**EACB [2015]:**

This is not directly related to in memory computation, but the other paper that is about in-memory computation using 6T SRAM cells utilizes this approach to get around some non idealities involved in the circuits they implemented.

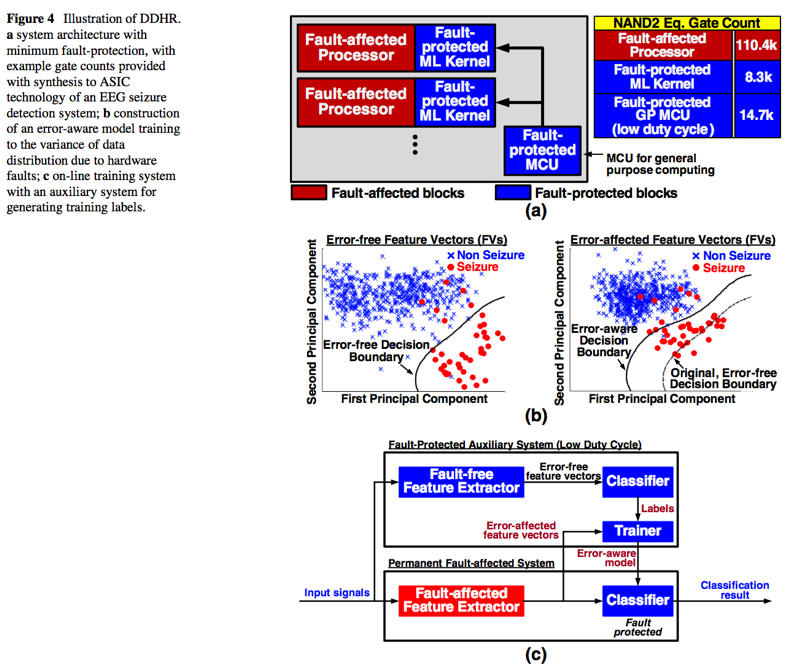
**In a nutshell:**

Dedicated hardware (i.e. ML accelerators) or in memory computation have been some approaches that people have recently adapted with regards to inference phase of machine learning for embedded sensing application (for example, seizure detection system). The broad idea in most of them is that they train their classifier somewhere outside of their chip with some co-optimization between hardware and algorithm in mind, and then load the learned weights onto their hardware to do the inference task. But if the hardware turns out to be faulty it will damage the accuracy of the inference task. This paper first talks about some of the approaches that have been used commonly to mitigate this issue, and of course, they propose their own method to deal with this which turns out to be more energy efficient and can tolerate more defects with less hardware required.

**Details of their proposal:**

**Previous solution(s):**

**Data-driven hardware resilience (DDHR):** The architecture of this work is demonstrated in Fig below part c. Implemented as a seizure detection system emulated on FPGA with programmable control over fault injections. This work only overcame the faults in the feature detection part. Basically the classifier will learn while the inputs have passed the faulty feature detection part while their labels are given by some master from outside. To make this possible they use an auxiliary fault protected system that will provide the true labels and may be energy intensive. But the rate at which this part operates is low enough that would not cause substantial energy overhead.  
Figure below (a) shows the parts which were fault protected and those that were not. It is roughly 20% of the total number of gates that have to be fault protected. The author claims that the problem of seizure detection does not have that much of a complexity in the ML kernel side, this is why the number of gates is much less than the rest of the system. But for more sophisticated problems this part will scale up and become the bottle neck. Hence EACB is about addressing the effect of non idealities and a way to deal with them in the classification section.



