Enhancing SRAM-Based PUF Reliability Through Machine Learning-Aided Calibration Techniques

Kuheli Pratihar, *Student Member, IEEE,* Soumi Chatterjee, *Student Member, IEEE,*
Rajat Subhra Chakraborty, *Senior Member, IEEE,* and Debdeep Mukhopadhyay, *Senior Member, IEEE.*

***Abstract*—SRAM-based physically unclonable functions (PUFs) utilize unpredictable start-up values (SUVs) for key generation, making them widely adopted in cryptographic systems. This unpredictability in SUVs is accompanied by device noise that escalates with process-voltage-temperature (PVT) variations, resulting in significant deviations from the golden response collected at ambient conditions, thereby increasing the bit-error-rate (BER) of the PUF responses. To reduce this high ($\geq 15\%$) BER, either an involved error correcting code (ECC) circuitry with significant overhead is required, or more helper information needs to be generated at varying operating conditions, resulting in increased information leakage. We address this issue by proposing the first reported application of machine learning to re-calibrate the responses by predicting the golden responses of the SRAM-PUF at different operating conditions with high accuracy. Our re-calibration technique is based on a novel collective decision that involves observing the neighborhood cells of the SRAM-PUF, as opposed to the traditional single-cell approach. By leveraging a memory map exhibiting a high correlation in ambient reliability amongst neighboring cells, we indirectly use the physical co-location of SRAM cells to assist neighborhood error prediction. It leads to efficient post-processing for SRAM-PUFs by using helper data generated at ambient conditions only while employing a fixed ECC designed for the same. Subsequently, to justify our claims and validate the efficacy of our proposed methodology, we demonstrate extensive experimentation results over multiple SRAM-PUF instances implemented on the Arduino UNO (an 8-bit microcontroller unit) and its scaled-up version, the Arduino Zero (a 32-bit microcontroller unit) boards, by varying supply voltages from 3.8 to 6.2 V and 7 to 12 V respectively, and temperature from -25° to 70° C in both cases. Our observations show a vast drop in BER from 17.02% to $\approx 1\%$. Although worst-case conditions with both voltage and temperature variations at play resulted in a BER of 20%, using our proposed approach reduces it to $\approx$ 1-2%, in turn demonstrating the high efficacy of our scheme.**

***Index Terms*—Physically unclonable functions (PUFs), static random access memory (SRAM), helper data, min-entropy, machine learning (ML), transfer learning, PVT variations.**

I. Introduction

In the ever-expanding realm of interconnected devices, Static Random Access Memory based Physically Unclonable Functions (SRAM-PUFs) are a pertinent solution for low-cost key generation in FPGAs, microcontrollers, and ASICs [1]. Although there exist other PUF architectures like the oxide-rupture PUFs [2], DRAM PUFs [3], at the same time, SRAM-PUFs have withstood the test of time and are a mature and viable security component that has achieved widespread adoption in commercial products. As per terminologies accepted in the PUF community, SRAM-PUFs popularly operate as what are called as weak PUFs, and are used to generate secret keys for crypto operations.

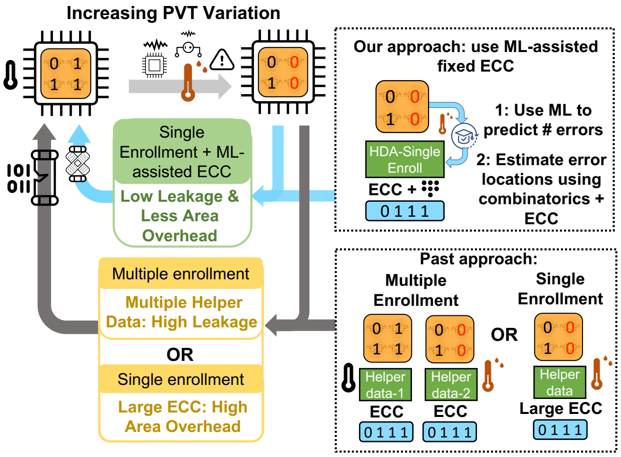

Increasing PVT Variation
0 1
1 1
0 0
1 0
101
011
Single Enrollment + ML-assisted ECC
Low Leakage & Less Area Overhead
Our approach: use ML-assisted fixed ECC
1: Use ML to predict # errors
HDA-Single Enroll
ECC +
0 1 1 1
2: Estimate error locations using combinatorics + ECC
Multiple enrollment
Multiple Helper Data: High Leakage
OR
Single enrollment
Large ECC: High Area Overhead
Past approach:
Multiple Enrollment
Single Enrollment
OR
Helper data-1
Helper data-2
Helper data
ECC
ECC
Large ECC
0 1 1 1
0 1 1 1
0 1 1 1

Fig. 1: Our proposed approach of using ML-assisted ECC for correcting PVT variation-induced errors. It achieves low leakage and less hardware footprint in SRAM-PUFs than conventional approaches.

However, the reliability of SRAM-PUFs is a significant concern that affects the authenticity of the established keys when the devices are subjected to process, voltage, and temperature [4], [5] (PVT) variations. The presence of this unreliability introduces a bit-error rate (BER) in the responses of the SRAM-PUF. This increase in BER directly contributes to an increased key-error rate (KER), which can jeopardize any subsequent cryptographic operation. Classical techniques for handling the unreliability of SRAM-PUFs typically involve using helper data algorithms (HDAs) such as Secure Sketch [6] together with error correcting codes (ECC). However, it has been reported in the literature that the helper information, while enhancing reliability, reduces the entropy of the keying material [7]. This has motivated the search for alternative strategies to trade off the leakage and reliability obtained effectively. In conventional applications of SRAM-PUFs with scenarios where the temperature varies, it is imperative to use multiple helper information during enrollment [8]. Alternatively, the helper data collected at ambient conditions

Manuscript received XX August 2024; accepted XX August 2024. Date of publication YY October 2024; date of current version ZZ October 2024. This article will be presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2024 and appears as part of the ESWEEK-TCAD special issue. This work was supported in part by Qualcomm (Reference #IND-492686). (*Corresponding author: Kuheli Pratihar*)

Kuheli Pratihar, Soumi Chatterjee, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay are with the Department of Computer Science and Engineering, Indian Institue of Technology, Kharagpur, India.

can be used in conjunction with an ECC circuit capable of correcting the many errors introduced due to more aggressive PVT variations. Both these approaches suffer from their respective demerits. The former method of using multiple helper information increases entropy loss. At the same time, the latter requires the design of an involved ECC circuitry [9], which not only has an adverse effect on the resource requirements but also suffers from higher leakage via helper information [10] (see Fig. 1).

