Error Adaptive Classifier Boosting (EACB): Leveraging Data-Driven Training Towards Hardware Resilience for Signal Inference

Zhuo Wang, *Student Member, IEEE*, Robert E. Schapire, and Naveen Verma, *Member, IEEE*

***Abstract*—The continued scaling of CMOS technologies and consideration of post-CMOS technologies has elevated hardware reliability to a first-class challenge, particularly in energy- and resource-constrained embedded sensor applications. In such applications, there is an increasing emphasis on inference functions. Machine-learning algorithms play an important role by enabling the construction of data-driven models for inference over data that is too complex to model analytically. This paper explores how data-driven training can be exploited to also overcome computational errors due to hardware faults within an inference stage. FPGA emulation with randomized fault injections shows that the proposed architecture restores system performance to the level of a fault free system, with $<1\%$ of the hardware requiring explicit fault protection, and with digital faults affecting $>2\%$ of the circuit nodes in the rest of the hardware. To train an error-aware inference model, a training algorithm is presented whose hardware (memory) and energy requirements are reduced by $65\times$ and $10\times$ compared to previously reported algorithms (AdaBoost and FilterBoost respectively), thereby enabling model construction entirely on the device.**

***Index Terms*—Circuit reliability, fault tolerance, pattern classification, pattern recognition.**

I. Introduction

MACHINE-LEARNING algorithms are becoming increasingly critical in embedded sensing applications [1]–[3], as they enable the construction of data-driven models for analyzing signals that are otherwise too complex to model analytically. Algorithms for classification and regression are of particular interest due to the importance of embedded recognition functions [4]. Aside from the value that machine learning brings for analyzing application signals, recent studies have also begun to expose its substantial potential for overcoming complex non-idealities in the platform hardware itself. In [5], [6], an approach called data-driven hardware resilience (DDHR) was presented that overcomes high levels of digital faults and data-conversion/instrumentation non-linearities by utilizing a classification algorithm to model data *in the presence of the resulting errors*. This enables high performance despite severe errors. In the DDHR systems, however, the inference stage applying the classification model is sensitive to errors due to hardware faults. Although generally machine-learning algorithms exhibits some level of inherent resilience [7], [8] against computational errors, this is found to be inadequate in the face of the high error rates and error magnitudes that occur due to even modest levels of hardware faults. Thus, in the DDHR systems, the inference stage itself needs to be fault-free, requiring 10–30% of the hardware to be explicitly fault protected [6]. Unfortunately, as shown in [9], the inference stage scales with the complexity of the model applied, likely making it a dominating component in DDHR systems as both application signals and platform hardware scale in various dimensions.

To address this, we present an approach referred to as error-adaptive classifier boosting (EACB) wherein the inference stage is itself allowed to be highly affected by faults. The basic idea of EACB was presented in [10]. This work extends that idea by proposing and evaluating a practical system architecture, wherein faults are addressed across the entire architecture (feature extractor and classifier), and details for embedded training of the classifier are developed. Faults affecting over 0.01% of circuit nodes in the feature extractor and over 2% of circuit nodes in the classifier itself are overcome, restoring performance to a level near that of a fully fault-free system. This is achieved with less than 1% fault-protected hardware in the real-time detection system. The specific contributions of this work are as follows:

1) We present the EACB approach, and demonstrate its operation within a system for epileptic-seizure detection using sensed electroencephalogram (EEG) data. The demonstration is based on an FPGA implementation, which permits injection of faults at controllable rates and in a randomized manner, enabling quantitative characterization. Extending the work in [10], experiments are performed considering faults in both the feature-extraction and classification stages, and detailed rationale is provided behind the fault models employed.
2) We present an algorithm and architecture to efficiently train the EACB system entirely on the platform at run time, the details of which have not been presented before. In particular, data-driven training typically requires a large training set to adequately represent the data statistics. This leads to high energy and large embedded memory. For the demonstrated system, our algorithm reduces the memory by $> 65\times$ and energy by $> 10\times$ compared to two state-of-the-art algorithms (AdaBoost and FilterBoost, respectively). Further, our architecture generates training labels

Manuscript received August 11, 2014; revised November 19, 2014; accepted January 06, 2015. Date of current version March 27, 2015. This work was supported by SRC, NSF (CCF-1253670), as well as Center for Future Architectures Research (CFAR) and Systems on Nanoscale Information fabriCs (SONIC), two of the six SRC STARnet Centers, sponsored by MARCO and DARPA. This paper was recommended by Associate Editor M. Seok.

Z. Wang and N. Verma are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: zhuow@princeton.edu; nverma@princeton.edu).

R. E. Schapire is with the Department of Computer Science, Princeton University, Princeton, NJ 08544 USA, and also Microsoft Research, New York, NY 10019 USA (e-mail: schapire@cs.princeton.edu).

Digital Object Identifier 10.1109/TCSI.2015.2395591

itself, enabling training without any inputs from an external expert/oracle.

3) We present an experimental design-space exploration and analysis of the critical architectural parameters. Beyond the initial design points considered in [10], here we describe the design-space trade-offs in detail. We quantify the trade-offs (gate-count/memory, energy, fault resilience) associated with the real-time hardware (feature extractor, classifier) and non-real-time hardware (trainer). This establishes guidelines for the design of EACB systems.

The remainder of this paper is organized as follows. Section II presents background both on existing fault-resilience approaches (with an emphasis on DDHR) as well as the key principles utilized in EACB. Section III first discusses the specific design objectives (including the fault types targeted), then presents an overview of EACB along with an architecture for efficient embedded training. Section IV then defines the design space of interest and the experimental approach, and finally presents results and analysis. Finally, Section V concludes.

II. Background

A. Emerging Approaches to Fault Tolerance

The increasing susceptibility of advanced-CMOS technologies [11]–[13] and the projected susceptibility of post-CMOS technologies (carbon nanotubes, tunneling FETs) [14], [15] to device-level variabilities and manufacturing defects has prompted substantial research in fault resilience. Many emerging directions focus on algorithmic and architectural approaches rather than circuit- and device-level approaches, since 1) higher-level approaches are showing increased leverage for handling faults [16], and 2) many applications enable some level of error tolerance, offering various alternatives for overcoming the effects of faults [17]. Prominent examples include algorithmic noise tolerance (ANT) [18], soft N-modular redundancy (Soft NMR) [19], and stochastic sensor network on chip (SSNOC) [20]. A key attribute in the implementation of all these is the use of some, preferably minimal, fault-protected component. As another example, error-resilient system architecture (ERSA) [8] employs an explicitly fault-protected programmable core to enable resilient, programmable control operations over fault-prone data-processing cores.

B. Data-Driven Hardware Resilience (DDHR)

The approach of DDHR aims to minimize the fault-protected hardware by incorporating the error-handling mechanism within an existing system-level algorithmic framework. More importantly, at the system level, it is able to overcome errors in the context of the input data and the output results of interest. As a result, it achieves a more fundamental level of resilience, shown to be set by the level of information retained for the processing of interest.