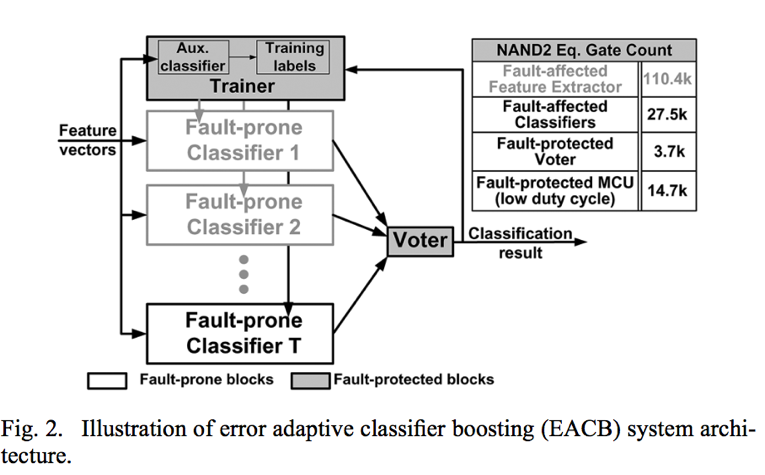
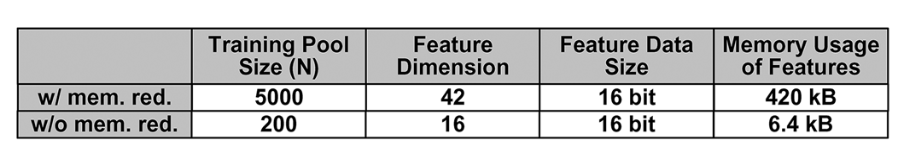
**AdaBoost:** it is an ML method that utilizes a collective vote of some weak classifiers to do the strong classification task. If we have a weak classifier that is trained but cannot make prediction precisely (but the accuracy is still more than 50% meaning that there is some correlation between true label and predicted label) we can train the next weak classifier to correct for the mistakes that the first classifier made and this can keep on going such that each classifier trained later is compensating for the mistakes that the previous classifier made, thus if we cast a vote out of their decision, in overall, the result gets more accurate.

**FilterBoost:** is another ML method that does not require a lot of data to be available prior training. It will learn sequentially and refine the ML model in an iterative training of the weak classifiers, as the new data comes in.

**Error adaptive classifier boosting (EACB):**

Figure below shows the overall architecture. It uses T parallel fault-prone classifiers each of them giving their classification result to a fault-free voter. It is important to note that the voter and the top level trainer should still be fault free, but they comprise a small portion of the overall number of devices. This paper uses some sort of adaboost approach to train this classifier, meaning that if some classifier is faulty another classifier may be able to compensate for that. It turns out the structure of each classifier is important per se but they utilize decision tree based structure which they argue is beneficial because of simple control path. The reason is that if any defect happens in the control path it could be the case that ECAB won’t work, since it can obliterate the correlation of the true labels and the predictions.

This work uses some algorithm in training which is a little bit involved, but what it basically does is that it leverages a combination of Adaboost and Filterboost to bring down the number of samples needed for training. It also uses some feature dimensionality reduction algorithm that allows it to use a reduced dimension version of feature vectors for learning, further bringing down the need for memory capacity. Table below illustrates how much memory they need with and without using this algorithm.



They demonstrated this on an FPGA, and they claim 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.

**Challenges and vague points in their proposal:**

1. How are they even planning to build a fault free system on top of this faulty system? What are the costs associated with this?  
   This is answered in section III.A
2. What kind of faults are they assuming to overcome?? Faults that are constant through operation of the circuit. But things like lowering the VDD due to some reason or variation of temperature and random variations across time of operation is not studied. Maybe we can implement this in asic flow and address those issues??