In this paper, we investigate a machine learning (ML) based post-processing methodology that works on enrolled helper information collected only at ambient conditions. To the best of our knowledge, we propose the first post-processing scheme using ML-based models to predict the varying reference response of the SRAM-PUF at extreme operating conditions. In our setup, the PUF is enrolled in the ambient condition with the golden response, and thereafter, the reference response in various operating conditions is re-calibrated using our proposed ML model. Our technique can predict the number of errors in the golden response at adverse conditions with high accuracy and subsequently uses the enrolled helper information at ambient conditions to assist the fixed ECC circuitry to correct the acquired noisy response. The proposed methodology is built in two distinct steps.

1) First, we train a model that predicts the expected number of erroneous bit positions in the SRAM memory array due to changes in the operating conditions. To put it simply, for a given PUF, we train our ML model continually with voltage or temperature variations and use the trained parameter to re-calibrate the PUF response with a fixed ECC. This phase of the learning is referred to as *continual learning* in our following exposition. Next, we explore the transferability of our models across devices by training on multiple reference devices but extending the same to a new device-under-test (DUT) using only the reliability information at the ambient condition. This phase of learning, which we shall be referring to as *transfer learning*, is an important technique to demonstrate that the ML-based error correction is feasible on new boards without needing to re-train the models from scratch.
2) Thereafter, on obtaining the expected number of erroneous bit positions, we propose a novel combinatorial approach that takes into account the number of errors predicted in the neighborhood of a target cell and helps us make a better collective decision while deriving the exact error locations in the memory grid.

Finally, the predicted error locations are used to re-calibrate the reference response at varying operating conditions. Under no circumstances does the adversary have access to raw responses from SRAM-PUF, a category of weak-PUF constructions that do not reveal challenge-response pairs (CRPs) [11]. Our proposed technique following the assumptions of the helper data-based secure sketch algorithm [10] does not leak any information about the derived PUF response, and hence the secret key.

We perform extensive experimentation to validate the efficacy of our scheme by creating *test-beds to perform wide voltage fluctuations* from 3.8 to 6.2 volts across eight, and temperature variations from $-25°$ to $70°$C across ten Arduino UNO boards respectively. Additionally, to evaluate the scalability of our proposed methodology to larger SRAM memories, we extended our experiments to another 18 instances of the Arduino Zero board with 32 kB SRAM as opposed to 2 kB SRAM in the UNO board. Each of these boards houses an SRAM-PUF instance. The response is corrected with a concatenated code with Repetition[3,1] and Reed-Muller code (RM[2,5] for UNO and RM[2,4] for Zero) capable of only correcting errors in the neighborhood of the ambient condition and executed on a connected Arduino Nano 33 BLE Sense board. The Nano 33 BLE Sense board executes the ML-based re-calibration code, the combinatorial unit, and the ECC with fixed parameters. This entire implementation utilizes only 6.5K instructions on a 32-bit ARM Cortex-M4 CPU running at 64 MHz, incurring a latency of 672 clock cycles and an energy consumption of 2.3 μJ/decode for key generation. We demonstrate that the proposed ML-based re-calibration effectively reduces the BER from a whopping 17.02% to $\approx$ 1%. Both of these BERs translate to a desirable key-error rate (KER) of $< 10^{-6}$ for generating a stable 128-bit cryptographic key [5]. Furthermore, the high-entropy claim of our proposed work is demonstrated by an empirical estimation of min-entropy loss using a simple BCH[15,11,1]. We also observe a consistent *drop in the min-entropy loss across all temperatures* using our ML-assisted ECC when compared to standard ECC with multiple enrollments achieving a minimum (maximum) of 1.7 (2.5) bits compared to 2.4 (2.7) bits across $-20°$C ($70°$C).

Advantages over prior works: Although the previously proposed circuit methods in the literature are aimed at achieving a low BER under extreme operating conditions, they often relied on increasing the PUF reliability at ambient conditions by constructing a robust PUF cell [12], [13] or by increasing PUF reliability at ambient conditions using majority voting and accelerated aging [14]. However, these on-chip specialized circuit modifications exhibit limitations in their applicability, as they may not generalize well to all implementations of SRAM-PUFs [15]–[18]. Although ML has been used to identify and classify noisy SRAM-PUF [19] and DRAM-PUF [20] responses by processing the features in the 2D binary image obtained by transforming the bitstream, its application for error correction in SRAM-PUFs has not been explored. Our proposed ML-based post-processing scheme re-calibrates the PUF responses over a wide range of operating conditions while the ECC scheme and its parameters (implemented in hardware/software) remain unchanged. We comprehensively compare state-of-the-art helper data algorithms using standard ECC and soft-decoding techniques in Section IV-D.

To summarize, the main contributions of this paper are:

- **ML model for PUF re-calibration:** We present for the first time an ML model for re-calibration of SRAM-PUF's golden response at varying operating conditions. We believe this model may be of independent interest.
  - Our ML training method involves continually learning errors in the presence of temperature/voltage variations, which is improved upon using transfer learning to make

the trained model time-independent and agnostic across SRAM-PUF instances from the same device family.

- **Novel post-processing methodology:** We propose a post-processing methodology for SRAM-PUFs which works in conjunction with helper data generated at the ambient condition and a fixed ECC capable of only correcting a few bits as in the nominal condition.
- **Extensive experimental evaluation:** Finally, we validate the adequacy of our methodology by performing extensive experiments on 20 SRAM-PUF instances from two device families: the Arduino UNO (an 8-bit microcontroller unit with 2 KB SRAM) and the Arduino Zero (a 32-bit microcontroller unit with 32 KB SRAM). We obtained promising results on the SRAM-PUF instances operating over a wide range of temperature and voltage fluctuations, both in terms of resources when compared to a variable ECC circuit and with respect to entropy leakage when multiple helper information is utilized.

The artifacts to replicate the results of this work are available at GitHub: https://github.com/SEAL-IIT-KGP/The-SRAM-PUF-Calibration-Chronicles. The rest of the paper is organized as follows. Sections II and III detail our proposed two-step methodology for improving the effective reliability of SRAM-PUFs by re-calibrating the PUF responses. The experimental setup, results of our proposed methodology, and comparison with state-of-the-art approaches are presented in Section IV. Thereafter, Section V throws light on some interesting perspectives with Section VI concluding the paper.

II. Proposed Approach (Step-1): Predicting Number of Window Errors using ML

The challenge of predicting bit-flips at a given location in SRAM-PUFs arises from the limited information available for a single SRAM cell to understand its error dependence on operating conditions. The reliability loss can be attributed to fundamental effects like Negative Bias Temperature Instability (NBTI) [21], which is difficult to model without the knowledge of device-level parameters of the SRAM cell. Fortunately, there exists spatial correlation (neighborhood analysis) in SRAM-PUF bits, and it is a well-studied area in the literature [22]. This neighborhood analysis revealed that the most stable bits depend on the stability of their neighbors, i.e., those surrounded by other stable bits flip less over time. This reported information motivated us to take a closer look into the spatial correlation in the SRAM-PUF cells with respect to errors arising from environmental variations, i.e., temperature and voltage fluctuations.