DDHR applies to systems that employ data-driven modeling for implementing inference functions. Specifically, it achieves resilience against faults by utilizing data derived from fault-affected hardware in order to train a model for classification or regression [5], [21]. The concept was demonstrated on a supervised classification framework as shown in Fig. 1(a). The framework consists of a trainer and a detector. The trainer employs previously observed data to learn statistically how the data corresponds with the inference of interest (i.e., the class membership). The detector performs inference on newly observed data based on the constructed model by employing a classification kernel. Generally, faults occur randomly and cause unpredictable errors; by using error-affected data, DDHR enables modeling of the data statistics in the presence of the particular faults within a given instance of hardware. A machine-learning algorithm, implemented on a small kernel of fault-protected hardware is exploited within DDHR to construct and apply the model. Fig. 1(b) shows a DDHR system that was demonstrated for EEG-based seizure detection. The gate counts (from RTL synthesis) for the fault-prone and the fault-protected hardware (shown in gray) are given in the table. The fault-affected blocks include feature-extraction stages while the fault-protected blocks include machine-learning classifiers and a microcontroller, used for infrequent model training. The classifier used in [5], [21] is a support-vector machine (SVM) [22]. Being a supervised-learning algorithm, an SVM requires training data, in this case derived from the fault-affected hardware, as well as training labels. While labels are conventionally provided by an external expert, in [5] an architecture capable of estimating labels without any external inputs is proposed. This involves the use of a temporary error-free system implemented in software on the fault-protected microcontroller. Although the software implementation is energy intensive and cannot run in real time, it can be actuated temporarily during training, providing class declarations as training labels. This was shown to yield performance close to that achieved by training with perfect labels.

Fig. 1(c) illustrates DDHR. On the left, the data and classification model from a fault-free system are shown; on the right the data from a fault-affected system (emulated using an FPGA) are shown. The resulting errors substantially alter the feature-vector distributions; however, a new classification model, called an *error-aware model*, can be learned and constructed for the resulting distributions. The errors can thus be viewed as altering the way that information for classification is encoded. With a strong learner, capable of fitting complex distributions, the severity of the errors is no longer critical. In fact, it is shown that performance close to a fault-free system is achieved even at high fault rates (i.e., fault affecting over 0.02% of the circuit nodes), causing high bit-level error (i.e., bit-error rates of 20–50%). Rather, what is critical is the fundamental level of *information* retained in the new encoding. This is demonstrated in [21] by computing the mutual information (between the error-affected data and the class membership) and showing that this exhibits strong correspondence with the system performance.

Though DDHR is thus able to achieve a more fundamental level of performance in the presence of errors, we see from Fig. 1(a) that substantial fault-protected hardware is required. While the gate counts are modest in the demonstrated systems [21], we expect the machine-learning kernels to scale with the complexity of the model required [9]. Thus, as both the application signals and the fault-prone platform scale in complexity, these kernels become a dominating aspect. This work focuses on making the inference kernel itself capable of overcoming high rates of faults. This is achieved by exploiting a machine-learning algorithm know as Adaptive Boosting (AdaBoost).

C. Adaptive Boosting (AdaBoost)

AdaBoost is a machine-learning algorithm that achieves a strong classifier through a combination of iteratively-trained

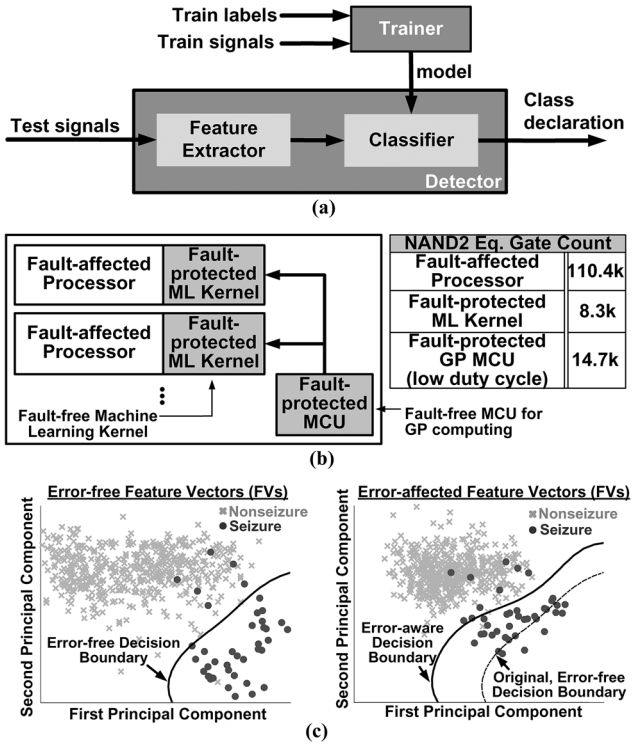


Fig. 1. Illustration of DDHR [21]. (a) Supervised classification framework consists of a trainer and a detector performing inference function. (b) The architecture of an implemented EEG-based seizure detector uses a small kernel of fault-protected hardware (shown in gray) to enable high system-level performance. (c) The DDHR concept is based on constructing an error-aware (EA) model of the error-affected EEG-feature data (principal component analysis is used to visualize the high-dimensionality feature data).

weak classifiers [23]. Weak classifiers are typically classifiers whose output may be only weakly correlated with the true class. In practice, this often comes about due to the use of simple decision rules, unable to fit complex distributions of the data. In this work, the concept is extended by treating *fault-prone classifiers* as weak classifiers. The aim is not only to boost the final decision rule, but also, as in the case of DDHR, to enable adaptive training in response to unpredictable classifier output statistics generated due to random hardware faults.

In addition to forming a fault-resilient strong classifier, this work also develops an efficient algorithm for embedded training, which can thus be achieved entirely on a low-power device. As we describe in Section III-C, a key challenge is the amount of memory required for the training data in order to ensure low generalization error. To address this, we build on the idea of FilterBoost [24], which is set in the scenario where training data can be acquired sequentially from an oracle to refine the classification model during iterative training stages of the weak classifiers. In this work, we propose a new algorithm extending FilterBoost to reduce both the computational energy of acquiring the training data and the instantaneous memory required for training each iteration.

III. Error-Adaptive Classifier Boosting (EACB)

The aims of the EACB system are as follows: 1) achieving a strong classifier based on data-driven learning that enables scalable decision rules as well as minimal energy and hardware complexity for real-time classification; 2) high classification performance in the presence of very high fault rates; 3) minimizing the ratio of fault-protected to fault-prone hardware;

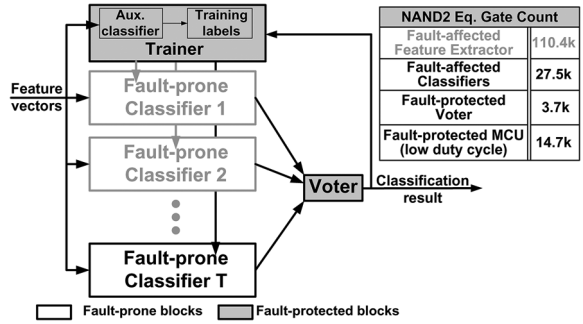


Fig. 2. Illustration of error adaptive classifier boosting (EACB) system architecture.

and 4) embedded model training with low energy and low hardware overhead. The following subsections describe the EACB architecture, implementation (along with design parameters), and training algorithm.

A. Architecture of Error Adaptive Classifier Boosting

The architecture for EACB is shown in Fig. 2. It leverages the AdaBoost algorithm that we previously discussed in Section II-C. The classifier consists of a set of T weak classifiers, which are fault affected, as well as voter and trainer blocks, which are both fault protected. Generally, protection against faults can be achieved through various means (e.g., selectively using a higher supply voltage in low-voltage systems [25], [26], selectively employing conservative physical design rules for deeply-scaled technology manufacturing [27], etc.) The table shows actual gate counts from the hardware experiments of Section IV for the fault-affected and fault-protected block, both derived from synthesis of RTL to a gate-level netlist. We point out that although the classifier requires somewhat more hardware than that in the previous DDHR systems [Fig. 1(b)], in both cases the classifier hardware represents a small portion of the overall systems.

