ENABOL: Enabling Stable On-Chip Learning via Adaptive Lipschitz Budgeting in Edge-AI Systems

Anonymous Author(s)

Abstract

From households to scientific environments, edge-AI systems are increasingly present in the modern computing paradigm. These systems are often subject to performance degradation caused by distribution changes, sensor drift, and persistent hardware faults. Existing approaches lack support for stable on-chip backpropagation under fixed-point arithmetic. To bridge this gap, we present **ENABOL**, an extension to the widely adopted high-level synthesis toolchain `hls4ml` to design stable *fixed-point backpropagation*. By implementing a per-layer operator-norm projection constrained to a global Lipschitz budget, ENABOL enforces stability by provably bounding weights and gradients throughout training. ENABOL fully automates gradient computation, weight updates, and batch-level control, substantially lowering the barrier for on-chip learning and hardware–algorithm co-design. We evaluate ENABOL through a practical deployment workflow: models are pretrained in software, deployed to hardware, and then fine-tuned on-chip to recover performance under post-deployment shifts and faults. Our experiments span benchmarks evaluated on both MLPs and CNNs, demonstrating that ENABOL achieves resilient edge adaptation under deterministic fixed-point operation, establishing a stable foundation for autonomous on-device learning.

Keywords

Hardware accelerators, HLS, backpropagation, Lipschitz budget

1 Introduction

While on-device inference is now ubiquitous for edge-AI systems, ranging from mobile devices to scientific platforms, it remains governed by real-time latency, power, and bandwidth constraints [12, 28, 51, 53, 62]. Conversely, on-device training, particularly using fixed-point arithmetic, remains significantly less explored. Most deployed models continue to operate with frozen weights or only a minimal set of fine-tuning parameters [22, 25, 32, 48].

The advent of autonomous on-edge learning would unlock seamless continuous adaptation [24, 46, 61]. This capability is essential for real-time personalization, drift compensation in long-term deployments, and maintaining robustness in dynamic environments. However, hardware-level implementation remains challenging, as fixed-point training must reconcile the inherent low-precision instability with resource limitations of edge devices. The main numerical failure mode is unbounded gain. Fixed-point datapaths can amplify signals through chained affine blocks until activations, weights, or gradients saturate. Once values hit the rails, updates either explode or vanish, causing learning to stall—not because the task itself is difficult, but because precision collapses. As a result, controlling per-layer gain is a core design challenge for stable on-device training.

We introduce ENABOL, a task-agnostic hardware backpropagation framework that maintains bounded activations, weights, gradients, and updates throughout fixed-point computation. ENABOL constrains the function class through a per-layer operator-norm projection, which limits amplification within each weight matrix and collectively enforces a global Lipschitz budget. This yields a 1-norm Lipschitz network, ensuring that small perturbations to inputs or weights produce proportionally small changes in outputs. In practice, ENABOL enforces per-layer induced-norm bounds—$\|W_l\|_\infty$ for forward sensitivity and $\|W_l\|_1$ for backward sensitivity—via a post-update projection step we call κ-budgeting.

To support these bounded updates directly in hardware, ENABOL extends the High-Level Synthesis (HLS) based tool flow `hls4ml` [17] from inference-only accelerators to in-hardware trainable accelerators. Our framework introduces design automation for differentiable operators, loss modules, gradient buffers, and autograd definitions for backpropagation computation. The `hls4ml` pipeline becomes capable of producing trainable hardware under explicit induced-norm budgets with a single configuration flag:

```
trainable=True
```

We deliberately avoid auxiliary activation clipping or special nonlinear caps; κ-budgeting alone keeps internal signals inside fixed-point envelopes, reducing knobs and simplifying the RTL.

Our evaluation workflow consists of scenarios where models are exposed to post-deployment shift. We first robustly pretrain in software, and then deploy to fixed-point hardware. Once *in situ*, we fine-tune on-chip to recover lost performance. Our experiments span multiple datasets and architectures, including MLPs and CNNs. We consider both affine drift and persistent sensor-column failures as representative forms of correlated, realistic degradation. ENABOL on-chip fine-tuning consistently improves accuracy (by 21.9 absolute percentage points on average, with the largest gains exceeding 35 absolute percentage points), while preserving deterministic fixed-point operation and compatibility with the original accelerator deployment flow. This corresponds to an average of 59% of the lost accuracy being recovered on average across all scenarios and 92% of lost accuracy recovered in the best case. We further demonstrate that these capabilities are immediately realizable on mid-range FPGAs already widely deployed in scientific instrumentation, industrial control, and edge AI—requiring no exotic hardware, no off-chip training infrastructure with substantial data movements, and no changes to the standard `hls4ml` deployment workflow.

ICCAD '26, San Jose, CA, USA

ACM ISBN 978-x-xxxx-xxxx-x/26/07
https://doi.org/10.1145/xxxxxxx.xxxxxxx

2 Related work

In-Hardware Training: A large fraction of prior work focuses on handwritten RTL of backpropagation structures for a specific network architecture [18, 33, 34, 60]. Other works have proposed backpropagation-like schemes in mixed-signal or neuromorphic hardware using analog synaptic devices [26, 41, 49, 58], low-precision float-point quantization [50, 55], or embedding reprogrammable registers on-chip [14, 39, 57]. Surveys analyzed energy-efficient on-device training accelerators and systems [28, 44, 63].

Bounding Training in Software: Early methods to stabilize training in software under adversarial or generative regimes relied on *weight clipping* [3]. While simple, this elementwise clipping is known to distort descent directions and harm convergence. Thus, gradient penalty quickly supplanted it [21]. Reparameterizations such as weight normalization decouple length from direction and speed up convergence but do not enforce absolute bounds on amplification [42]. Direct operator-norm control or Parseval networks improve robustness [10, 31], yet typically assume floating-point math and iterative norm estimation.

Quantized and low-precision training: A large body of work shows that end-to-end training tolerates aggressive precision when quantization error is controlled statistically using techniques such as RangeBatchNorm [5], adaptive fixed-point backpropagation [59], debiasing to recover convergence during ultra-low bit training [8, 45], and optimizer-state quantization [13].

None of these works, however, provide a bounded, stable route to on-chip backpropagation, nor do they offer automation and integration within *the ML accelerator design toolflows to target general network architectures.*

Our Contributions

No prior in-hardware learning framework combines:

A hardware-exact κ-budgeting scheme that projects each weight matrix into an induced-norm ball with power-of-two rescaling, so it compiles directly to `ap_fixed` shifts.

A global Lipschitz budget that allocates per-layer κ_ℓ under a target $\prod_l \kappa_l$, so sensitivity is bounded network-wide rather than layer by layer in isolation.

A fully automated HLS workflow for on-chip learning that inserts backward operators, gradient buffers, and κ-projectors into existing `hls4ml`-style accelerator design pipelines, enabling deployment-compatible adaptation.

A deployment-realistic validation of resilient on-chip adaptation in which pretrained models are fine-tuned directly in fixed-point hardware to recover performance under structured post-deployment shift, including affine drift and persistent sensor-column failures, across multiple datasets and architectures.

> Enabol is the first design framework to generate in-hardware training structures that combine deterministic bounded fixed-point backpropagation, explicit operator-norm control, and an automated HLS-based path to post-deployment on-chip adaptation.