Spatial correlations within SRAM arrays reveal that cells in close physical proximity exhibit similar reliabilities. However, the logical addressing in microcontrollers' embedded SRAMs doesn't directly mirror this physical arrangement. By applying insights from past reverse engineering efforts, specifically, those outlined in [23], we deduce the physical locations corresponding to logical addresses. For the ATMega328P, leveraging architectural specifics—like its four SRAM banks of 64 × 64 cells and 8 : 1 column multiplexers—helps us develop an accurate memory map with high neighborhood correlation in reliabilities. This mapping shows that a physical row in a bank correlates to eight consecutive logical addresses, each yielding an 8-bit response, equating to 512 logical addresses per bank. Through this memory map, we observe an increase in the average number of unstable cells within a 5 × 5 window surrounding an unstable cell from 2.36[1] to 2.83, demonstrating a significant spatial correlation. Similarly, for the SAMD21 microcontroller in ARM Cortex M0+, we adjust the ATMega328P's memory map to accommodate 16 SRAM banks, each with four 64 × 64 sub-banks. Here, each physical row maps to eight logical addresses but spaced out by four, each producing a 32-bit response. This adapted mapping increases the average count of unstable cells in a 5×5 window around an unstable cell from 13.58[1] to 14.62, further affirming the presence of location-dependent reliability correlations.

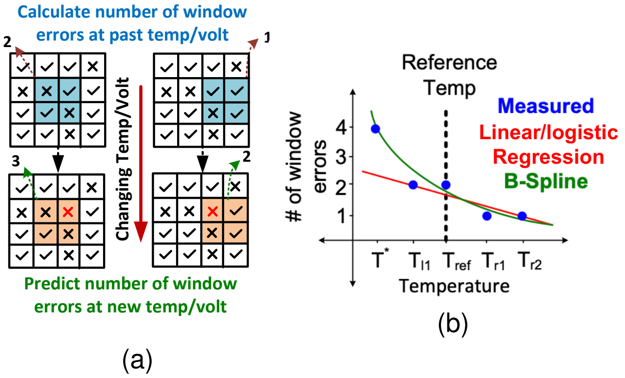


Fig. 2: (a) Overlapping 2 × 2 windows in SRAM-PUFs, and prediction of the number of window errors using (b) ML models (both linear and spline) for finding the best-fit curve.

Leveraging the existence of spatial correlations in SRAM-PUFs and the reliability-based memory map, we propose predicting the number of errors in an $(s \times t)$-sized window rather than focusing on a single cell. Next, we employ small windows, i.e., choose relatively low values of s and t, to accurately capture the neighborhood dependence, as well as to have a manageable-sized search space to locate precise error locations. From our empirical analysis (described later in Section II-D), we found larger window sizes are less effective in finding precise error locations, as the search space expands exponentially and the dependence of SRAM cells within the same window but far from the reference cell diminishes.

In our methodology, we first partition the entire $N \times N$ SRAM-PUF array into $s \times t$ sub-arrays for chosen values of s, t, and predict the number of errors in each sub-array at a given temperature/ voltage (as the case may be), using the PUF response at different temperatures/voltages and devices. Figure 2a shows overlapping windows of size 2×2, where ✗ and correct positions denote errors are denoted by ✓. Next, we present an ML approach to predict the number of errors in a window. This encompasses the first step of our post-processing scheme. The variables required for ML training and inference are detailed in Table I. The second step, as entailed in Section III, would be to develop a novel combinatorial approach to use this information of the number of erroneous

[1] average number of unstable cells within a 5 × 5 window surrounding an unstable cell is computed using a random memory map

	Notation	Description	Typical Values
Operating Condition	u	temperature or voltage	$[-25°, 70°]$ C, $[3.8, 6.2]$ V
	u^*	target u for prediction	
	$\mathbf{U}$	set of all u's	
	u_{ref}	u value at ambient condition	$24°$ C, 5 V
	u_{step}	resolution of temperature/voltage change	$2.5°$ C, 0.2 V
SRAM-PUF	p	device index	-
	p_t	target device during inference	-
	$\mathbf{P}$	set of all p's	-
	$\mathbf{c}$	challenge (address)	-
	$\mathbf{C}_w$	set of $\mathbf{c}$'s in a window w	-
	$\text{PUF}_{p,u}(\mathbf{c})$	response for p^{th} device at u and challenge $\mathbf{c}$	$\{0, 1\}$
	nMeas	number of PUF measurements	15
	$\text{M}(\text{PUF}_{p,u}(\mathbf{c}))$	majority voted $\text{PUF}_{p,u}(\mathbf{c})$	$\{0, 1\}$
	$\text{Rel}_p(\mathbf{c})$	response reliability corresponding to $\mathbf{c}$ for device p	$[50\%, 100\%]$
Windowing Condition	w	window index	-
	$s \times t$	window dimension with s rows & t columns	2×2
	D	total number of windows	8K
	$\text{Err}_w(u)$	total number of measured errors in w at u	$\{0, \dots, 4\}$
B-Spline Inputs	d_s	degree of piecewise polynomials	3
	M	number of knots	10
	L	number of control points $d_s + M + 1$	14
B-Spline Model Parameter	$\mathbf{o}_i$	i^{th} control point $\mathbf{o}$	-
	$N_{i,d_s}(u)$	i^{th} basis function $N(u)$ of degree d_s	-
	$f_{w,l}(u)$	lower model for w $\forall u < u_{\text{ref}}$	-
	$f_{w,u}(u)$	upper model for w $\forall u > u_{\text{ref}}$	-
	$f_w(u)$	complete model for w $\forall u$	-
Spline Output	$\text{PErr}_w(u)$	total number of predicted errors in w at u	$\{0, \dots, 4\}$
Learning Continual	τ	time instance of measuring PUF response	-
	τ_k	training time instance	-
	τ_l	test time instance	-
	u_k	operating condition at τ_k	$[-25°, 70°]$ C, $[3.8, 6.2]$ V
	u_l	operating condition at τ_l	
	τ_{ref}	time instance at ambient condition	-
	τ_{step}	time resolution for temporal measurements	30 minutes

Table I: Notations and typical values for ML training and inference.

cells in the window and reliability information to predict the exact error locations. *This information of the neighboring cells in a window helps us make better collective decisions for the fixed ECC to correct errors much beyond its capability at extreme operating conditions.*

A. Model for Predicting the Number of Errors in a Window

The number of errors in an $s \times t$ window ranges from zero to a maximum of $s \cdot t$. The increasing trends of these errors with changing operating conditions follow a non-linear relationship, which cannot be captured accurately with linear/logistic regression techniques, as shown in Fig. 2b. On the other hand, standard polynomial regression techniques using the least squares method suffer from oscillatory behavior at the end-points [24]. This oscillatory effect worsens for higher-degree polynomials, making them ill-suited to predict window errors at new operating conditions. To overcome these drawbacks, we resort to using *B-splines* or *basis splines*, a well-established approach [25] for curve fitting on experimental data to capture the non-linearity in the number of window errors with respect to the fine-grained change in the operating conditions. B-splines use piecewise polynomials of degree d_s as basis functions, providing modeling flexibility and high accuracy for lower degrees (3-to-4) compared to higher-degree polynomial regression. Furthermore, they parsimoniously adapt to the number of knots - joining point for piecewise polynomials, preventing the oscillatory behavior at boundaries.