The principle behind EACB extends from DDHR [10]. EACB is based on the recognition that a processing stage capable of data-driven training introduces the possibility of learning the statistics of its input data. Given that hardware faults in the preceding stages cause errors whose effect is to alter these statistics, data-driven training enables desired outputs to be produced in the presence of these errors as long as the information required to derive the desired outputs is preserved in the resulting statistics. As shown in [5], [21], the information is very often preserved by a fault-prone processor even in the case of severe output errors. To exploit this idea, EACB implements classification via a structure where data-driven training is performed in *successive* stages, so that each stage raises some possibility of training to, and thus overcoming, errors in the previous stages. Such a classifier structure can be trained using the AdaBoost algorithm, wherein weak classifiers are trained in an iterative manner. However, a critical aspect in EACB is that a weak-classifier's output depends not only on the input data and the corresponding classification model (derived from training), but also on the error statistics generated by faults within the weak classifier. The aim is thus to train subsequent weak classifiers in response to the resulting error statistics. During each iteration of training, the error affected outputs from previous iterations are fed to the trainer and used along with training data (feature vectors) to establish 1) the

weight values for classifier voting, and 2) the weight distribution for applying the training data in subsequent iterations; the algorithm for training is described in Section III-C below.

B. Weak Classifier for Minimal Fault Protection

The choice and implementation of the weak classifiers strongly influences overall performance (i.e., tradeoff between accuracy and simplicity of weak classifiers [23], [28]), training complexity, and level of fault tolerance. Among the various classifiers that have been considered for boosting (support vector machines [29], neural networks [30], decision trees, and decision stumps [28], [31]), decision trees and decision stumps enable minimal training complexity and, as described below, offer the potential for very high fault tolerance within the EACB architecture.

A critical aspect for fault tolerance is the manner in which computation control is handled in the hardware implementation. A requirement from the theory of AdaBoost is that the weak classifiers must provide *some* correlation with the true class membership (i.e., must be better than 50-50 guessing) [23]. While data-path faults perturb (perhaps severely) the output statistics generated by a classifier, the probability of obtaining outputs exhibiting some correlation remains high. However, control-path faults can often result in degenerate outputs, which might thus be inadequate for even a weak classifier within AdaBoost. An implementation that predefines the tree structure in hardware, as in [32], minimizes the control path (while also enhancing speed and energy efficiency). Alternatively, [33] exploits the similarity between decision trees and threshold networks, implementing the nodes as a "hidden layer" whose outputs are used to construct a logical function for classification; this once again minimizes the control path. However, in EACB a high degree of programmability in the classification model is critical for error-adaptive training. In both implementations above, the programmability is too limited. On the other hand, implementations that roll the tree into a single universal node (e.g., [34]), offer a high degree of programmability (while also minimizing the hardware), but rely too heavily on a control path.

Fig. 3 shows the implementation of the classification processor developed in this work with decision tree weak classifiers. The T classifiers (TREE 1 to TREE T) all employ the same hardware, and the voter is implemented as a T-input signed adder for deriving the final hypothesis h, where the sign bit is provided by the weak-classifier outputs Y_t and the corresponding weight W_t is provided from a register file. The decision-tree implementation consists of three stages. The first stage essentially implements the nodes of the decision tree by testing the n features $(x_1, x_2, \cdots, x_n)$ composing a feature vector $\vec{X}$ against their corresponding thresholds $(th_1, th_2, \cdots, th_n)$ via the comparison blocks CMP. The benefit of performing this operation first is that the n-dimensional feature vector is reduced to n bits, corresponding to the binary values evaluated at the n nodes. The second stage uses m n-to-1 multiplexers MUX to essentially select the nodes to be included in the tree, also derived from model training. Notice that typically trees will have fewer nodes m than the total number of features n. Accordingly, in the first stage, only thresholds for the m selected features need to be considered (others can be set arbitrarily, to zero in our implementation). With such a structure, the number of nodes in the tree is limited to m during training. Finally, the third stage

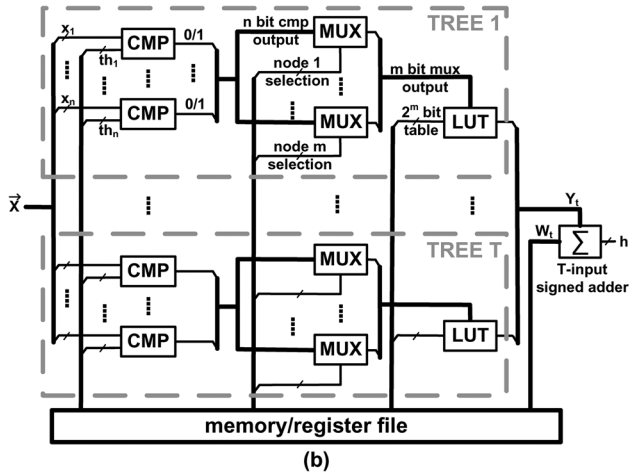


Fig. 3. Classification processor consisting of T m-node decision-tree weak classifiers operating on n-dimensional feature vectors.

uses the m-bit output from the multiplexers as the index to a look-up table (LUT) to derive the single-bit classifier output. The LUT entries are also derived from training. While such an implementation minimizes control-flow circuitry, for improved fault resilience, it does lead to increased hardware complexity. Even so, it is worth mentioning that taking the seizure detector presented in Section IV as an example, the energy per classification (estimated from hardware measurements) is reduced by a factor of 16$\times$ compared to an SVM classifier with RBF kernel implemented with an accelerator (from 2.22 μJ to 0.14 μJ).

A key aspect of the classifier implementation is that it minimizes the control path while enabling a high degree of model programmability, limiting only the number of nodes in the tree. The primary cost paid is that the tree is computed in a flattened manner. Where a tree conventionally enables a logarithmic computation scaling, now the computation scales linearly with the number of features (first stage) and linearly with the total number of nodes (second stage). To reduce the number of features, the training algorithm (described below) performs feature selection by exploiting a metric computed off-line during an initial training phase (though feature selection helps to reduce the classifier complexity, as we describe below, it is primarily performed to reduce trainer complexity). To reduce the number of nodes, we pursue design-space exploration over the tree parameters (Section IV). We find that in practice a tree with very few nodes can be used without substantially increasing the number of iterations required in the overall classifier. As a result, the computational complexity of the classifiers can be small.

C. Low-Hardware, Low-Energy Trainer

To implement the trainer, EACB employs a small fault-protected microcontroller (in the system presented, an OpenMSP core is used [35]). In a supervised-learning algorithm such as AdaBoost, classifier training requires both feature-vector data and training labels. The training feature vectors are derived at runtime by the system. The training labels are estimates, as in the case of DDHR [5], [21], that correspond to the output from a temporary error-free classifier implemented in software on the fault-protected microcontroller. Although a software implementation of the classifier is energy intensive and cannot run in

	Training Pool Size (N)	Feature Dimension	Feature Data Size	Memory Usage of Features
w/ mem. red.	5000	42	16 bit	420 kB
w/o mem. red.	200	16	16 bit	6.4 kB

Fig. 4. Memory usage comparison for adaptive-boosting training and the proposed on-line training algorithm (the case of weak-classifiers based on decision stumps is considered).

real-time, it is viable for infrequent training phases. The dominant concern during each iteration of training is the hardware complexity and energy required to compute the weak-classifier model. In particular, this is limited by the large amount of training data typically needed to ensure generalization of the classifier model, resulting in a large embedded memory and large number of computation cycles. Below, we describe a training algorithm thus optimized to reduce the memory and energy of embedded training.

