2019.
- [54] V. Tjeng, K. Y. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.
- [55] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Huber, O. Lhotak, M. Zaharia, and S. Madden. ModelDB: A system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA)*, 2016.
- [56] A. Virmaux and K. Scaman. Lipschitz regularity of deep neural networks: Analysis and efficient estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.
- [57] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.
- [58] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium*, pages 1599–1614, 2018.
- [59] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the machine learning lifecycle with MLflow. volume 41, pages 39–45, 2018.
- [60] Y. Zhang, Z. Zhao, G. Chen, F. Song, and T. Chen. QVIP: An ILP-based formal verification approach for quantized neural networks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022.