Model Construction: The shape of the curve is determined by L control points, where $L = d_s + M + 1$ with M being the number of knots and d_s being the polynomial degree. Our model $f_w(u)$ is formulated as a linear combination of control points $\mathbf{o}_i$ and basis function $N_{i,d_s}(u)$ and is denoted as $f_w(u) = \sum_{i=0}^{L-1} \mathbf{o}_i N_{i,d_s}(u); L > d_s - 1$. The basis functions $N_{i,d_s}(u)$ are piece-wise polynomials of degree d_s, each of which is defined over the domain of a knot vector $\mathbf{U}_k = \{u_{k,1}, u_{k,2}, \dots, u_{k,M}\}$. All basis functions have the same degree and continuity properties but individually model the number of errors in each knot span $u_{k,i} < u < u_{k,i+1}$. The control points computed based on the knots determine the overall shape of the curve along with the basis functions.

Algorithm 1: PUF specific continual learning

```
Training until $\tau_k$ where, $\tau_k > \tau_{\text{ref}}$:
Initialize $\tau_k = \tau_{\text{ref}}$, $f_w(u) = 0 \;\forall\; u \in \mathbf{U}$
while (!stop) do
    $w = 0$
    while $w < D$ do
        if ($u_k = u_{\text{ref}}$):$f_w(u) = 0$;
        else: /* Update continually */
        $\text{Err}_w(u_k) = \sum_{\mathbf{c} \in \mathbf{C}_w} |\text{PUF}_{n,u_k}(\mathbf{c}) - \text{M}(\text{PUF}_{n,u_{\text{ref}}}(\mathbf{c}))|$
        $f_{w,l}(u)$ = model.fit($[u, \text{Err}_w(u)] \;\forall\; u_k < u < u_{\text{ref}}$)
        $f_{w,u}(u)$ = model.fit($[u, \text{Err}_w(u)] \;\forall\; u_k > u > u_{\text{ref}}$)
        /* aggregate model for either side of $u_{\text{ref}}$ */
        $f_w(u) = (f_{w,l}(u), f_{w,u}(u)) \;\;\forall\; u \in U$
        $w = w + 1$;
    $\tau_k = \tau_k + \tau_{\text{step}}$
Inference at time $\tau_l > \tau_k$ with temperature/ voltage $u_l^*$:
$f_w(u_l^*)$ = model.predict($u_l^*$)
Evaluate $\text{PErr}_w(u_l^*)$
```

Next, we train the B-spline model $f_w(u)$ for w^{th} window as a function of temperature/ voltage u using training data with the commonly used least square loss function. During the testing phase, the model predicts the number of errors at a target temperature/ voltage condition u^* for the w^{th}, $s \times t$ window and is given by $\text{PErr}_w(u^*)$ as:

$$\text{PErr}_w(u^*) = \begin{cases} 0, & f_w(u^*) < 0.5 \\ r, & f_w(u^*) \geq r - 0.5, < r + 0.5 \\ & \forall\; r \in \{1, \dots, st - 1\} \\ st, & f_w(u^*) \geq st - 0.5 \end{cases} . \quad (1)$$

Using L, d_s, M as independent inputs to the B-spline, the model parameters $\mathbf{o}_i, N_{i,d_s}(u), f_w(u)$ are learnt during training, to obtain $\text{PErr}_w(u^*)$ as output as shown in Table I.

B. Continual Learning (CL): Specific to the device under test

The optimal B-spline parameters predicting the number of window errors are found using continuous ML training on a target device under test p_t. The training dataset comprises the PUF challenge $\mathbf{c}$, temperature/ voltage $u \in \mathbf{U}$ as its feature set, and the measured number of window errors $\text{Err}_w(u_k)$ with respect to the ambient condition u_{ref} as its label. We reset our model at the ambient condition (time τ_{ref}) and continually update it at different training time instances τ_k where $\tau_k > \tau_{\text{ref}}$. The PUF golden response at ambient condition (u_{ref}) is used as the reference to measure the number of window errors, $\text{Err}_w(u_k)$ for each window w, where u_k is the operating condition at τ_k. This is given by:

$$\text{M}(\text{PUF}_{p,u_{\text{ref}}}(\mathbf{c})) = \text{Mode}\left([\text{PUF}_{p,u_{\text{ref}}}(\mathbf{c})]^{\text{nMeas}}\right)$$

$$\text{Err}_w(u_k) = \sum_{\mathbf{c} \in \mathbf{C}_w} |\text{M}(\text{PUF}_{p,u_k}(\mathbf{c})) - \text{M}(\text{PUF}_{p,u_{\text{ref}}}(\mathbf{c}))| \quad (2)$$

where, $\mathbf{C}_w$ is the challenge (address) set in window w, nMeas is the number of measurements, and $\text{M}(\text{PUF}_{p,u_k}(\mathbf{c}))$ is the majority-voted response at time τ_k with operating condition u_k for challenge $\mathbf{c}$ on device p.

Specifically, the training dataset for temperature/ voltage at τ_k comprises of all readings from $u_k < u < u_{\text{ref}}$ or $u_{\text{ref}} <$

Algorithm 2: Transfer learning ML model from reference devices to device-under-test of the same family

```
/* training once during enrollment;                                    */
Training for all $u \in [u_{\mathbf{min}}, u_{\mathbf{max}}]$;
Initialize $w = 0$
while $w < D$ do
    /* find $p_w^*$ with largest correlation in reliability
       with $p_t$                                                      */
    $p_w^* = \text{argmax}_{p \in \mathbf{P} \setminus \{p_t\}} \prod_{\mathbf{c} \in \mathbf{C}_w} \text{Rel}_p(\mathbf{c})\text{Rel}_{p_t}(\mathbf{c})$
    $u = u_{\text{min}}$
    while $u < u_{max}$ do
        $\text{Err}_w(u) = \sum_{\mathbf{c} \in \mathbf{C}_w} |\text{PUF}_{p_w^*, u}(\mathbf{c}) - \text{M}(\text{PUF}_{p_w^*, u_{\text{ref}}}(\mathbf{c}))|$
        $u = u + u_{\text{step}}$;
    /* Transfer model: $p_w^* \rightarrow p_t$                         */
    $f_{w,l}(u) = \text{model.fit}([u, \text{Err}_w(u)] \ \forall \ u < u_{\text{ref}})$
    $f_{w,u}(u) = \text{model.fit}([u, \text{Err}_w(u)] \ \forall \ u \geq u_{\text{ref}})$
    /* aggregate model for either side of $u_{\text{ref}}$             */
    $f_w(u) = (f_{w,l}(u), f_{w,u}(u)) \ \forall \ u \in \mathbf{U}$
    $w = w + 1$;
Inference at target temperature/ voltage $u^*$:
$f_w(u^*) = \text{model.predict}(u^*)$
Evaluate $\text{PErr}_w(u^*)$
```