3 Methodology

Our overarching goal is to make fixed-point training in hardware behave like a well-damped control loop: every layer's forward gain and backward sensitivity are explicitly budgeted, every update is maintained within those budgets, and the whole stack respects a global limit on amplification. We achieve this with *κ-budgeting*, a pair of induced-norm caps per layer (one for the forward map, one for backpropagation), enforced as cheap, integer, post-update projections. The result is a numerically stable, hardware-exact learning rule that translates directly into `ap_fixed` shifts and saturating adds, without auxiliary floating-point paths or stochastic rounding.

3.1 Layer activation and gradient norms

An affine[1] block $z_{\text{out}} = Wz_{\text{in}} + b$ is a linear map with worst-case bound $\|z\|_\infty^{\text{out}}$ during forward pass defined by the propagation of the bounds of the input map $\|z\|_\infty^{\text{in}}$, and the ℓ_∞ operator norm (max row-sum) $\|\theta\|_\infty$ of parameters $\theta = \{W, b\}$, as shown in Eq. (1). During backpropagation, the worst-case amplification of the upstream gradient $\|g\|_\infty^{\text{up}}$ is captured by the ℓ_1 norm (max absolute column-sum) $\|W\|_1$ of the weight matrix W, as shown in Eq. (2).

$$\|z\|_\infty^{\text{out}} \le \|W\|_\infty \|z\|_\infty^{\text{in}} + \|b\|_\infty \qquad \|W\|_\infty = \max_j \sum_i |W_{ji}| \quad (1)$$

$$\|g\|_\infty^{\text{up}} \le \|W\|_1 \, \|g\|_\infty^{\text{down}} \qquad \|W\|_1 = \max_i \sum_j |W_{ji}| \quad (2)$$

3.2 Per-layer κ budgets

Since the norms $\|z\|_\infty^{\text{out}}$ and $\|g\|_\infty^{\text{up}}$ depend on the norms of the layer's internal parameters, imposing a *budget* on θ enforces, by construction, a limit precisely on the former values. Similarly, stacking layers sequentially simply multiplies these gains, and introducing local caps for each layer's parameters also guarantees that rails are capped globally. Hence, we introduce three budget values, one for each of the constraints we need to impose onto the parameters: κ_∞, κ_1 and b_∞. These parameters are defined as a safe fraction ρ of the rails of the precision for the parameter being constrained (e.g., for a parameter stored using an `ap_fixed<2,0>` precision, its rail is $R = 1$, and thus one could set $\kappa = \rho R$). Hence, prior to synthesis and deployment, and most importantly *given the bounds for input $\|x\|_\infty$ and output $\|y\|_\infty$ expected for the model*, Enabol computes the rails for each activation and parameter in the model, for both the forward and backward loops, such that each κ budget is established for each layer, a safe distance[2] ρ away from each parameter's rails.

3.3 Projection operators

After each training update, we check whether any parameter's budget has been exceeded and if necessary, we *project* the layer back into its budget. Applying our κ-budgeting means that, at any time, Eq. (3) **must always** hold. For the forward cap, we scan each output row: if its absolute row-sum $\|W\|_\infty$ exceeds κ_∞, we rescale that row uniformly by a power of two so the new sum falls just under the cap κ_∞. We do exactly the same with the bias b. For the

[1] Note that the formulation is equivalent for other types of weight layers such as convolutional or batch-normalization. Even though we introduce dense layers (affine maps) for its simplicity and for clarity here, the caps induced by our budgeting technique hold for any type of linear weight layer.

[2] A $\rho = 1$ value –no headroom– is not advisable, since during updates and just before projection, θ would indeed exceed the rails, thus distorting the gradient $\partial_\theta \mathcal{L}$.

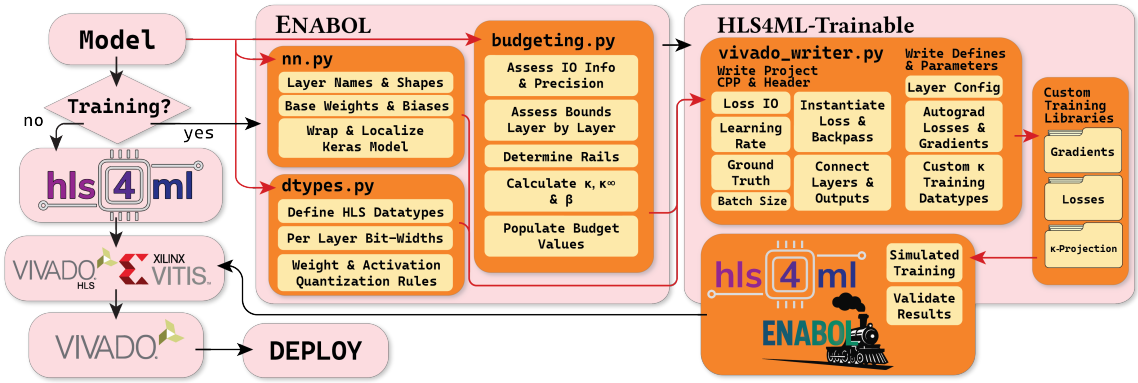

Figure 1: ENABOL toolchain flow and integration with the `hls4ml` ecosystem.

backward cap, we apply the column-wise projection to weights (exact $\|W\|_1$ control). In practice, we implement this projection as a right-shift operation to avoid extra DSPs and divisions.

$$\|z^{(l)}\|_\infty \le \kappa_\infty^{(l)} \|z^{(l-1)}\|_\infty + \|b^{(l)}\|_\infty \qquad \|g_z^{(l-1)}\|_1 \le \kappa_1^{(l)} \|g_z^{(l)}\|_1 \quad (3)$$

3.4 A global Lipschitz budget

A map f is L-Lipschitz when $|f(x)-f(y)| \le L|x-y|$. The Lipschitz constant L quantifies how much the function can amplify signals or perturbations. For differentiable networks, this is equivalent to bounding the Jacobian operator norm $\sup_x |J_f(x)| \le L$; for linear layers[3] , the constant reduces to the induced matrix norm $\|W\|_{\star\to\star}$. Common pointwise activations such as ReLU and LeakyReLU are 1-Lipschitz, so the Lipschitz constant of a multilayer network becomes the product of per-layer norms. Constraining this constant prevents exploding activations and gradients and offers a mathematically grounded mechanism for ensuring stability — especially important for quantized or fixed-point on-chip training.

$$\prod_{l=1}^{L} \kappa_l^{(\infty)} \approx L_{\text{target}}^{(\text{fwd})} \qquad \prod_{l=1}^{L} \kappa_l^{(1)} \approx L_{\text{target}}^{(\text{bwd})} \quad (4)$$

$$\sum_l \Delta_l \approx \log_2 \frac{L_{\text{target}}}{\prod_l \kappa_l} \qquad \kappa_l \leftarrow 2^{\Delta_l} \kappa_l \qquad \Delta_l \in \{-1, 0, +1\} \quad (5)$$

Local caps prevent any *single* layer from running hot; the global product protects the entire *network*. We maintain a target L_{target} for the forward and the backward pass and allocate per-layer budgets $\{\kappa_l\}$ so their product stays near that target, as shown in Eq. (4). Layers that consistently push the envelope of their caps (e.g., 95$^{\text{th}}$ percentile of row/column sums above a threshold) receive one extra bit of budget (cap doubled); under-utilized layers give one bit back. A one-bit change avoids chattering while reacting to drift. This update rule is implemented with a scheduler following Eq. (5).