1) Memory Reduction: Fig. 4 considers the memory required for training the seizure detector (based on decision-stump weak classifiers) presented in Section IV, both before and after optimizations. As shown, without applying the proposed optimizations, AdaBoost training requires a training batch size of approximately 5000 training-data instances to ensure adequate data diversity. Considering feature vectors based on EEG spectral-energy distribution (having a dimensionality of 42 and a per-feature bit width of 16 bits, as described in Section IV-C), 420 kB of memory is required just for storing the training data. As shown in the second row, the memory requirements in this work are reduced to 6.4 kB (for the case of decision stumps, discussed in Section IV-D). This is achieved through two approaches: 1) feature-vector dimensionality reduction via a learner-driven selection metric; and 2) training-set reduction (while ensuring generalization) through an algorithm leveraging the idea of FilterBoost [24].

With regards to feature-dimensionality reduction, approaches have included feature construction via space-dimensionality reduction and reduced-set selection [36]. Feature-construction methods such as principal component analysis (PCA) raise additional computational requirements both during training and classification, elevating energy and susceptibility to faults. Reduced-set selection, on the other hand, aims to simply discard features that are least informative. While there exist generic metrics such as mutual information to assess the features, here we employ a metric directly related to the choice of weak-classifier employed, namely decision trees. This metric is computed during off-line training of a generic fault-free boosted classifier; training of one such classifier is required for generating training labels for all fault-affected instances of the system. During this training process, a histogram is computed of how often each feature is utilized within the set of weak-classifiers (i.e., for the decision-tree nodes). Then, features most frequently used by the generic classifier are selected. Thus, this metric aims to identify features that are of greatest value for the classifier implementation chosen. Fig. 5 is an example of such a computed histogram from the system presented in Section IV. The black bars identify the sixteen features thus selected, which can be seen to occur with substantially higher frequency than the others.

With regards to training-set reduction, learning theory tells us that for batch AdaBoost algorithms, the generalization error of training will increase as we reduce the training set size [23]. So, to reduce the size of the training set for on-line training,

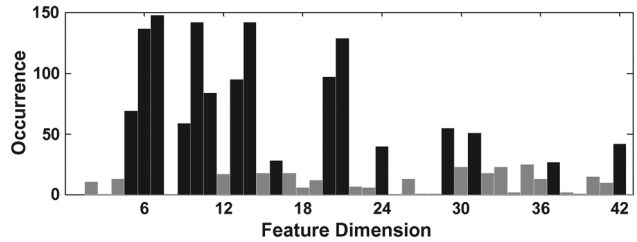


Fig. 5. Histogram for evaluating how informative each feature is.

while maintaining low generalization error, we leverage an idea from FilterBoost [24]. Generally, boosting algorithms can be grouped into two types: 1) boosting by filtering, and 2) boosting by reweighting. In boosting by filtering, training-data instances arrive in a stream, and the algorithm selects which instances to use for training and which to discard. In boosting by reweighting, the algorithm takes a batch of training-data instances, which are used (and reused) every iteration, but with new weighting applied to the batch instances. FilterBoost is a "boosting-by-filtering" algorithm. Standard AdaBoost is a "boosting-by-reweighting" algorithm [23]. Compared to standard AdaBoost, FilterBoost draws new data for each iteration of training. This allows us to reduce the training set with respect to that required for generalization in a standard AdaBoost algorithm. This is possible because generalization is enhanced through boosting thanks to diverse training sets employed for each iteration. We leverage this to enhance the diversity of the data set across the training iterations, permitting the data set used at each iteration to be substantially smaller. However, the original FilterBoost algorithm does not specifically target data-set minimization at each iteration, and it incurs high energy for computing metrics to select the train-data instances. So we introduce a modified on-line training algorithm, which takes advantage of both "boosting by filtering" and "boosting by reweighting." Like "boosting by filtering," we learn from a stream of instances. This addresses training memory. However, as shown in the following section, the instances are weighted rather than being filtered. This addresses training energy. For the system presented in Section IV, the proposed on-line training algorithm reduces the memory required at each iteration to 200 instances and the energy by $> 10\times$ (compared to FilterBoost algorithm). Combined with feature-dimensionality reduction, the total memory required is thus reduced to 6.4 kB, as shown in Fig. 4, representing $65\times$ reduction compared to a standard AdaBoost algorithm.

2) Energy Reduction: The proposed algorithm, achieving both memory and energy reduction, is specified in Algorithm 1. Within the system, training data $\vec{x}$ with label y is acquired at runtime from sensor data source S, and the trainer is initiated at selected periods (either once at startup or periodically during the lifetime of the device).

Compared to the FilterBoost algorithm, the major differences in the proposed on-line training algorithm are as follows. First, the FilterBoost algorithm assumes that a source of data S is available from which instances preferred for the training set can be selected. This is done by computing a weight D_i for an instance under consideration, and accepting that instance with a probability corresponding to the weight. In the targeted system, the data $\vec{x}$, y ultimately presented to the EACB trainer is being actively sensed at run time and is thus constrained. For this

input : data source S for generating training set
total number of weak classifiers T
training pool size N
output: classification hypothesis $\mathtt{H}$

// get initial training set
1 $(\vec{\mathsf{x}}_i^{(0)}, \mathsf{y}_i^{(0)}) \leftarrow \mathtt{getData}(\mathsf{S}), i \in \{1, \ldots, \mathsf{N}\}$;
// initialize hypothesis
2 $\mathtt{F}^{(0)}(\vec{\mathsf{x}}) \leftarrow 0$;
3 **for** $t \leftarrow 1$ **to** T **do**
// assign training weight
4 $\mathsf{d}_i^{(t-1)} \leftarrow 1/(1 + \exp(\mathsf{y}_i^{(t-1)} \cdot \mathtt{F}^{(t-1)}(\vec{\mathsf{x}}_i^{(t-1)})))$;
5 $\mathtt{normalize}(\mathsf{d}_i^{(t-1)}), i \in \{1, \ldots, \mathsf{N}\}$;
// get weak classifier
6 $\mathtt{WC}^{(t)} \leftarrow \mathtt{train}(\vec{\mathsf{x}}^{(t-1)}, \mathsf{y}^{(t-1)}, \mathsf{d}^{(t-1)})$;
// get new training set
7 $(\vec{\mathsf{x}}_i^{(t)}, \mathsf{y}_i^{(t)}) \leftarrow \mathtt{getData}(\mathsf{S}), i \in \{1, \ldots, \mathsf{N}\}$;
// get weak classifier weight
8 $\varepsilon^{(t)} \leftarrow \frac{\sum_i 1\{\mathsf{y}_i^{(t)} \neq \mathtt{WC}^{(t)}(\vec{\mathsf{x}}_i^{(t)})\}}{\mathsf{N}}, i \in \{1, \cdots, \mathsf{N}\}$;
9 $\alpha^{(t)} \leftarrow \frac{1}{2} log \frac{1-\varepsilon^{(t)}}{\varepsilon^{(t)}}$;
// set current hypothesis
10 $\mathtt{F}^{(t)}(\vec{\mathsf{x}}) \leftarrow \mathtt{F}^{(t-1)}(\vec{\mathsf{x}}) + \alpha^{(t)} \cdot \mathtt{WC}^{(t)}(\vec{\mathsf{x}})$;
11 **end**
// output final hypothesis
12 $\mathtt{H} \leftarrow \mathtt{sign}(\mathtt{F}^{(\mathsf{T})})$;