$u < u_k$ to learn the decreasing as well as increasing temperature/ voltage trends separately, where $u_{\text{ref}} = T_{\text{ref}}$ (ambient temperature) or $u_{\text{ref}} = V_{\text{DD}}$ (ambient voltage). The B-spline model fitting function, `model.fit()` shown in Algorithm 1 learns the model parameters, which are then used to predict the number of window errors at test time instance τ_l ($\tau_l > \tau_k$) using model prediction function `model.predict()`, evaluating $f_w(u_l^*)$. This process is repeated for all the D numbers of $s \cdot t$ windows. The trained model is deployed on a controller board equipped with additional memory, which continuously monitors the changing operating conditions of the PUF under consideration. Thereafter, the trained parameters are stored on the controller and used for re-calibration corresponding to device p at a new time instance, as shown in Fig. 3a. However, the aforementioned *continual learning* has to be conducted chip-by-chip because the temperature/ voltage variation behavior for each PUF instance can be different due to the inevitable intrinsic device properties. To alleviate this issue, in the subsequent sub-section, we answer the previously raised question: *Can we incorporate transfer learning for estimating the errors in SRAM-PUF responses at adverse operating conditions?*

C. Transfer Learning (TL): Learning from reference devices of the same family

To overcome the scalability challenge and the time dependence of PUF-specific training, which requires us to train for each PUF instance in sequential time steps, we propose using transfer learning for the first time in the context of SRAM-PUFs. This methodology comprises the *Training* phase and the *Testing* phase. In the training phase, we collect the raw PUF responses from a set of reference devices belonging to the same manufacturer as our device-under-test. This is done by sweeping the temperature/ voltage to find the number of window errors $\text{Err}_w(u)$ concerning the golden response at ambient condition (u_{ref}) for each of these reference devices. At this point, a natural question arises: *Which reference device should we use to predict the number of errors for a given window in the "device-under-test"?*
We address this by choosing the reference device p_w^*, which has the highest correlation of its ambient reliability information with that of the device-under-test p_t for a given window w. The reliability information at u_{ref} for PUF p at $\mathbf{c}$ is given by $\text{Rel}_p(\mathbf{c}) = Pr\left(\text{PUF}_{p,u_{\text{ref}}}(\mathbf{c}) = \text{M}(\text{PUF}_{p,u_{\text{ref}}}(\mathbf{c}))\right)$ where $\text{PUF}_{p,u_{\text{ref}}}(\mathbf{c})$ is the measured response at temperature/ voltage u_{ref}. The reference device p_w^* for window w is given by $\text{argmax}_{p \in \mathbf{P} \setminus \{p_t\}} \prod_{\mathbf{c} \in \mathbf{C}_w} \text{Rel}_p(\mathbf{c})\text{Rel}_{p_t}(\mathbf{c})$ where, $\text{Rel}_{p_t}(\mathbf{c})$ is the ambient reliability information of the target device for challenge $\mathbf{c}$, $\mathbf{C}_w$ denotes the set of challenges in window w, and $\mathbf{P} \setminus \{p_t\}$ refers to the set of devices used for training (excluding p_t). We compute the maximum correlation over the product of reliabilities as it is more sensitive to change in per-bit reliability than averaging with the sum. The reference device p_w^* for each window w is chosen during the enrollment phase of the device-under-test using its ambient reliability information. We split the training data for window w, obtained from p_w^*, into two subsets, i.e., $u < u_{\text{ref}}$ and $u \geq u_{\text{ref}}$ to learn the decreasing/increasing trends separately as shown in Algorithm 2. Thereafter, we *transfer* the B-spline parameters of a window w from p_w^* to p_t, to predict its errors at target operating conditions u^*. Finally, the trained parameters are programmed onto a controller device in the test phase to re-calibrate the PUF response of the "target" device as shown in Fig. 3b. Nevertheless, a crucial factor in our approach involves *determining the correct window size* to reap maximum benefit from the proposed methodology. To do so, we perform a thorough empirical analysis and determine the suitable window size, as detailed in the next subsection.

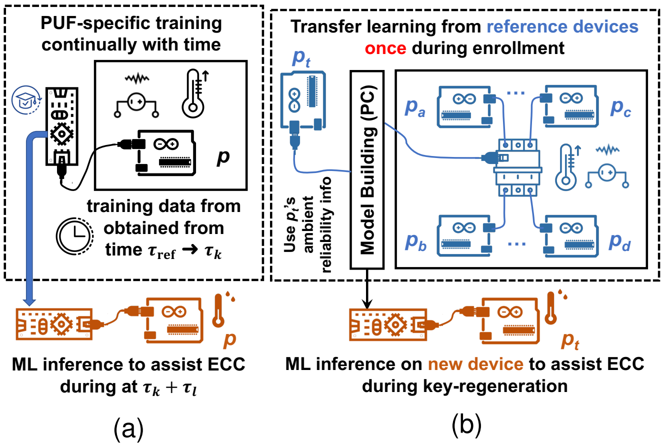

PUF-specific training continually with time
p
training data from obtained from time $\tau_{ref} \rightarrow \tau_k$
ML inference to assist ECC during at $\tau_k + \tau_l$
(a)
Transfer learning from reference devices once during enrollment
p_t
Use p_t's ambient reliability info
Model Building (PC)
p_a
p_c
p_b
p_d
p_t
ML inference on new device to assist ECC during key-regeneration
(b)

Fig. 3: Block diagram for (a) continual learning and (b) transfer learning.

D. Determining Optimal Window Size

The accuracy of the B-spline model determines the window size used to predict the number of errors. The errors in the SRAM cells induced due to PVT variations are spread out across the memory array, with clusters of local errors in the vicinity of selected cells [22]. This discourages the use of large asymmetric windows, such as 32 × 1, as they lack local neighborhood error information and make it more difficult to accurately predict the number of errors in a single large chunk. Furthermore, the possible number of error patterns grows exponentially with the window size, which makes it challenging to uncover the exact error location. Figure 4 shows that the predicted number of errors for continual learning (CL)

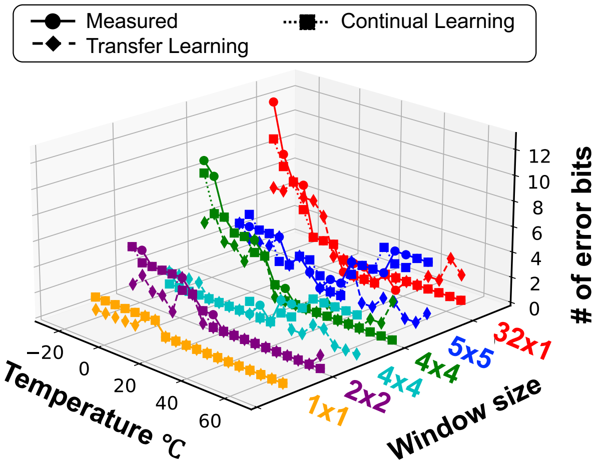


Fig. 4: B-Spline model with ten knots, and third-order basis polynomials predicting errors for a specific window having maximum errors at $-20°$ C and varying size of 32×1, 5×5, 4×4, 3×3 2×2 (f) 1×1.

and transfer learning (TL) for a sample window at $-20°$ C have maximum misprediction value of 13 and 7 respectively.