**In memory computation using 6T SRAM cells:**

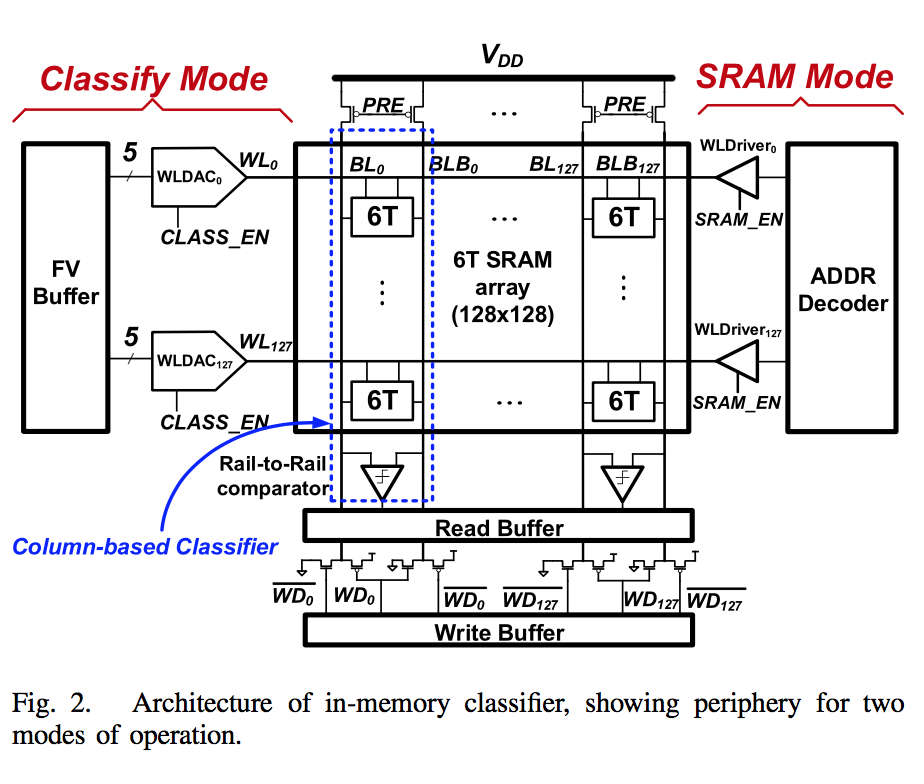
**In a nutshell:**

This paper uses the standard 6T SRAM array structure to build an inference machine which does not need to fetch data out of memory and do the computation, this is specifically useful in embedded sensing applications where power is a crucial element, and driving data in an out of memory the most power hungry part in inference task. They demonstrate their work for hand written digit classification (on MNIST data set) and claim to have 113x more power efficiency than discrete SRAM cell and standard learning algorithm. In order to make this happen they leverage their optimized learning algorithm. They further claim that if a discrete SRAM and their algorithm is used they still get and efficiency of 13x higher than the system just described.

**Details of their proposal:**

**System Overview:**

Accessing memory has shown to be more than 200x more energy hungry than a multiply operation in 45nm. In the demonstrated system, computation is performed in-place by the bit cells of an SRAM, avoiding energy-intensive accesses. This faces two key challenges: 1) the constrained structure of standard 6T arrays limits the computations possible; and 2) circuit non-idealities, especially high variability in bit cells, degrades the quality of outputs. To get rid of these two issues they propose the following structure:



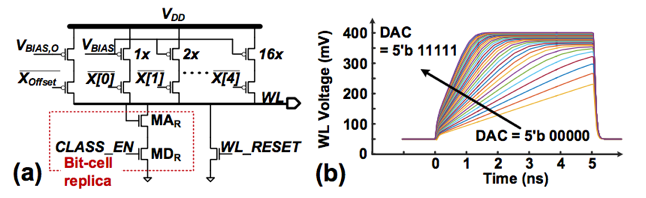
This system operates at two modes: (1) SRAM mode which is the standard SRAM behavior with read/write access (2) Classify mode in which the word lines are all driven by the analog representation of the input feature vector. (multiple row wordlines will be driven in contrast to the SRAM mode) In a simple linear classifier in ML the goal is to compute something in the form of where is the weights that are learned throughout the learning phase, and sgn is a decision making function. ML community has shown that if s’ are constrained to values of there could be no loss in accuracy, if trained properly. This representation of weights make life a lot easier in circuit implementation.