3.5 Fixed-point bounded training loop

We accumulate per-sample gradients in widened, saturating registers; average by a right-shift of $\log_2 B$ at batch end (with B as the batch size); and pass the averaged gradients to the optimizer (SGD/Adam) in fixed point. Before applying the update to θ, we throttle the step if the implied post-update row-sum would exceed the caps, as introduced in Section 3.3. This throttling is applied using base-2 exponents of every hyperparameter, as shown in Eq (6), so everything can be implemented using shift operations. The update for the bias b is applied analogously.

$$W_{j,(:)} \leftarrow 2^{-k_j} W_{j,(:)} \qquad k_j = \max\Big(0, \lceil \log_2 \|W_{j,(:)}\|_\infty \rceil - \lceil \log_2 \kappa_\infty \rceil\Big) \quad (6)$$

Figure 2 visualizes a trainable neural network generated by ENABOL. The upper path is the forward pass. On the right, we depict κ calculations during the forward pass. These are then used in back pass shown in the lower path. The loss ($\mathcal{L}$), labels (y), batch size (B), and learning rate (η) are used to calculate gradients of the weights and biases. These values propagate backwards through each layer.

3.6 Stability Guarantees on Hardware

Every safety action — averaging, throttling, projection, and even global re-allocation — is a right-shift with round-to-nearest plus a saturating cast. Because we use a single rounding point per sink, `AP_SAT` overflow, and integer control logic, the behavior is deterministic and fully reproducible. Most importantly, the forward and backward maps never exceed their budgets, so the fixed-point dynamic range chosen at synthesis remains valid throughout training, including under optimizer transients and data drift.

3.7 System integration: a one-flag extension

Figure 1 depicts the integration of ENABOL with `hls4ml` and the complete HLS flow. Users retain the familiar YAML/JSON configuration, adding a single option

```
trainable: True
```

which enables on-chip backpropagation under ENABOL's stability measures. The extension supports:

(1) **Full on-chip learning** (all trainable weights/biases).
(2) **Selective fine-tuning** by per-layer toggles to enable learning on only selected layers after software pre-training (e.g., `LayerX.trainable=False`).

[3] The bound holds for affine layers, pooling, and pointwise 1-Lipschitz activations. Layers with unbounded Jacobians (e.g., poorly conditioned normalizations or unbudgeted attention blocks) require their own κ-caps and are outside this scope.

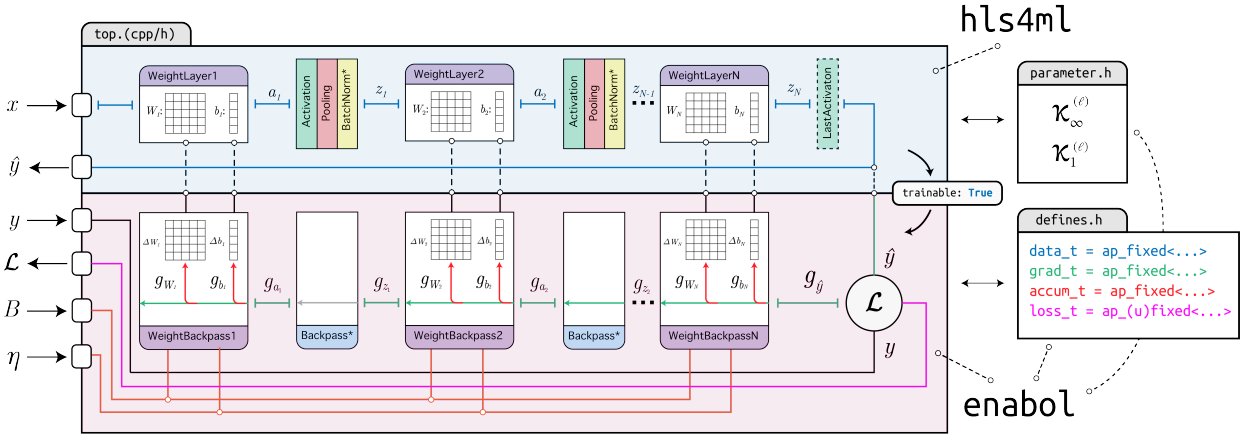

Figure 2: ENABOL forward and back propagation architecture supporting budgeting.

(3) **Deterministic fixed-point execution** (user-specified `ap_fixed` formats) with explicit bounds on all internal signals.

Hardware hooks: At synthesis time, ENABOL inserts:

- Per-layer norm projectors (accumulate ℓ_2 norms, reciprocal-sqrt, vector rescale); power-of-two scaling.
- Budget controller (epoch-rate updates): maintains per-layer κ_l and a global target $\prod_l \kappa_l \approx L_{\text{target}}$ via integer counters/percentiles + hysteresis.

These modules are clock-gated between updates and re-use existing DSPs/BRAM where possible. The driver API mirrors `hls4ml` (stream I/O, DMA), so migrating from inference only to trainable hardware requires no redesign.

4 Theoretical guarantees

Let $\mathcal{S}$ denote the *capped manifold*: the set of parameters for which all per-layer forward and backward budgets hold. With bounded inputs $\|x\|_\infty$ and bounded loss derivatives $\|\mathcal{L}(x, y; \theta)\|_\infty$, the following statements formalize bounded activations/gradients, bounded parameter increments, and the invariance of $\mathcal{S}$ under the throttled update plus projection presented in this paper.

Lemma 1 (Bounded activations and gradients). If $\|W^{(l)}\|_\infty \le \kappa_\infty^{(l)}$ and $\|b^{(l)}\|_\infty \le \beta_l$, then activations remain bounded as in Eq. (7). If $\|W^{(l)}\|_1 \le \kappa_1^{(l)}$ and the output gradient is bounded in ℓ_1, then upstream gradients are also bounded and satisfy Eq. (8). This encapsulates the core safety property: forward and backward amplification cannot exceed the product of per-layer budgets, so neither activations nor gradients can explode in fixed-point.

$$\|z^{(L)}\|_\infty \le \Big(\prod_{l=1}^{L} \kappa_\infty^{(l)}\Big)\|z^{(0)}\|_\infty + \sum_{r=1}^{L}\Big(\prod_{l=r+1}^{L} \kappa_\infty^{(l)}\Big)\beta_r \tag{7}$$

$$\|g^{(0)}\|_1 \le \Big(\prod_{l=1}^{L} \kappa_1^{(l)}\Big)\|g^{(L)}\|_1 \tag{8}$$

Lemma 2 (Bounded parameter increments). The integer throttling step s_j in Eq. (9) guarantees that the scaled update $\Delta\widetilde{W}_{j,(:)}^{(l)} = 2^{-s_j}\eta\,\Delta W_{j,(:)}^{(l)}$ cannot violate the row-wise budget, as stated in Eq. (10).