Algorithm 1: Proposed on-line training algorithm

reason, we fix the training set size to N for each iteration, and accept all actively sensed data for training (Line 1, 7). This precludes the need to compute an acceptance probability. In this way, the data acquired for on-line training is not filtered, but rather associated with a weight d for training (Line 4). Second, in the FilterBoost algorithm, each iteration $t = 1, \cdots, T$ of training requires two subsets of data that are explicitly diverse (i.e., independent samples). The first subset is used to train the current weak classifier $WC^{(t)}$ (i.e., establish the nodes, thresholds, and structure of the decision tree). The second subset is used to derive the weight $\alpha^{(t)}$ associated with the current weak classifier $WC^{(t)}$ [i.e., for voting in the overall classifier $F^{(T)}$ (Line 12)]. However, in the proposed algorithm, only one new data set $x^{(t)}$, $y^{(t)}$ is required at each iteration. The current weak classifier $WC^{(t)}$ is trained using the data set acquired during the previous iteration $x^{(t-1)}$, $y^{(t-1)}$ (Line 6) and the current weak-classifier's weight $\alpha^{(t)}$ is derived using a newly acquired data set $x^{(t)}$, $y^{(t)}$ (Line 9) (this data set is then used to train the next weak classifier). Third, the FilterBoost algorithm derives the weak-classifier's weight α by continually applying training data until a probabilistic confidence bound is met for the computed weight. However, the proposed algorithm uses the acquired data set and derives a best estimate for the weight based on this.

Both the FilterBoost algorithm and the proposed algorithm were implemented. On the tested cases, the classifier performance was close. Execution clock-cycle counts were determined through cycle-accurate simulation of an MSP430[1] microcontroller, which is also used for the implementation and experiments in Section IV. For each iteration of training, the original algorithm requires ∼55 M clock cycles, while the proposed algorithm requires only ∼5.0 M clock cycles, making embedded training much more viable. The proposed algorithm is used for the experiments in Section IV.

IV. Experiments

In the subsections below, we first describe the design space over which experimentation is performed. Then we describe the details of the experimentation approach. Next, we present the EEG-based seizure-detection system used for demonstration. Finally, we present the results and analysis.

A. Evaluation Metrics and Design Space

Given the aims of EACB, the critical metrics for evaluation are the following: 1) the fraction of hardware that must be fault protected; 2) the energy and hardware complexity of the low-duty-cycle trainer (consisting of a microcontroller with embedded memory); 3) the energy and hardware complexity of the real-time classifier (consisting of the weak classifiers and a voter); and 4) the fault-rates tolerable while maintaining application-level performance. As described in Section III-B, the choice of weak classifier strongly influences these factors. We focus on decision trees due to their scalability thanks to simple implementation, their comparatively low-complexity training algorithm, and their fault tolerance due to the potential to minimize the control path. However, the parameters of the decision tree, most notably the number of nodes, has important implications on EACB (recall the reduced-control-path decision-tree implementation proposed also fixes the number of nodes). Thus, for design exploration, we thus consider three tree structures: 1) 7-node trees; 2) 4-node trees; and 3) 1-node trees (i.e., stumps).

B. Experimental Approach

For experimental demonstration, we focus on FPGA emulation. While custom IC prototyping is directly affected by manufacturing defects and variation scenarios, it is difficult to introduce resulting faults in a controllable and suitably-randomized manner to enable characterization. Simulation make such a characterization possible but unviable. The reason is the algorithmic approach of EACB requires system-level simulation, but accurate representation of faults necessitates models derived from practical technological defects, necessitating low levels of abstraction (transistor, gate level). System simulations over adequately large datasets are unviable at such levels.

On the other hand, FPGA emulation enables representation of faults at the gate level and enables processing of enough data to adequately evaluate the prototyped system. The emulation flow is shown in Fig. 6(a). An RTL description (Verilog) of the EACB system is synthesized into a gate-level netlist using an ASIC logic library. Faults are then introduced within the gate-level netlist. The fault model employed should have strong correspondence with physical fault sources (defects and variation) within hardware. We thus focus on stuck-at-1/0 faults, which are representative of a wide range of physical fault sources in CMOS processing (such as random dopant fluctuations affecting static

[1]The specific MSP430 architecture used is MSP430F5438A.

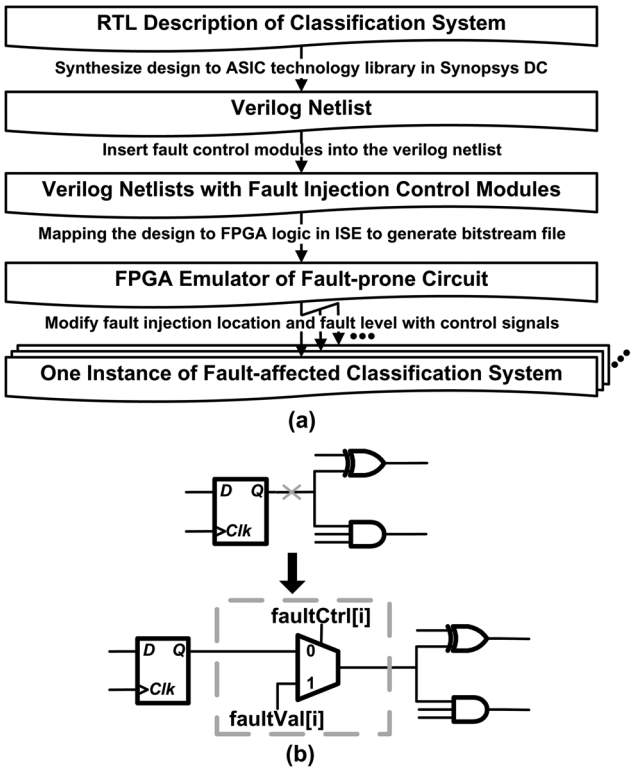


Fig. 6. Approach for emulating a fault-prone system [21]. (a) The FPGA mapping flow starts by synthesizing RTL of the processor and ends with mapping a circuit in which faults can be activated. (b) Configurable faults are introduced by adding multiplexers for enforcing stuck-at-0/1 nodes in the circuit.

noise margins in low-voltage circuits [25], [37], lithographic defects affecting nano-scale manufacturing [38], [39]). To introduce the faults, the gate-level netlist is edited (via a script) by including 1) multiplexers on a large number of nodes, and 2) a fault-control module for generating signals ($faultCtrl$ and $faultVal$) to control the multiplexers. Each multiplexer drives its output node with either the intended signal or with a static logic 1/0, as illustrated in Fig. 6(b). The fault-control module consists of a register file that programmably activates fault configurations over the multiplexed circuit nodes. Recognizing that, in addition to the fault rate, the precise nodes affected by faults can strongly impact the errors, multiple fault configurations are tested for each fault rate. A second FPGA is used strictly as an Ethernet transceiver, to load configuration data into the fault-control module and to retrieve output data from the system to a host PC [21].