On the other hand, symmetric windows help uniformly capture the neighborhood information of errors. However, the total number of possible errors increases exponentially with window size, leading to the problem of increased search space for finding the exact errors. This problem can be empirically seen in Figs. 4, where the mispredictions in the number of errors reduces as the window size decreases from 5×5 to 2×2 as it facilitates more inclusion of localized error neighborhoods while keeping the search space small. The 2×2 window offers the best possible symmetric window configuration for finding the error locations tractably, with the maximum (average) misprediction value of 3 (0.7). The further reduction of the window size to 1×1 (Fig. 4) represents the extreme case of predicting the error locations where the combinatorics step is not required. Both 2×2 and 1×1 windows are promising in terms of predicting the number of window errors as they have the shortest search space of all the windows (empirical comparison provided in Section 5.2). However, any mispredictions in the case of a 1×1 window contribute negatively to the final BER. Therefore, it has stringent requirements on the prediction accuracy ($> 99\%$) accuracy to achieve $< 1\%$ BER required to satisfy the desired KER level of <1e-6. This motivates us to formulate the second step of our proposed approach by using a novel combinatorics approach for identifying the precise error locations in the case of 2×2 window, which is described next.

III. Proposed Approach (Step-2): Combinatorics Approach in Predicting Bit-Flips

Once the number of window errors via ML prediction is known, an immediate question arises: *How do we unmask the error locations?* We do this by utilizing the number of errors in overlapping windows that allow us to predict the state of the bit-flip of the target SRAM cell. For a deeper understanding, let us consider a 3×3 window that can be split into four overlapping 2×2 windows, each with e_1, e_2, e_3, e_4 errors.

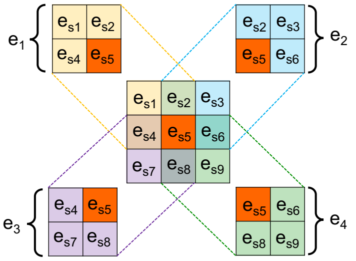


Fig. 5: Error locations in 3×3 window.

We use the term *Number of Errors* (NOE) to denote the set $\{e_1, e_2, e_3, e_4\}$ corresponding to a 3×3 window. This has been illustrated in Fig. 5. Now, each 2×2 window can have between 0 to 4 errors. All the 2×2 sub-windows in a 3×3 window include the middle cell (denoted by e_{s5} in Fig. 5), which leads to the middle cell bit-flip dependence on the NOE. In the case of some NOE's such as $(0, 0, 0, 0)$ (see Fig. 6a) or $(4, 4, 4, 4)$ (see Fig. 6b), the middle cell bit-flip is predicted with absolute certainty. If the number of errors in a window is 0 (4), then none (all) of the SRAM cells in that window have errors. However, in certain cases (as shown in Fig. 6d NOE = (3, 3, 3, 3) or in Fig. 6c NOE = (1, 1, 1, 1)), the middle cell can only be predicted with partial (87.5%) certainty as there exist multiple error patterns which satisfy the NOE but have different values for the middle cell. This is significantly better than randomly predicting the value of a single cell without any window information with a bit-flip probability of 50%.

Fig. 6c and Fig. 6d show one of the seven cases where the middle cell is predicted correctly for NOE (1, 1, 1, 1) and NOE (3, 3, 3, 3) respectively. One may note that all possible NOEs or combinations of e_1, e_2, e_3, e_4 are not physically feasible as the 2×2 windows overlap. For example, NOE = (4,0,0,0) is not possible since if one 2×2 window has four errors, other neighboring 2×2 windows should at least have one. However, the ML model may occasionally predict invalid NOE values, especially in transfer learning, where a window model from a reference device is utilized for the device-under-test. In such cases, we take corrective action by mapping the invalid NOE to the nearest valid NOE based on Euclidean distance. Consider an example where the wrongly predicted NOE is (4, 0, 0, 0). After correction, it gets mapped to (3, 1, 1, 0). Also, one is chosen arbitrarily if there are multiple equidistant valid NOEs from the wrongly predicted NOE.

Mathematically, this observation can be justified by formulating the problem through a set of linear equations. Let us consider the error in nine SRAM cells of a 3×3 window as $e_{s,i}$ where $i \in \{1, 2, \ldots, 9\}$ and $e_{s,i} \in \{0, 1\}$ (see Fig. 5). Then, using the information obtained from the ML prediction, we formulate the following equations:

$$e_{s,1} + e_{s,2} + e_{s,4} + e_{s,5} = e_1; \;\; e_{s,2} + e_{s,3} + e_{s,6} + e_{s,5} = e_2$$
$$e_{s,4} + e_{s,7} + e_{s,8} + e_{s,5} = e_3; \;\; e_{s,6} + e_{s,8} + e_{s,9} + e_{s,5} = e_4 \tag{3}$$

Our aim is to find $e_{s,5}$, which is a variable in all the sub-equations of Eqn. (3). The set of equations solved over the constrained solution space of possible error combinations has no unique solution as there are fewer equations than unknowns

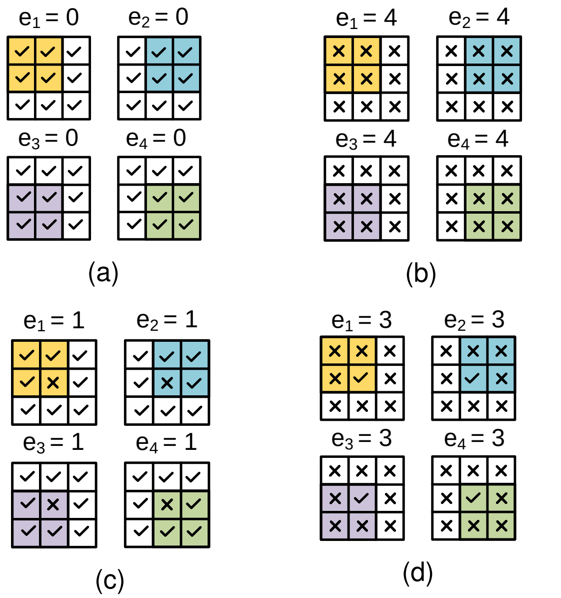
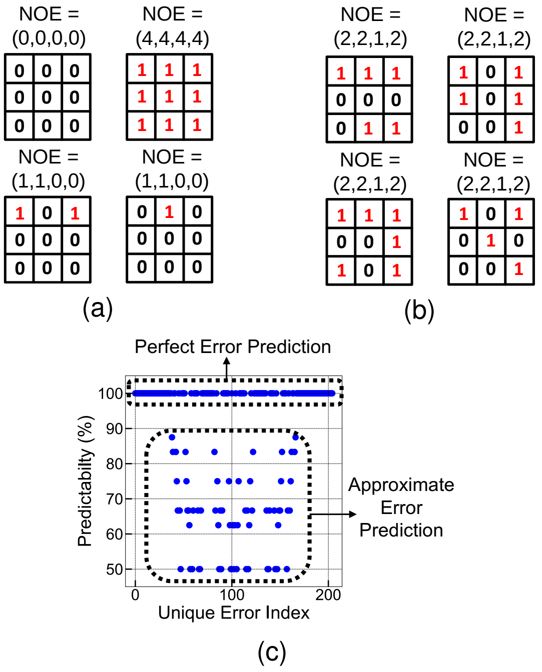