$$s_j = \max\Big(0,\ \lceil \log_2 \|W_{j,(:)}\|_1 \rceil - \lceil \log_2 \kappa_\infty \rceil\Big) \tag{9}$$

$$\big\|W_{j,(:)}^{(l)} - \Delta\widetilde{W}_{j,(:)}^{(l)}\big\|_1 \le \kappa_l^{(\infty)} \tag{10}$$

Corollary 1 (Positive invariance of $\mathcal{S}$). We define the nonnegative "distance-to-cap" functional $\mathcal{V}$ in Eq. (11), where $[\cdot]_+$ represents the ReLU operator. Under one ENABOL step, $\mathcal{V}$ is nonincreasing and decreases strictly whenever any cap is exceeded; hence $\mathcal{S}$ is positively invariant.

$$\mathcal{V}(W, b) = \sum_l \Big[W_\infty^{(l)} - \kappa_\infty^{(l)} + W_1^{(l)} - \kappa_1^{(l)} + \|b^{(l)}\|_\infty - \beta_l\Big]_+ \tag{11}$$

Lemma 3 (Bounded Jacobian; no gradient explosion). Let J_θ represent the Jacobian of the network with respect to its inputs and parameters θ. Under the caps, the forward Jacobian and parameter gradients are bounded as in Eq. (12) — where $\kappa_\star^{(l)}$ denotes $\kappa_\infty^{(l)}$ or $\kappa_1^{(l)}$ and Φ collects bounded data and loss terms–, thus ruling out gradient explosion in fixed-point and bounding the per-step parameter motion under the throttled update.

$$\|J_\theta\|_\star \le \prod_{l=1}^{L} \kappa_\star^{(l)}, \qquad \|\nabla_\theta \mathcal{L}\| \lesssim \Big(\prod_{l=1}^{L} \kappa_\star^{(l)}\Big)\Phi(x, y) \tag{12}$$

Corollary 2 (Deterministic safety; projected convergence). With `AP_SAT` arithmetic, widened accumulators, and integer rounding, all intermediate quantities remain bounded by Eqs. (7)–(12). In convex settings (e.g., a single affine layer with MSE), the ENABOL step reduces to projected gradient descent onto $\mathcal{S}$ and converges to the constrained minimizer:

$$\theta_{t+1} = \prod\nolimits_{\mathcal{S}}\big(\theta_t - \eta_t \nabla\mathcal{L}(\theta_t)\big) \ \to\ \arg\min_{\theta\in\mathcal{S}} \mathcal{L}(\theta). \tag{13}$$

For general deep networks, κ-budgeting guarantees stability . Convergence to stationary points under standard smoothness and step-size assumptions can then be analyzed via the usual nonconvex (projected) gradient descent theory [7, 19, 27].

5 Experiments

We evaluate ENABOL in a deployment-oriented setting where a model is pretrained in software, deployed to fixed-point FPGA hardware, and then fine-tuned on-chip to recover performance after post-deployment accuracy degradation. Our results demonstrate that (i) training remains stable in fixed point, and (ii) the accelerator hardware can restore useful performance even if it is exposed to input distributions that deviate from the design-time conditions.

(i) ***Dynamic Condition: Structured affine drift.***
Our first scenario depicting dynamic variation is a correlated affine drift of the input stream. For an input sample X, the deviation in the input is defined as

$$X \leftarrow \text{clip}(aX + b,\ X_{\min},\ X_{\max})$$

where the gain a and bias b control the severity of the drift and the clipping operation enforces the valid sensor or datatype range. This perturbation models structured changes such as gain mismatch, calibration drift, saturation, or slowly varying operating conditions. Figure 3 illustrates representative examples of such variation across image and audio modalities for 3 of the 4 datasets used.

(ii) ***Persistent localized sensor failure.***
Our second mode of variation in the operating environment reflects localized persistent sensor-column damage, in which one or more input columns are permanently zeroed across all subsequent samples. Unlike affine drift, which shifts the statistics of the entire input in a correlated way, this corruption removes localized information from a fixed spatial region and therefore models sensor hardware defects such as damaged readout columns or persistent acquisition artifacts. An example of this effect on images is shown in Figure 6, on top of each bar.

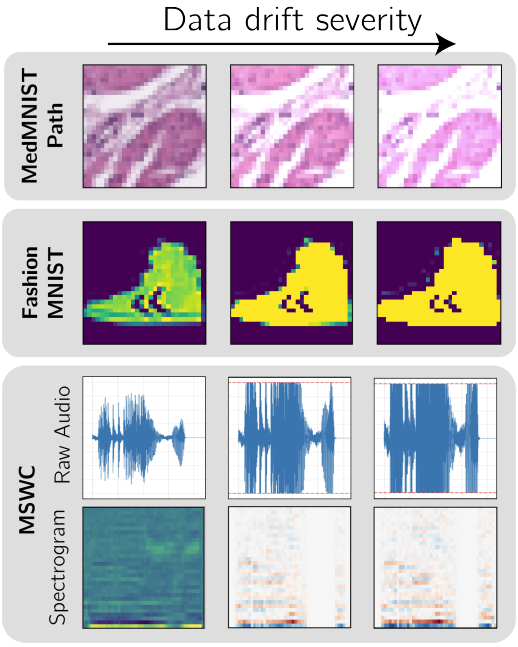

Figure 3: Structured affine drift with clipping, illustrating correlated gain and bias distortion across modalities.

These dynamic variation scenarios are intentionally different from uncorrelated stochastic perturbations such as i.i.d. Gaussian noise. Random environmental noise can often be characterized in advance and incorporated into calibration or offline model training. In contrast, affine drift and persistent sensor faults are structured, persistent, and deployment-specific, making them harder to anticipate accurately at design time and therefore more appropriate for evaluating post-deployment adaptation.

In such settings, relying only on reprogrammable inference hardware would typically require collecting degraded data, storing or transferring it to a host processor, retraining offline, and then reloading updated weights into the device. That workflow is not only cumbersome, but in many edge AI deployments it can be operationally impractical due to latency, bandwidth, storage, uptime, or privacy constraints. ENABOL instead enables this adaptation step to occur directly on-chip within the same bounded fixed-point accelerator flow, without requiring repeated offline retraining or external weight reprogramming.

5.1 Tasks and benchmarks

To evaluate ENABOL across a representative range of deployment conditions, we consider both image and audio workloads, and pair them with both MLP and convolutional neura; network (CNN) models, as summarized in the table below. FashionMNIST [54] and MedMNIST/Path [56] represent visual classification tasks with different image structure, from clothing silhouettes to biomedical tissue patterns. JetTagging [35, 37, 38] represents a structured scientific tabular task evaluated with an MLP, while Multilingual Spoken Words Dataset (MSWC) [30] provides a keyword-spotting in speech workload evaluated with a CNN. Together, these tasks span multiple data modalities, feature structures, and model classes.