C. Application for Demonstration

For experimentation, we employ EACB in a system for EEG-based detection of epileptic seizures. The system consists of a feature-extraction stage as well as a classifier and trainer. The feature-extraction processing is shown in Fig. 7. The features correspond to the spectral-energy distribution of each EEG signal channel [40]. Since the band of interest is below 20 Hz, the EEG signals sampled at 600 Hz are down-sampled by 8 with a decimation filter. This results in 75 Hz samples, which are then fed to a bank of seven band-pass filters (BPF1 to BPF7) with center frequencies from 0 Hz to 19 Hz and bandwidth of 3 Hz. The magnitudes of the resulting output samples are then accumulated over a two-second epoch (150

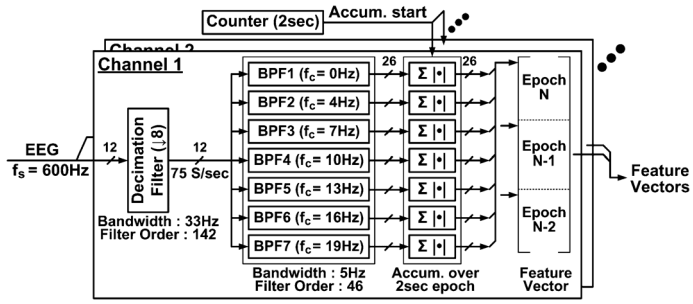


Fig. 7. Architecture for feature extractor of EEG-based seizure-detection system [21].

samples) to represent the spectral energies. This results in seven features per EEG channel, which are then concatenated with those from two previous epochs, giving a total of 21 features per channel. With two EEG channels used in the application, the total feature-vector dimensionality is 42. As mentioned, three implementations are considered for the decision trees used within the classifier (7-, 4-, 1-node trees). The training algorithm runs on a microcontroller, implemented using an OpenMSP core [35]. For this, both the training algorithm and the temporary error-free classifier for deriving training labels are coded in C and compiled for execution.

EEG data for testing is obtained from the MIT-CHB Seizure Database [41], [42]. The application-level performance metrics are as follows [40]: 1) sensitivity, corresponding to the percentage of seizure correctly detected; 2) number of false alarms; and 3) latency, corresponding to the delay in detecting a seizure compared to the point of onset declared by an expert (expert-annotated labels are provided in the dataset). A total of 10 k seconds of EEG data are used for testing.

D. Experimental Results

In this section, we present experimental results, first introducing faults only in the classifier, to demonstrate extension of previous approaches to DDHR (which required the classifier to be fault protected). Then we introduce faults in the entire system, demonstrating that EACB simultaneously achieves DDHR over errors in the classifier *and* feature extractor.

1) System Performance With Faults: Faults are injected on circuit nodes of the classifier at a rate ranging from 0% (representing fault-free system performance) up to 3%. Five cases of fault configurations are tested at each rate, with the fault-affected nodes randomly selected in each case. The system performance employing EACB is shown in Fig. 8, along with that not employing EACB (i.e., with weak-classifiers models and weights based on training a generic classifier rather than training the particular fault-affected classifiers via the on-line training algorithm). As shown for the fault rates considered, with EACB system performance is consistently at the level of the error-free case (represented in the plots by the start-shaped marker). On the other hand, without EACB, system performance rapidly degrades, as reflected by reduced sensitivity and elevated false alarms across the cases tested. This highlights the limited inherent resilience of the machine-learning algorithm in the face of errors caused by even modest levels of faults. As seen, a similar trend is observed for all tree structures considered, suggesting a design space compatible with the EACB architecture.

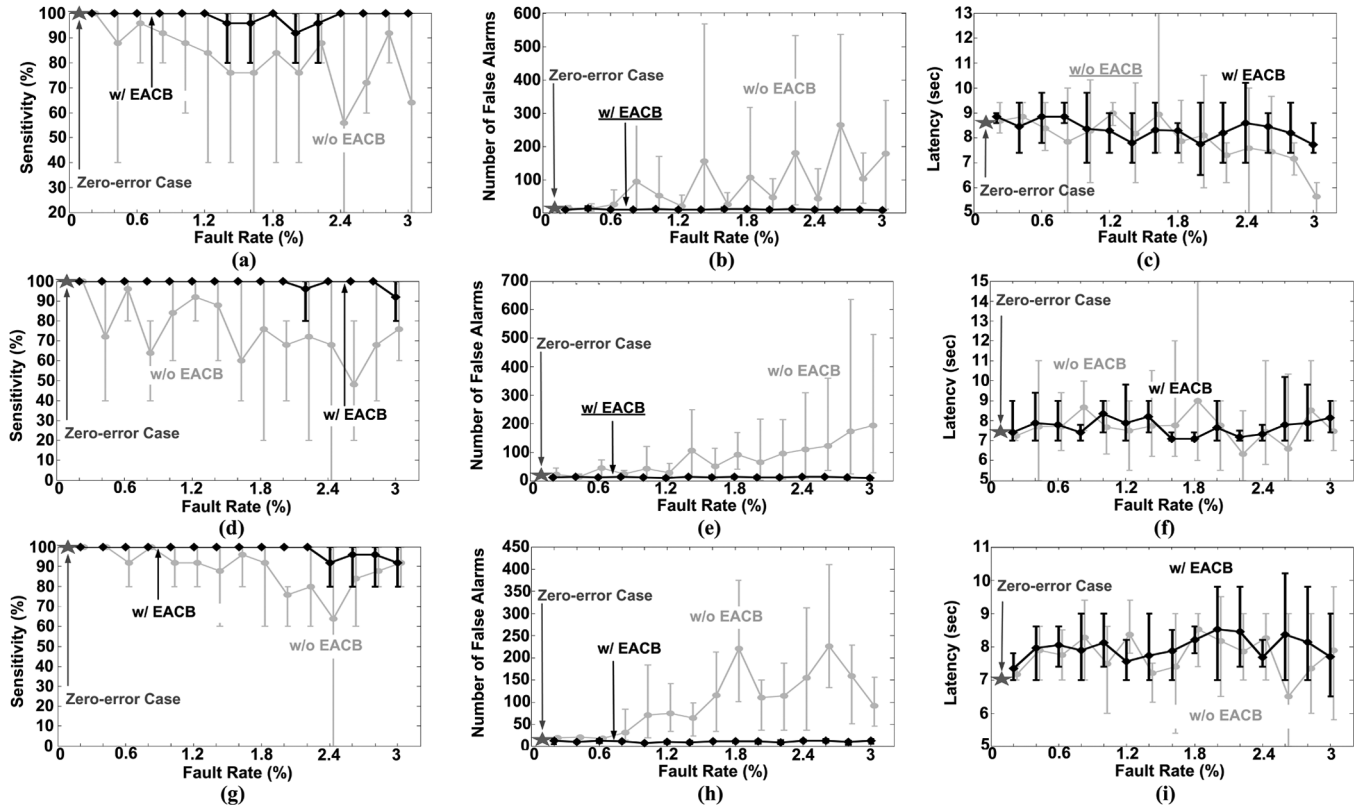


Fig. 8. Performance of the systems with respect to the fault rates, for the cases with and without EACB. Three types of weak classifiers are implemented (a)–(c) decision stumps (d)–(f) 4-node decision trees (g)–(i) 7-node decision trees. Five fault configurations are tested at each fault rate. In the bar plots, the markers indicate the average value over the five test cases, while the error bars denote maximum and minimum cases.

Weak Learner Type	Stump	4 nodes	7 nodes
Fault Rate Tolerable	> 3%	> 3%	> 3%
# Iterations T for Convergence	64	22	19
Equivalent NAND Gates per Tree	1169	1250	1575
Total Hardware Gates	74815	27493	28997
Fault-affected Hardware	82.0%	86.5%	89.4%

Fig. 9. Comparison of EACB system parameters for various weak-classifier choices (stumps, 4-node trees, and 7-node trees).

As we analyze below, the size of the trees impacts important parameters associated with the trainer and classifier hardware.