Fig. 6: Error patterns for different NOE's with absolute ((a) & (b)) and partial ((c) & (d)) predictability.



NOE = (0,0,0,0)

0	0	0
0	0	0
0	0	0

NOE = (4,4,4,4)

1	1	1
1	1	1
1	1	1

NOE = (1,1,0,0)

1	0	1
0	0	0
0	0	0

NOE = (1,1,0,0)

0	1	0
0	0	0
0	0	0

NOE = (2,2,1,2)

1	1	1
0	0	0
0	1	1

NOE = (2,2,1,2)

1	0	1
1	0	1
0	0	1

NOE = (2,2,1,2)

1	1	1
0	0	1
1	0	1

NOE = (2,2,1,2)

1	0	1
0	1	0
0	0	1

Fig. 7: Examples of error patterns (a) with 100% predictability (b) with < 100% predictability, and (c) empirical predictability for all unique error patterns.

(an "underdetermined" system of linear equations). Based on the choice of our window size, we empirically analyze the solution space rather than resorting to closed-form expressions for the optimal solution.

In order to analyze NOEs further, consider an example of a 3×3 window with zero errors. Each of the nine SRAM cells can have an error leading to $2^9 - 1 = 511$ unique error patterns. Figure 7a shows the case of error patterns corresponding to an NOE with 100% predictability, and Fig. 7b depicts the partial predictability case. The latter case occurs when multiple solutions exist for Eqn. (3), which necessarily do not have the same value of $e_{s,5}$. In our empirical analysis for Eqn. (3), we find an NOE associated with each error pattern leading to 205 unique NOE, which covers the space of 511 error patterns. Furthermore, we empirically conclude there are 142 NOEs, which always map to a deterministic middle cell error leading to 100% predictability as shown in Fig. 7c. For the remaining 63 NOEs, the mapping to the middle cell depends on the specific error pattern. In such a case, the question arises, *Which error pattern is the most probable, and how do we determine that?* To answer the question, consider the row vector $\mathbf{e} = [e_{s,1} e_{s,2} \ldots e_{s,9}]$ denoting the error patterns for a given NOE. We find the probability of occurrence $P_o(\mathbf{e})$ for each error pattern $\mathbf{e}$ using the reliability information at u_{ref}, Rel($\mathbf{c}$). We evaluate $P_o(\mathbf{e})$ as: $\prod_{i=1}^{9} ((1 - e_{s,i})\text{Rel}(\mathbf{c}_i) + e_{s,i}(1 - \text{Rel}(\mathbf{c}_i)))$.

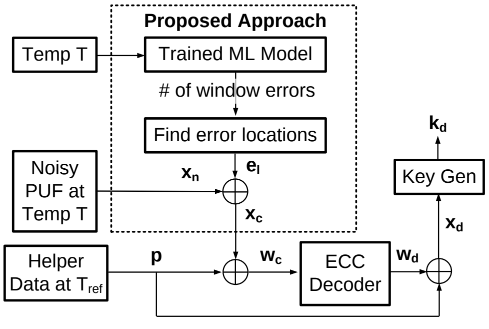


Fig. 8: Reconstruction using helper data & ML-assisted ECC.

The error pattern with the highest $P_o(\mathbf{e})$ is chosen for a given NOE, thereby providing the most probable error pattern $\mathbf{e}$ and, in turn, revealing $e_{s,5}$. Lastly, the error location vector $\mathbf{e}_l = [e_{l,1}, e_{l,2}, \ldots, e_{l,n}]$ is XORed with the noisy PUF response $\mathbf{x}_n$ to obtain the re-calibrated PUF response $\mathbf{x}_c$ (see Fig. 8). Thereafter, the standard PUF reconstruction steps using a fixed ECC are followed to obtain the decoded PUF response $\mathbf{x}_d$. One may note that our ML approach re-calibrates the systematic errors due to voltage/ temperature, along with a fixed low-overhead ECC corrects the unpredictable noise-related errors. In essence, both together achieve a close to the desired 100% reliability. However, the probabilistic nature of the ML approach may lead to false positives in the error location vector $\mathbf{e}_l$. This arises for the i^{th} location when, $e_{l,i}$[†] is predicted as 1 even though its correct value is 0. Interestingly, our empirical observation showed that the 142 NOEs with deterministic middle cell error do not contribute to false positives. The remaining 63 NOEs with non-deterministic middle cell error contribute to the false positives, occurring in less than 2% of the total windows in the worst case. The presence of isolated error patterns in training data having low spatial correlation (windows with single error) contributes to the false positive rate. We observe that the false positive rate is the minimum for 2×2 window compared to 1×1 windows with no neighborhood information and 3×3 windows with a higher range of errors. The false positive rate can be reduced by correctly predicting the isolated errors with no spatial correlation, which is challenging to achieve without knowledge of SRAM cell parameters. Nevertheless, the fixed

[†] In the error location vector $\mathbf{e}_l$, the i^{th} index value $e_{l,i}$ is obtained from middle cell error $e_{s,5}$ of the i^{th} enclosing 3×3 window.

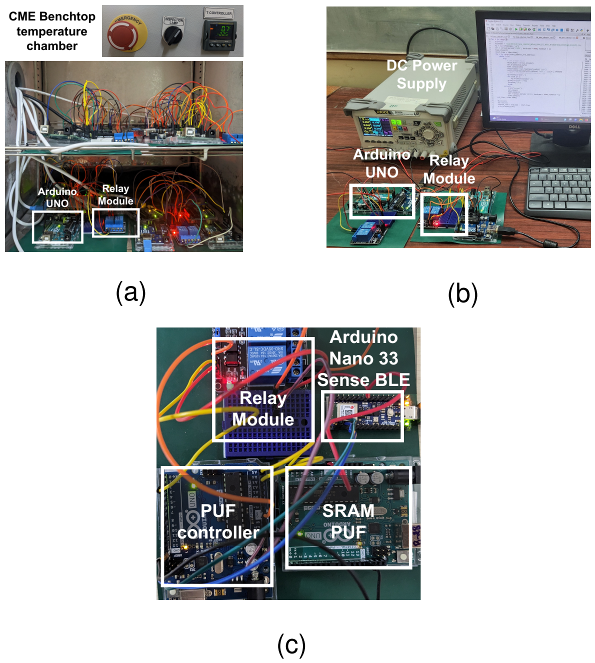


Fig. 9: Our test-bed architecture at the time of data collection using (a) temperature chamber, (b) DC power supply across multiple SRAM-PUFs, and our (c) ML Inference setup with Arduino Nano Sense.