Dataset	Input type	Model	Task & purpose
FashionMNIST	Grayscale image	CNN	Visual drift recovery
MedMNIST/Path	Biomedical image	CNN	Drift & sensor-fault recovery
JetTagging	Scientific tabular	MLP	Non-image shift adaptation
MSWC	Audio / spectral	CNN	Non-image CNN adaptation

Models are built in `TensorFlow/Keras`, pretrained in software, and then passed through the ENABOL+`hls4ml` workflow to generate HLS-ready training testbenches and fixed-point hardware implementations. During evaluation, `csim` executes the full hardware

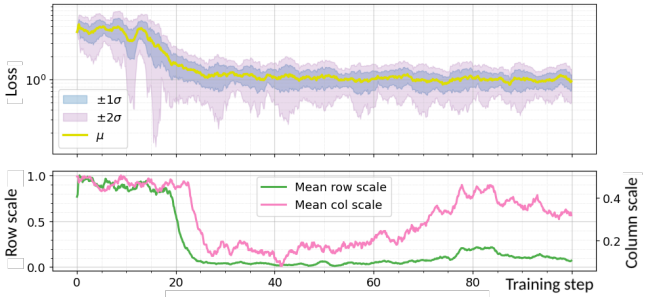

Figure 4: Loss and κ row/column scaling evolution for fine-tuning MedMNIST/Path under 0.25 drift case in hardware.

training loop as an independent fixed-point learner. A budgeting pass assigns `ap_fixed` datatypes across activations, gradients, accumulators, and updates under the global κ-constraint, after which the validated designs are synthesized for a Xilinx® Kintex FPGA representative of a mid-range FPGA device used in edge AI deployment. For a more detailed comparison, we have also synthesized these designs onto other FPGA families.

5.2 Post-deployment data drift and recovery

We first evaluate recovery under the structured affine drift introduced above. Figure 5 summarizes the resulting recovery in accuracy across four workloads: MedMNIST/Path and FashionMNIST using CNNs, JetTagging using an MLP, and MSWC using a CNN.

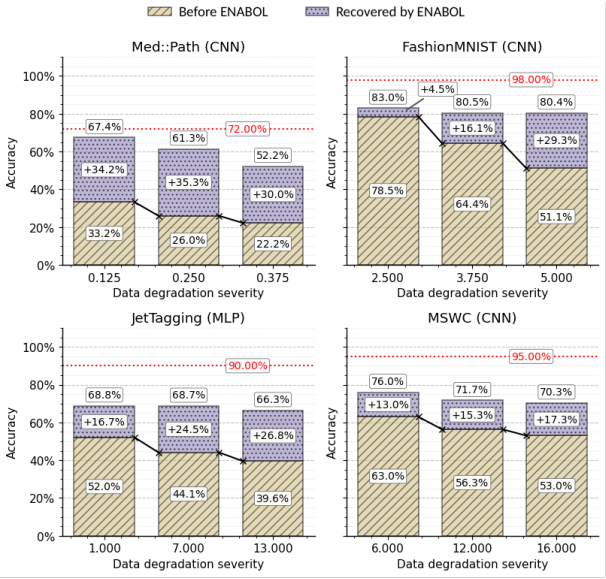

Figure 5: Recovery under structured affine drift after on-chip fine-tuning with ENABOL.

In all cases, the beige bars indicate the degraded accuracy observed when the pretrained model is evaluated directly under drift, while the purple bars show the accuracy recovered after on-chip fine-tuning with ENABOL; the red dotted line denotes the original no-drift reference accuracy of software training.

For MedMNIST/Path, as drift severity increases, the pretrained model drops its performance from the original 72.0% reference to 33.2%, 26.0%, and 22.2%, respectively. ENABOL recovers the performance back to 67.4%, 61.3%, and 52.2%, respectively. In the strongest case, this approximately doubles the degraded accuracy, improving it from 33.2% to 67.4%. Figure 4 illustrates the evolution of in-hardware training under 0.25 drift. The top graph illustrates the loss and achievement of convergence. The bottom subplot shows the κ budgeting interventions. The *row scale* (κ_∞) represents the scaling applied to the weights, per row, such that the total gain remains within boundaries; while the *col scale* (κ_1) represents the scaling applied per column, such that gradients stay within a safe margin for backpropagation.

FashionMNIST shows the same trend, dropping from the original accuracy of 98% to degraded accuracies of 78.5%, 64.4%, and 51.1% and recovering back to 83.0%, 80.5%, and 80.4% upon retraining. JetTagging and MSWC also show consistent recovery, demonstrating that the effect is not limited to image tasks and extends to structured tabular and audio workloads as well.

Across the evaluated affine-drift settings, on-chip fine-tuning improves accuracy by 21.9±9.5 absolute points on average, with the largest gain reaching 35.3 points. This corresponds to an average of 59% of the lost accuracy being recovered on average across all scenarios and 92% in the best case.

5.3 Persistent sensor-column failure

We next evaluate ENABOL under persistent sensor-column faults by permanently zeroing one or more contiguous image columns across all deployed samples. Fault severity is escalated by increasing the number of defective columns while keeping the corruption pattern fixed across the deployed input stream. Figure 6 summarizes the resulting recovery for the MedMNIST/Path CNN. The style and colors of the plot match the ones introduced in subsection 5.2.

As fault severity worsens, the pretrained model degrades down to 43.4%, 35.3%, and 28.3% accuracy, respectively, while on-chip adaptation elevates performance up to 69.7%, 67.3%, and 63.4%, respectively. Across the three tested fault severities, on-chip fine-tuning improves accuracy by 31.1±3.7 percentage points on average, with the largest gain reaching 35.1 points. These results indicate that ENABOL can recover useful performance even after persistent, spatially localized sensor degradation.

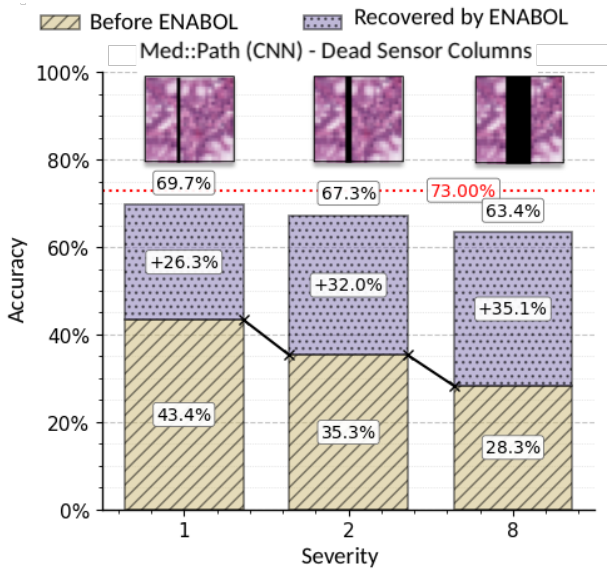

Figure 6: Recovery under persistent sensor-column failure after on-chip fine-tuning with ENABOL.

Table 1: FPGA logic synthesis resource usage for inference-only and ENABOL'ed (trainable) models, including latency and deployment compatibility.