Each of the columns is a weak classifier and EACB is used to overcome the fitting errors due to being a weak classifier as well as the non-idealities in the circuit.

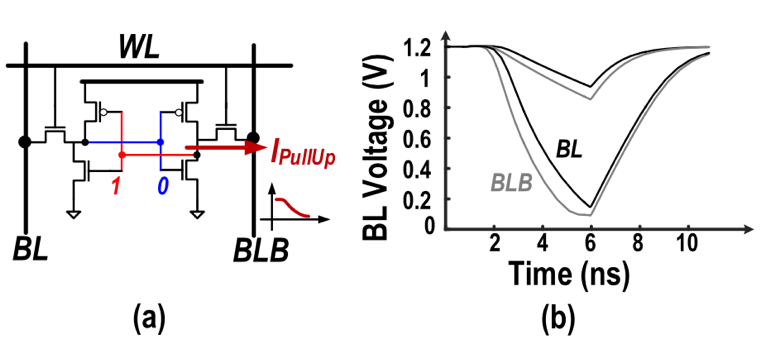
First BL/BLB’s are precharged. Then all the WL’s are driven at once to analog voltages corresponding to the elements of x (i.e., the features); this is done via the peripheral WLDACs. Thus, each bit cell pulls a current modulated by its WL voltage from either BL or BLB, depending on its stored state. This approximates multiplication by +/-1. The bit-cell currents from the column then add together, as in an inner product, discharging BL or BLB. Finally, a comparator provides sign thresholding of the differential signal.

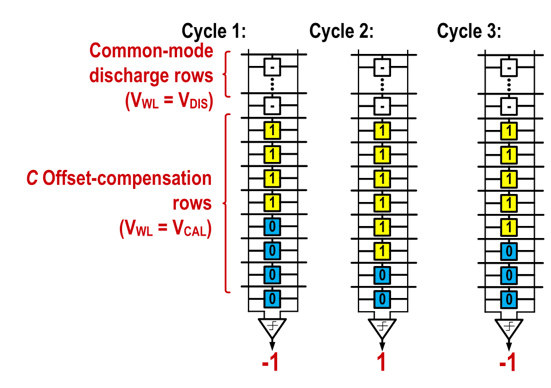
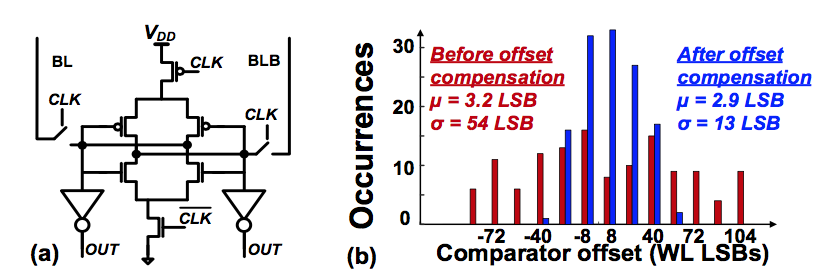
Circuit Implementation:

In SRAM mode every thing is typically designed.

Fig below shows the WLDAC circuit and simulated output (WL) waveform. Digital features are provided as 5 bits X[4:0], to select binary-weighted PMOS current sources; an additional PMOS current source is included to enhance the linearity of the charge drawn by a bit cell. The current from the PMOS sources is used to bias a replica of the bit cell, consisting of a switch (MDR) and diode-connected transistor (MAR), which drives the WL. Thus, all bit cells in the row provide a modulated current IBC (i.e., scaled by the replica sizing ratio). Note that with all WLs driven this way, a potentially large number of bit cells drive BL/BLB at the same time (128 in the prototype). To not saturate discharge of BL/BLB capacitances, 𝑄𝐵𝐶 of each is designed to be low <10fC, requiring WL voltages <0.4V.