2) Classifier-Complexity Considerations: Fig. 9 shows the implementation details, comparing the three weak-classifier choices. Notably, since smaller trees result in weaker classifiers, somewhat more iterations (T) are required for performance convergence when we assume an unconstrained training-set size. In the hardware implementation used for the trees (Fig. 3), smaller trees result in only minor reduction of gate counts (since the implementation in Section III-B applies thresholding to all input features in the first stage, regardless of the number of nodes). Consequently, the increased iterations lead to higher total gate counts for the smaller trees. The voter hardware, which must be fault protected, consisting of a T-input signed adder. The adder hardware scales more than linearly with the number of iterations. As a result, smaller trees with more iterations lead to higher total ratio of fault-protected hardware. Nonetheless, across all cases EACB enables 82–89% of the classifier hardware to be fault affected. As we describe below, by enabling faults in the feature-extraction processor as well, the total fault-protected hardware is reduced to below 1% for the entire system.

3) Trainer-Complexity Considerations: While the classifier-complexity favors the use of larger trees, here we find that smaller trees are favored in terms of the trainer complexity. Accordingly, a tradeoff emerges between classifier complexity and trainer complexity. The training algorithm presented in Section III-C reduces the amount of trainer memory by 1) performing feature selection (based on a learner metric), and 2) enhancing generalization by acquiring a new data set for training at each iteration. Thus, increasing the weak-classifier iterations *or* increasing the data-set size improves generalization. The classifier-complexity analysis presented above (Fig. 9) does not consider constraints on the size of the data set for training each iteration. Fig. 10(a) shows that, in fact, the number of iterations required to achieve convergence depends on the constrained size of the data set- larger data sets enable fewer iterations thanks to greater generalization achieved at each iteration. In this scenario, we see that for the 7-, 4-, and 1-node trees, the number of iterations required for convergence tracks each other. As a proxy to energy, Fig. 10(b) shows the trainer clock cycles (corresponding to execution on the OpenMSP core), illustrating the relative complexity of training larger trees. Accordingly, while it is true that larger trees yield the potential for fewer iterations (and thus the classifier-complexity benefits described above), realizing this benefit would require a larger trainer memory and more clock cycles. As a point of reference, we show classifier and trainer parameters (namely, trainer memory and clock cycles) in Fig. 11, that yield a reasonable balance between the tradeoffs discussed.

4) EACB for Simultaneously Overcoming Feature Extractor and Classifier Faults: In this section, we present results when

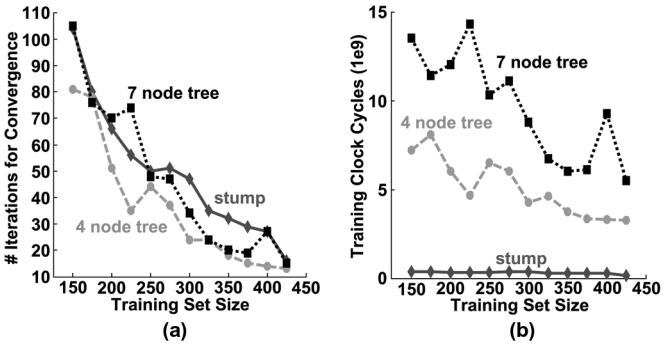


Fig. 10. On-line training system design trade-offs enabled by modifying weak classifier type (stumps, 4-node trees, or 7-node trees) and training data pool size.

Weak Learner Type	Stump	4 nodes	7 nodes
# Iterations T for Convergence	56	24	15
Trainer memory (kB)	7.2	10.4	13.6
Trainer clock cycles (x10^9)	0.33	2.8	2.6

Fig. 11. Comparison of trainer complexity for different sized trees (design parameters are chosen that yield a reasonable balance between system tradeoffs.

weak learner	affected nodes extractor	affected nodes class.	ratio fault protected	# iterations for convergence				
stump	0.011 %	2 %	4.4 %	76	11	22	15	28
4 nodes	0.011 %	2 %	1.2 %	8	6	14	4	18
7 nodes	0.011 %	2 %	0.9 %	13	9	5	18	6

Fig. 12. EACB system performance with faults injected in both the feature extractor and real-time classifier.

faults are injected in the entire system (including feature extractor and classifier). For experimental exploration, we inject stuck-at faults in the classifier at a fixed rate of 2% of the total hardware (generally regarded as a very high fault rate). Then we progressively increase the fault rate in the feature extractor. For each experiment, we measure the number of iterations required to reach convergence (i.e., minimal further improvement in system performance). As before, five fault configurations are tested at each fault rate to represent faults in different randomized locations of the circuit. From the experiments, we find that system performance is consistently maintained for feature-extractor fault rates of 0.011%. Fig. 12 summarizes the system parameters for all three tree topologies considered, where the fault rate is set to 0.011% and 2% for the feature extractor and classifier, respectively. We can see that the iteration required for convergence varies substantially for the different tree topologies, but follows a similar trend to that shown in Fig. 10. By applying the proposed techniques, the only component within the entire real-time system that must be fault protected is the final voter (Fig. 2), which accounts for 1–4% of the total circuit hardware.

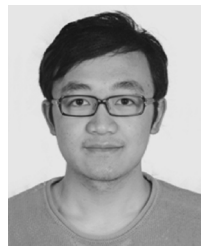
V. Conclusions

In this paper, we present the concept of error adaptive classifier boosting (EACB). EACB applies iterative training over fault-prone weak classifiers to achieve strong classification resilient against hardware faults. A system for EEG-based seizure detection is presented, implemented via an FPGA to enable controllable injection of faults. The system includes a fault-protected microcontroller (based on an OpenMSP) executing a proposed training algorithm, wherein the trainer memory required is reduced by 65× to 6.4 kB and the trainer energy is reduced by 10× (compared to state-of-the-art algorithms). Results from FPGA emulation show that the system is able to overcome high fault rates, affecting over 3% of the total circuit nodes in the classifier. The paper analyzes the complexity of the classifier and trainer with respect to the key design parameters.