Benchmark	Variant	Resources				Latency		Fits All FPGA Targets
		LUTs	FFs	DSPs	BRAMs	(cycles)	(μs @ 5ns clk)	
JetTagging	Inference-only	17,175	32,451	21	5	4,286	21	✓
	ENABOL'ed (Trainable)	194,424	141,166	625	150	63,840	319	✓
MedMNIST	Inference-only	30,992	63,876	10	10	11,128	56	✓
	ENABOL'ed (Trainable)	159,480	163,688	684	189	110,009	550	✓
MSWC	Inference-only	34,328	78,759	10	80	14,304	72	✓
	ENABOL'ed (Trainable)	129,099	154,540	539	212	138,030	690	✓
FashionMNIST	Inference-only	10,629	24,689	12	2	2,295	11	✓
	ENABOL'ed (Trainable)	72,317	77,399	436	45	31,642	158	✓
FPGA Resource Capacities	`Kintex UltraScale (XCKU035)`	203,128	406,256	1,700	1,080			
	`ZCU104 (XCZU7EV)`	230,400	460,800	1,728	1,080			
	`Alveo U250 (XCU250)`	1,728,000	3,456,000	12,288	5,376			

5.4 Hardware implementation and trade-offs

Table 1 summarizes our hardware implementation analysis across our benchmarks. The baseline in each case is the forward-pass-only `hls4ml` generated design configured for maximum resource efficiency. Crucially, the backward pass does **not hinder forward-pass inference output**: the prediction is produced as soon as inference completes, and training proceeds afterward. Therefore, the circuit delays of the forward-pass structure and the latency of the inference are not affected by the addition of the on-chip training feature. This design decision was important for edge-AI workloads with strict latency budgets (e.g. closed-loop control, embedded vision, real-time experimental analysis). Designers may still tune HLS parameters such as reuse factor and pipeline depth to trade off forward-pass latency and resource usage as they would during their design space search for the optimal inference structure. Aside and independent of that, ENABOL imposes essentially the same resource tax regardless of those parameters. The forward-backward resource gap in Table 1 should therefore be interpreted as just one point in a broader latency–resource design space. For a forward pass structure that is further optimized for latency, its relative size with respect to the training hardware would be larger (since training hardware does not change) and this gap would be smaller.

The forward inference remains same as the baseline inference designs and independent of the addition of training and it ranges between 11 and 72 μs. For the trainable configurations, the reported latency includes gradient computation, operator-norm projection, and weight updates, i.e., the full cost of enabling on-chip learning relative to inference only. Across benchmarks, the latency of the training path ranges between 158 to 690 μs. Moreover, gradient updates occur only when the system asserts `apply_update` at the top-level interface.

In many target deployments, the backward pass does not require ultra-low latency. Applications such as real-time physics data acquisition, embedded vision, and long-duration industrial or environmental monitoring are often affected by concept drift, where sensor aging, environmental variation, or changing operating conditions gradually degrade model performance. Mitigating such drift typically requires only **periodic** adaptation rather than microsecond-scale continual retraining [6, 11, 29, 63]. This is precisely why ENABOL exposes control over **when and how often** updates are applied. In this context, the observed latencies are a practical trade-off as ultra low-latency inference is still preserved while enabling on-chip training only when needed.

The measured full Fwd+Bwd latencies (~158–690 μs) are also practical relative to software alternatives. A comparable batch-size-1 iteration on the host CPU takes roughly 650 μs to 3 ms for our benchmark models, and small-model GPU execution at batch size 1 is likewise known to be latency-inefficient [23]. Unlike host- or GPU-based adaptation, ENABOL also avoids PCIe or host-device transfer overhead by completing both the backward pass and weight update directly in the on-chip sensor readout datapath.

We evaluate the resource requirements of the on-chip training structures created by ENABOL against several representative FPGA targets in Table 1. This selection reflects the wide array of deployment contexts, performance regimes, and application scenarios for FPGA AI accelerators as documented in the literature [9, 12, 40]. The Kintex™ UltraScale 035 represents a practical mid-range device family used in edge AI, scientific instrumentation, and commercial data-acquisition products [4, 16, 36, 52], while remaining relatively affordable [1]. The AMD Zynq™ UltraScale+™ MPSoC ZCU104 is a widely used embedded-vision platform [2]. The Alveo U250 represents a high-end datacenter-class platform suitable for scaling to larger models and higher-throughput workloads and has also been leveraged in many scientific and other deep learning applications [15, 20, 43, 47]. All ENABOL'ed designs can be accommodated by the target devices illustrating that ENABOL is well suited for FPGAs already widely deployed in edge AI and scientific systems.

Overall, ENABOL provides design automation for on-chip training with predictable latency and resource overhead while preserving immediate forward inference and leaving designers flexibility to choose when, and how often, adaptation occurs.

6 System level implications

The fundamental value proposition of Enabol is not performance on any single benchmark, but three broader contributions:

(i) ***A theoretical formulation for stable online training.*** As discussed in this paper, adding the training feedback loop can easily destabilize a fixed-point system, leading to register saturation, gradient explosion, or activation death. In practice, this is why online learning in edge devices is often either highly application-specific or restricted by simple constraining techniques that preserve safety at the cost of trainability. Enabol addresses this directly through κ-budgeting, providing a hardware-exact formulation that stabilizes on-chip training without sacrificing the ability to adapt.

(ii) ***The elimination of the data capture–retrain–redeploy bottleneck.*** In conventional reprogrammable AI, adapting to distribution shift requires capturing data, transmitting it off-chip, retraining in software, re-validating, and re-flashing –a cycle that can take hours to days and may require the system to go offline entirely. Enabol replaces this with continuous on-chip adaptation: the same FPGA that sits in the sensor datapath houses the learning algorithm, weight updates occur at the point of data acquisition with zero transfer overhead, and the forward-pass latency budget is never touched. For systems where the data is too voluminous, too sensitive, or too time-critical to move off-chip [6, 11, 24, 29], or where privacy constraints make data transfer undesirable, this bottleneck becomes especially deterrent.

(iii) ***Complete automation into existing flows with adaptation via*** `hls4ml`**.** Because Enabol is implemented as an integratable component of the existing `hls4ml` workflow, designers can adopt trainable hardware with minimal disruption to established inference-oriented pipelines. This lowers the barrier to experimentation with post-deployment adaptation and makes stable on-chip learning accessible within a familiar design flow.

7 Conclusions

We introduced Enabol, a general-purpose toolchain that enables stable, fixed-point backpropagation for trainable neural network accelerators, while requiring only a single configuration flag in the existing `hls4ml` pipeline. At the core of Enabol is a hardware-exact κ-budgeting scheme that applies per-layer induced-norm projections under a global Lipschitz budget, ensuring bounded activations, gradients, and updates throughout training. Our evaluation demonstrates that Enabol maintains stable learning dynamics under fixed-point quantization during training, while also recovering a substantial portion of the performance lost under structured post-deployment degradation, including affine drift and persistent sensor-column failure, across multiple datasets and architectures. More broadly, Enabol provides a practical foundation for on-device learning and hardware–algorithm co-design within standard HLS workflows for FPGAs and also potentially for ASICs.

References

[1] AMD. n.d.. XCKU035-1SFVA784I Kintex UltraScale FPGA. https://www.digikey.com/en/products/detail/amd/XCKU035-1SFVA784I/6132047. Digi-Key Electronics product page. Accessed: 2026-04-10.

[2] AMD. n.d.. Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit. https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/zcu104.html Accessed: 2026-04-10.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. PMLR, 214–223.