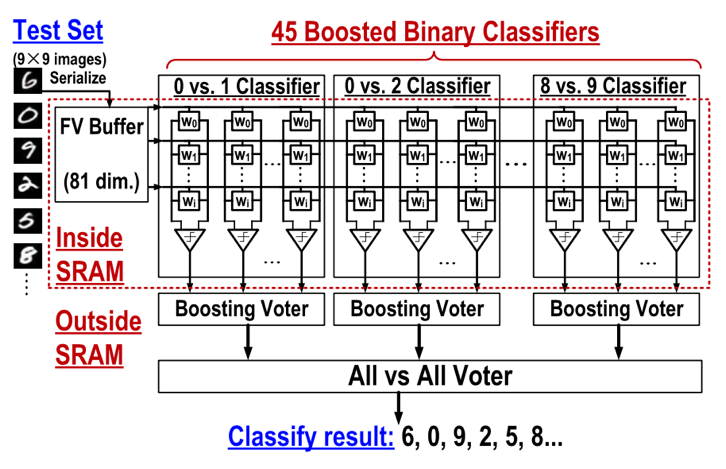
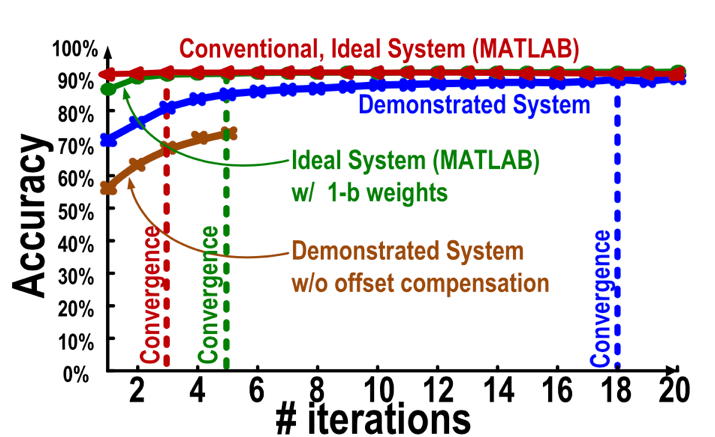
There is another source of error that needs to be addressed. There can be large swings on BL/BLB, and this will cause two issues: 1) the currents from all cells pulling down are reduced due to decreasing vds across their access and driver transistors and 2) cells not pulling down begin to pull up, albeit through a weakly biased Nmos. Consequently, as shown in the simulations of Fig below (b), a lower common-mode voltage causes the BL/BLB differential voltage to be compressed. However, if comparator offset is small this would not change the classification result. Furthermore, not only do they compensate the offset by some mean, they use EACB to mitigate this variation as well.



This system utilizes a rail to rail comparator. Also a clever way of offset cancelation using the 6T SRAM array structure itself is employed. Comparator offset is compensated by allocating some SRAM rows to an offset-compensation set. All rows in this set receive the same WLDAC code, resulting in the same nominal Qbc. During offset compensation, other rows in the array are selected to discharge BL/BLB by a nominally equal amount. Preferably, various random selections of rows are employed and averaged, to mitigate bit-cell variability. Then, for all columns, data is written to bit cells in the offset-compensation set to equalize the probability of either comparator. This is done in a binary-search manner for rapid convergence. For the demonstration, 32 rows are designated to the offset-compensation set, with WLDAC code of 16. Fig Below left shows the measured pre-/post-compensated offset over 128 comparators, in terms of the difference in WLDAC code for BL/BLB discharge at the trip point.

**The results:**

The demonstrated system achieves the ideal accuracy after 18 iterations, meaning that while using EACB, it needs 18 times to go over all weak classifiers to achieve the accuracy needed. Both curves with and without compensation of comparator are presented to illustrate the impact of compensation of comparators to some extend. In the end they provide some trade off analysis which might be useful…

****

Challenges they still have: I did not fully understand the compensation method??