References

[1] F. M. Khan, M. G. Arnold, and W. M. Pottenger, "Hardware-based support vector machine classification in logarithmic number systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2005, vol. 5, pp. 5154–5157.
[2] K. H. Lee and N. Verma, "A 1.2-0.55 V general-purpose biomedical processor with configurable machine-learning accelerators for high-order, patient-adaptive monitoring," in *Proc. IEEE ESSCIRC*, 2012, pp. 285–288.
[3] J. Park, J. Kwon, J. Oh, S. Lee, J.-Y. Kim, and H.-J. Yoo, "A 92-mW real-time traffic sign recognition system with robust illumination adaptation and support vector machine," *IEEE J. Solid-State Circuits*, vol. 47, no. 11, pp. 2711–2723, 2012.
[4] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," *Technol.@Intel Mag.*, vol. 9, no. 2, pp. 1–10, 2005.
[5] N. Verma, K. H. Lee, K. J. Jang, and A. Shoeb, "Enabling system-level platform resilience through embedded data-driven inference capabilities in electronic devices," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2012, pp. 5285–5288.
[6] Z. Wang, K. H. Lee, and N. Verma, "Hardware specialization in low-power sensing applications to address energy and resilience," *J. Signal Process. Syst.*, vol. 78, no. 1, pp. 49–62, 2015.
[7] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Proc. Design Autom. Conf.*, 2010, pp. 555–560.
[8] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra, "ERSA: Error resilient system architecture for probabilistic applications," in *Proc. Design, Autom., Test Eur. Conf. Exhib.*, 2010, pp. 1560–1565.
[9] K. H. Lee, S.-Y. Kung, and N. Verma, "Improving kernel-energy trade-offs for machine learning in implantable and wearable biomedical applications," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2011, pp. 1597–1600.
[10] Z. Wang, R. Schapire, and N. Verma, "Error-adaptive classifier boosting (EACB): Exploiting data-driven training for highly fault-tolerant hardware," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2014, pp. 3884–3888.
[11] M. Bohr, "The new era of scaling in an SoC world," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, 2009, pp. 23–28.
[12] J. W. McPherson, "Reliability challenges for 45 nm and beyond," in *Proc. Design Autom. Conf.*, 2006, pp. 176–181.
[13] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein, "Scaling, power, and the future of CMOS," in *IEDM Dig. Tech. Papers*, 2005, pp. 7–15.
[14] J. Zhang, A. Lin, N. Patil, H. Wei, L. Wei, H. Wong, and S. Mitra, "Robust digital VLSI using carbon nanotubes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 4, pp. 453–471, 2012.
[15] S. Datta, H. Liu, and V. Narayanan, "Tunnel FET technology: A reliability perspective," *Microelectron. Rel.*, vol. 54, no. 5, pp. 861–874, 2014.
[16] N. R. Shanbhag, S. Mitra, G. D. Veciana, M. Orshansky, R. Marculescu, J. Roychowdhury, D. Jones, and J. M. Rabaey, "The search for alternative computational paradigms," *IEEE Design Test Comput.*, vol. 25, no. 4, pp. 334–343, 2008.
[17] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, "Stochastic computation," in *Proc. Design Autom. Conf.*, 2010, pp. 859–864.
[18] N. Shanbhag, "Reliable and energy-efficient digital signal processing," in *Proc. Design Autom. Conf.*, 2002, pp. 830–835.
[19] E. P. Kim and N. R. Shanbhag, "Soft N-modular redundancy," *IEEE Trans. Comput.*, vol. 61, no. 3, pp. 323–336, 2012.
[20] G. V. Varatkar, S. Narayanan, N. R. Shanbhag, and D. Jones, "Sensor network-on-chip," in *Proc. IEEE Int. Syst. Chip Conf.*, 2007, pp. 1–4.
[21] Z. Wang, K. H. Lee, and N. Verma, "Overcoming computational errors in sensing platforms through embedded machine-learning kernels," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, to be published.
[22] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
[23] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. Cambridge, MA, USA: MIT Press, 2012.

[24] J. K. Bradley and R. E. Schapire, "FilterBoost: Regression and classification on large datasets," in *Proc. Adv. Neural Inf. Process. Syst. Conf.*, 2007, pp. 185–192.
[25] J. Kwong and A. P. Chandrakasan, "Variation-driven device sizing for minimum energy sub-threshold circuits," in *Proc. Int. Symp. Low Power Electron. Design*, 2006, pp. 8–13.
[26] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw, "RazorII: In situ error detection and correction for PVT and SER tolerance," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, 2009, vol. 44, no. 1, pp. 32–48.
[27] B. Wong, A. Mittal, Y. Cao, and G. W. Starr, *Nano-CMOS Circuit and Physical Design*. Hoboken, NJ, USA: Wiley, 2005.
[28] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Mach. Learn.*, vol. 40, no. 2, pp. 139–157, 2000.
[29] X. Li, L. Wang, and E. Sung, "Adaboost with svm-based component classifiers," *Eng. Appl. Artif. Intell.*, vol. 21, no. 5, pp. 785–795, 2008.
[30] H. Schwenk and Y. Bengio, "Boosting neural networks," *Neural Comput.*, vol. 12, no. 8, pp. 1869–1887, 2000.
[31] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2006, pp. 161–168.
[32] S. Lopez-Estrada and R. Cumplido, "Decision tree based FPGA-architecture for texture sea state classification," in *Proc. IEEE Int. Conf. Reconfigurable Comput. FPGA*, 2006, pp. 1–7.
[33] A. Bermak and D. Martinez, "A compact 3D VLSI classifier using bagging threshold network ensembles," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1097–1109, 2003.
[34] J. Struharik, "Implementing decision trees in hardware," in *Proc. IEEE Int. Symp. Intell. Syst. Informat.*, 2011, pp. 41–46.
[35] O. Girard, OpenMSP430, 2009.
[36] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.
[37] N. Verma, J. Kwong, and A. P. Chandrakasan, "Nanometer MOSFET variation in minimum energy subthreshold circuits," *IEEE Trans. Electron Devices*, vol. 55, no. 1, pp. 163–174, 2008.
[38] X. Shi, S. Hsu, F. Chen, M. Hsu, R. Socha, and M. Dusa, "Understanding the forbidden pitch phenomenon and assist feature placement," *Proc. SPIE*, vol. 4689, pp. 985–996, 2002.
[39] S. Ghosh and F. J. Ferguson, "Estimating detection probability of interconnect opens using stuck-at tests," in *Proc. 14th ACM Great Lakes Symp. VLSI*, 2004, pp. 254–259.
[40] A. H. Shoeb and J. V. Guttag, "Application of machine learning to epileptic seizure detection," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 975–982.
[41] A. H. Shoeb, "Application of machine learning to epileptic seizure onset detection and treatment," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 2009.
[42] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.



Zhuo Wang received his B.S. degree in Microelectronics from Peking University, Beijing, China, in 2011. He received his M.S. degree in 2013 and is currently a Ph.D. candidate in the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA. His research interest is on robust system design. More specifically, he is interested in designing robust VLSI systems for highly-energy-constrained applications, and how machine-learning techniques can be exploited, not only to model sensor signals, but also hardware faults affecting the platform.

Robert E. Schapire received his Sc.B. in math and computer science from Brown University, Providence, RI, USA, in 1986, and his S.M. (1988) and Ph.D. (1991) from the Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Ronald Rivest. After a short Postdoc at Harvard, he joined the technical staff at AT&T Labs (formerly AT&T Bell Laboratories) in 1991. Since 2002, he has been with the Computer Science Department at Princeton University, Princeton, NJ, USA, and was named the David M. Siegel '83 Professor in Computer Science in 2013. He joined Microsoft Research in New York City as a Principal Researcher in 2014 (on leave from Princeton). His awards include the 1991 ACM Doctoral Dissertation Award, the 2003 Gödel Prize, and the 2004 Kanelakkis Theory and Practice Award (both of the last two with Yoav Freund). He is a fellow of the AAAI, and a member of the National Academy of Engineering. His main research interest is in theoretical and applied machine learning, with particular emphasis on boosting and online learning.

Naveen Verma received the B.A.Sc. degree in electrical and computer engineering from the University of British Columbia, Vancouver, BC, Canada, in 2003 and the M.S. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2005 and 2009, respectively. Since July 2009, he has been an Assistant Professor of Electrical Engineering at Princeton University, Princeton, NJ, USA. His research focuses on ultra-low-power integrated circuits and systems with an emphasis on sensing applications. Of particular importance is the use of emerging devices for the creation of functionally diverse systems and the use of advanced signal-analysis frameworks for low-power inference over embedded signals. On the circuit level, his focus spans low-voltage digital logic and SRAMs, low-noise analog instrumentation and data-conversion, and integrated power management. Prof. Verma is co-recipient of 2008 ISSCC Jack Kilby Award for Outstanding Student Paper, and 2006 DAC/ISSCC Student Design Contest Award. He is recipient of the Alfred Rheinstein Junior Faculty Award at Princeton and the 2013 NSF CAREER award.