[4] S Astrand, L Boggia, M Borsato, L Bozianu, C E Cocha Toapaxi, F I Giasemis, et al. 2026. Perspective on machine learning for real-time analysis at the Large Hadron Collider experiments ALICE, ATLAS, CMS and LHCb. *Machine Learning: Science and Technology* 7, 1 (jan 2026), 013001. doi:10.1088/2632-2153/ae35cc

[5] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2018. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems* 31 (2018).

[6] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 119–135. https://www.usenix.org/conference/nsdi22/presentation/bhardwaj

[7] Léon Bottou, Frank E Curtis, and Jorge Nocedal. 2018. Optimization methods for large-scale machine learning. *SIAM review* 60, 2 (2018), 223–311.

[8] Xinrui Chen, Yizhi Wang, Yao Li, Xitong Ling, Mengkui Li, Ruikang Liu, Minxi Ouyang, Kang Zhao, Tian Guan, and Yonghong He. 2024. Low bit-width zero-shot quantization with soft feature-infused hints for iot systems. *IEEE Internet of Things Journal* (2024).

[9] Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, Simon J. L. Billinge, Elizabeth Holm, Shyue Ping Ong, and Chris Wolverton. 2022. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials* 8, 1 (2022), 59. doi:10.1038/s41524-022-00734-6

[10] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. 2017. Parseval networks: Improving robustness to adversarial examples. In *International conference on machine learning*. PMLR, 854–863.

[11] Zachary A. Daniels, Jun Hu, Michael Lomnitz, Phil Miller, Aswin Raghavan, Joe Zhang, Michael Piacentino, and David Zhang. 2023. Efficient Model Adaptation for Continual Learning at the Edge. arXiv:2308.02084 [cs.LG] https://arxiv.org/abs/2308.02084

[12] Allison McCarn Deiana, Nhan Tran, Joshua Agar, Michaela Blott, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Scott Hauck, Mia Liu, Mark S. Neubauer, Jennifer Ngadiuba, Seda Ogrenci-Memik, et al. 2022. Applications and Techniques for Fast Machine Learning in Science. *Frontiers in Big Data* Volume 5 - 2022 (2022). doi:10.3389/fdata.2022.787421

[13] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2021. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861* (2021).

[14] Giuseppe Di Guglielmo, Farah Fahim, Christian Herwig, Manuel Blanco Valentin, Javier Duarte, Cristian Gingu, Philip Harris, James Hirschauer, Martin Kwok, Vladimir Loncar, et al. 2021. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. *IEEE Transactions on Nuclear Science* 68, 8 (2021), 2179–2186.

[15] Javier Duarte, Philip Harris, Scott Hauck, Burt Holzman, et al. 2019. FPGA-Accelerated Machine Learning Inference as a Service for Particle Physics Computing. *Computing and Software for Big Science* 3, 1 (Oct. 2019). doi:10.1007/s41781-019-0027-2

[16] Euresys. n.d.. Coaxlink Quad CXP-12 Frame Grabber. https://www.euresys.com/en/products/frame-grabber/coaxlink-quad-cxp-12/ Accessed: 2026-04-10.

[17] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. 2021. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579* (2021).

[18] Rafael Gadea, Joaquín Cerdá, Franciso Ballester, and Antonio Mocholí. 2000. Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation. In *Proceedings of the 13th international symposium on System synthesis*. 225–230.

[19] Saeed Ghadimi and Guanghui Lan. 2013. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM journal on optimization* 23, 4 (2013), 2341–2368.

[20] Fotis I. Giasemis, Vladimir Lončar, Bertrand Granado, and Vladimir Vava Gligorov. 2025. Comparative Analysis of FPGA and GPU Performance for Machine Learning-Based Track Reconstruction at LHCb. In *2025 23rd IEEE Interregional NEWCAS Conference (NEWCAS)*. 533–537. doi:10.1109/NewCAS64648.2025.11106977

[21] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. *Advances in neural information processing systems* 30 (2017).

[22] Houxuan Guo, Manuel Blanco Valentín, Xiuyuan He, and Seda Ogrenci. 2025. Toward Reconfigurable In-Pixel Computing: A Fault-Tolerant Design Flow for Machine Learning Accelerators. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 261–267.

[23] Jussi Hanhirova, Teemu Kämäräinen, Sipi Seppälä, Matti Siekkinen, Vesa Hirvisalo, and Antti Ylä-Jääski. 2018. Latency and throughput characterization of convolutional neural networks for mobile computer vision. In *Proceedings of the 9th ACM Multimedia Systems Conference* (Amsterdam, Netherlands) *(MMSys '18)*. Association for Computing Machinery, New York, NY, USA, 204–215. doi:10.1145/3204949.3204975

[24] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, et al. 2023. RECL: Responsive Resource-Efficient continuous learning for video analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 917–932.

[25] Aymen Rayane Khouas, Mohamed Reda Bouadjenek, Hakim Hacid, and Sunil Aryal. 2024. Training machine learning models at the edge: A survey. *arXiv preprint arXiv:2403.02619* (2024).

[26] Dongseok Kwon, Suhwan Lim, Jong-Ho Bae, Sung-Tae Lee, Hyeongsu Kim, Young-Tak Seo, Seongbin Oh, Jangsaeng Kim, Kyuho Yeom, Byung-Gook Park, et al. 2020. On-chip training spiking neural networks using approximated backpropagation with analog synaptic devices. *Frontiers in neuroscience* 14 (2020), 423.

[27] Guanghui Lan, Tianjiao Li, and Yangyang Xu. 2024. Projected gradient methods for nonconvex and stochastic optimization: new complexities and auto-conditioned stepsizes. *arXiv preprint arXiv:2412.14291* (2024).

[28] Jinsu Lee and Hoi-Jun Yoo. 2021. An overview of energy-efficient hardware accelerators for on-device deep-neural-network training. *IEEE Open Journal of the Solid-State Circuits Society* 1 (2021), 115–128.

[29] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review . *IEEE Transactions on Knowledge & Data Engineering* 31, 12 (Dec. 2019), 2346–2363. doi:10.1109/TKDE.2018.2876857

[30] Mark Mazumder, Sharad Chitlangia, Colby Banbury, Yiping Kang, Juan Manuel Ciro, Keith Achorn, Daniel Galvez, Mark Sabini, Peter Mattson, David Kanter, et al. 2021. Multilingual spoken words corpus. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[31] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957* (2018).

[32] Ji Joong Moon, Hyun Suk Lee, Jiho Chu, Donghak Park, Seungbaek Hong, Hyungjun Seo, Donghyeon Jeong, Sungsik Kong, and MyungJoo Ham. 2022. A new frontier of ai: On-device ai training and personalization. *arXiv preprint arXiv:2206.04688* (2022).

[33] Hesham Mostafa, Bruno Pedroni, Sadique Sheik, and Gert Cauwenberghs. 2017. Hardware-efficient on-line learning through pipelined truncated-error backpropagation in binary-state networks. *Frontiers in neuroscience* 11 (2017), 496.

[34] Subadra Murugan, K Packia Lakshmi, Jeyanthi Sundar, and K MathiVathani. 2014. Design and implementation of multilayer perceptron with on-chip learning in virtex-e. *AASRI Procedia* 6 (2014), 82–88.

[35] Patrick Odagiu, Zhiqiang Que, Javier Duarte, Johannes Haller, Gregor Kasieczka, et al. 2024. Ultrafast jet classification at the HL-LHC. *Machine Learning: Science and Technology* 5, 3 (jul 2024), 035017. doi:10.1088/2632-2153/ad5f10

[36] Pentek, Inc. n.d.. Jade XMC PCIe Family Overview. https://www.aes-eu.com/pentek-jade-xmc-pcie.php. Accessed: 2026-04-10.

[37] Maurizio Pierini, Javier Mauricio Duarte, Nhan Tran, and Marat Freytsis. 2020. *HLS4ML LHC Jet dataset (150 particles)*. doi:10.5281/zenodo.3602260 Machine Learning for Particle Physics (EU Open Research Repository), Version v1.

[38] Huilin Qu, Congqiao Li, and Sitian Qian. 2022. Particle transformer for jet tagging. In *International Conference on Machine Learning*. PMLR, 18281–18292.

[39] Adam Quinn, Manuel B Valentin, Thomas Zimmerman, Davide Braga, Seda Memik, and Farah Fahim. 2023. A cryogenic readout ic with 100 ksps in-pixel adc for skipper ccd-in-cmos sensors. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[40] Maithra Raghu and Erica Schmidt. 2020. A Survey of Deep Learning for Scientific Discovery. *ArXiv* abs/2003.11755 (2020). https://api.semanticscholar.org/CorpusID:214667531

[41] Eugenio Ressa, Alberto Marchisio, Maurizio Martina, Guido Masera, and Muhammad Shafique. 2024. Tinycl: An efficient hardware architecture for continual learning on autonomous systems. *arXiv preprint arXiv:2402.09780* (2024).

[42] Tim Salimans and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems* 29 (2016).

[43] Prabhav Saxena, Arjun Gupta, Shivansh Chadha, Suhani Grover, Sujal Gupta, and Anup Kumar Mandpura. 2024. Visual Servoing using FPGA-based Hardware Accelerated Deep Learning Solution. In *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Vol. 2. 1–6. doi:10.1109/IATMSI60426.2024.10502825

[44] Duckgyu Shin, Naoya Onizawa, Warren J. Gross, and Takahiro Hanyu. 2020. Training Hardware for Binarized Convolutional Neural Network Based on CMOS Invertible Logic. *IEEE Access* 8 (2020), 188004–188014. doi:10.1109/ACCESS.2020.3029576

[45] Xiao Sun, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, Vijayalakshmi Viji Srinivasan, and Kailash Gopalakrishnan. 2020. Ultra-low precision 4-bit training of deep neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 1796–1807.

[46] Iuliia Topko, Alexey Serdyuk, Tanja Harbaum, and Jürgen Becker. 2025. Hardware-Accelerated On-Device Learning: Training, Partitioning, and Compilation for Constrained Edge AI. In *2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Vol. 1. IEEE, 1–6.

[47] Vasiliy S. Usatyuk and Sergey I. Egorov. 2025. Boosting DNN Efficiency: Replacing FC Layers with Graph Embeddings for Hardware Acceleration. In *2025 27th International Conference on Digital Signal Processing and its Applications (DSPA)*. 1–6. doi:10.1109/DSPA64310.2025.10977895

[48] Manuel Valentin, Chinar Syal, Davide Giri, Farah Fahim, Giuseppe Di Guglielmo, Joseph Zuckerman, Luca Carloni, Maico Cassel Dos Santos, Nhan Tran, and Seda Memik. 2022. CryoAI – Prototyping cryogenic chips for machine learning at 22nm. https://indi.to/L8qZG.

[49] Eveline RW Van Doremaele, Tim Stevens, Stijn Ringeling, Simone Spolaor, Marco Fattori, and Yoeri van de Burgt. 2024. Hardware implementation of backpropagation using progressive gradient descent for in situ training of multilayer neural networks. *Science Advances* 10, 28 (2024), eado8999.

[50] Shreyas Kolala Venkataramanaiah, Jian Meng, Han-Sok Suh, Injune Yeo, Jyotishman Saikia, Sai Kiran Cherupally, Yichi Zhang, Zhiru Zhang, and Jae-Sun Seo. 2023. A 28-nm 8-bit floating-point tensor core-based programmable CNN training processor with dynamic structured sparsity. *IEEE Journal of Solid-State Circuits* 58, 7 (2023), 1885–1897.

[51] Xubin Wang, Zhiqing Tang, Jianxiong Guo, Tianhui Meng, Chenhao Wang, Tian Wang, and Weijia Jia. 2025. Empowering edge intelligence: A comprehensive survey on on-device ai models. *Comput. Surveys* 57, 9 (2025), 1–39.

[52] Y. Wei, R. F. Forelli, C. Hansen, J. P. Levesque, N. Tran, J. C. Agar, G. Di Guglielmo, M. E. Mauel, and G. A. Navratil. 2024. Low latency optical-based mode tracking with machine learning deployed on FPGAs on a tokamak. *Review of Scientific Instruments* 95, 7 (07 2024), 073509. doi:10.1063/5.0190354

[53] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. 2019. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 331–344.

[54] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv:cs.LG/1708.07747 [cs.LG]

[55] Qinkai Xu, Yijin Liu, Yuan Meng, Yang Chen, Yunlong Mao, Li Li, and Yuxiang Fu. 2025. LT-OAQ: Learnable Threshold Based Outlier-Aware Quantization and its Energy-Efficient Accelerator for Low-Precision On-Chip Training. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–6.

[56] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. 2023. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific data* 10, 1 (2023), 41.

[57] Jieun Yoo, Jennet Dickinson, Morris Swartz, Giuseppe Di Guglielmo, Alice Bean, Douglas Berry, Manuel Blanco Valentin, Karri DiPetrillo, Farah Fahim, Lindsey Gray, et al. 2024. Smart pixel sensors: towards on-sensor filtering of pixel clusters with deep learning. *Machine Learning: Science and Technology* 5, 3 (2024), 035047.

[58] Qingtian Zhang, Huaqiang Wu, Peng Yao, Wenqiang Zhang, Bin Gao, Ning Deng, and He Qian. 2018. Sign backpropagation: An on-chip learning algorithm for analog RRAM neuromorphic computing systems. *Neural Networks* 108 (2018), 217–223.

[59] Xishan Zhang, Shaoli Liu, Rui Zhang, Chang Liu, Di Huang, Shiyi Zhou, Jiaming Guo, Qi Guo, Zidong Du, Tian Zhi, et al. 2020. Fixed-point back-propagation training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2330–2338.

[60] Zhenyu Zhang, Guangsen Wang, Kang Wang, Bo Gan, and Guoyong Chen. 2023. Efficient On-Chip Learning of Multi-Layer Perceptron Based on Neuron Multiplexing Method. *Electronics* 12, 17 (2023), 3607.

[61] Xiaohan Zhao, Hualin Zhang, Zhouyuan Huo, and Bin Gu. 2023. Accelerated on-device forward neural network training with module-wise descending asynchronism. *Advances in Neural Information Processing Systems* 36 (2023), 52265–52292.

[62] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.

[63] Shuai Zhu, Thiemo Voigt, Fatemeh Rahimian, and Jeonggil Ko. 2024. On-device training: A first overview on existing systems. *ACM transactions on sensor networks* 20, 6 (2024), 1–39.