ECC, implemented after the combinatorics unit, effectively eliminates false positives arising in these 63 NOEs, thereby preserving the overall BER. We present our proposed scheme's experimental details, results, and analysis hereafter.

IV. Experimental Details, Results, and Analysis

This work involves implementing the SRAM-PUF in the immensely popular lightweight development board Arduino UNO, equipped with an 8-bit AVR ATmega328P microcontroller with 2kB SRAM memory. Additionally, to evaluate the impact of our proposed methodology on scaled-up memories, we extended our experiments to Arduino-Zero (a 32-bit microcontroller), consisting of an ARM Cortex M0+ and SAMD21 microcontroller with 32 kB SRAM. Both the Arduino UNO and Zero boards housing SRAM-PUF are driven by a controller and a relay module that acts as a switch for facilitating the automated collection of the start-up-values of the SRAM cell. The average uniformity, uniqueness, and reliability for UNO were obtained as 62.60%, 46.99%, and 98.02%. Additionally for Zero these values were 54.72%, 49.64%, and 96.08% respectively. Furthermore, the start-up data was collected 15 times at ambient conditions to generate the golden response using majority voting technique [14]. We use only the stable cells for our analysis having reliability between 90-100% as $\approx 90\%$ of the SRAM cell satisfies this criterion in both Arduino UNO and Zero boards.

A. Experimental Setup

Our test-bed architecture involves studying the impact of environmental variations across the temperature spectrum of $-20°$ to $70°C$. For this purpose, we use the CME Benchtop Temperature Chamber to perform a temperature sweep over the entire range and collect the training data in steps of $2.5°$ C (see Fig. 9a). On the other hand, for the voltage sweep, a programmable DC power supply was employed to control the input voltage to the "Vin" pin on the Arduino UNO (Zero) board within the range of 3.8 to 6.2 V (7 to 12 V) with step size of 0.2 V as shown in Fig. 9b. The voltage sweep range is tied to the operating range of the onboard voltage regulator for UNO and Zero boards, beyond which the entire power shuts off (lower limit) or the board ceases to be functional (upper limit). However, from our experiments (presented in Sec. IV-C1), we infer that the internal voltage of the SRAM memory array gets impacted by the external supply as bit-flips are observed in the presence of voltage fluctuations.

Next, to capture the variable number of window errors under changing operating conditions, the B-Spline model with ten knots and third-order basis polynomials was selected as this configuration of the cubic spline with uniformly spaced knots gives the highest prediction accuracy even for worst-case BER.

B. Comparison between 2×2 and 1×1 Windows

The reduction of window size to 1×1 represents the extreme case of predicting the precise error locations where the combinatorics unit for uncovering the location of bit-flips is not required. Although using a 1×1 window improves the prediction accuracy of transfer learning when compared to 2×2 window , the $< 90\%$ prediction accuracy for temperatures $T < 0°$ C leads to a post-ECC BER of $> 10\%$, which is not desirable. This is because predicting the error locations directly using the 1×1 window with no neighborhood information introduces false positives or additional errors. Henceforth, we shall use the 2×2 windows for detailed analysis.

C. Error Reduction after Re-calibration of PUF Response

Our proposed ML-assisted ECC approach provides us with a re-calibrated PUF response over a wide range of temperature and voltage fluctuations, which is closer to the PUF golden response at ambient conditions. This, in turn reduces the BER while achieving an equivalent min-entropy loss as a standard ECC scheme.

1) Re-calibration for PVT Variations: Our experimentation with multiple SRAM-PUF instances shows an increasing trend in BER as we move away from ambient operating conditions. This increased BER lies in the range of 10%- 17% at $-20°$C for SRAM-PUFs implemented on Arduino UNO and 5%- 7% on Arduino Zero as shown in Fig. 10a and 10b respectively. However, from Fig. 10c and 10d, it can be seen that the BER is relatively smaller with respect to voltage variations in both UNO and Zero boards respectively. This observation can be attributed to the presence of on-board voltage regulators. Nevertheless, the internal supply of the SRAM memory gets impacted due to the voltage fluctuations leading to bit-flips, necessitating the need for error correction techniques.

The increased BER induced due to temperature variations and voltage fluctuations are reduced via re-calibrating the reference response using our two-step approach. In the first step of window error prediction, our ML model demonstrates a prediction accuracy of approximately $> 95\%$ for continual learning (CL) across all temperatures (Fig. 11a) and voltage conditions (Fig. 11b). The CL accuracy trend

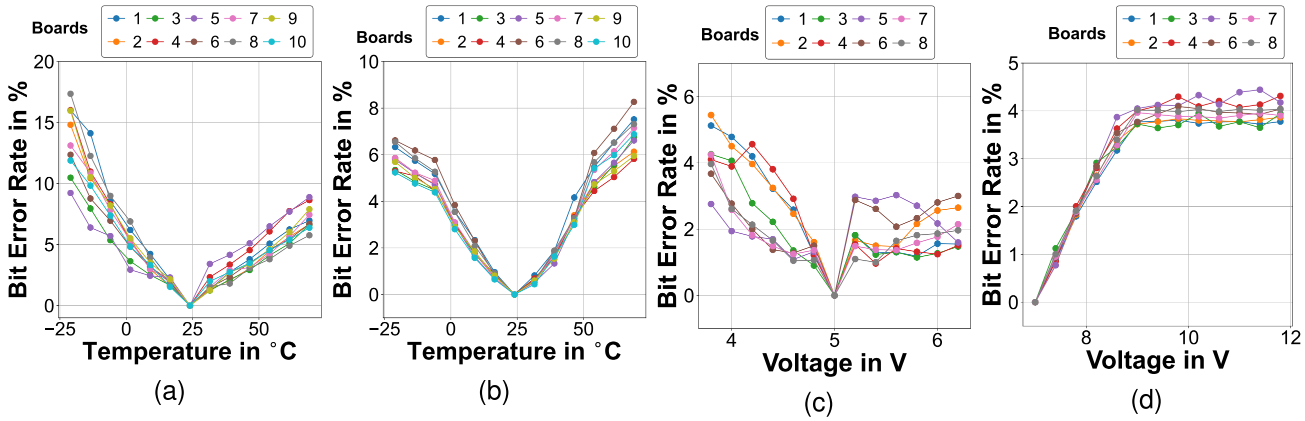


Fig. 10: The increasing BER across temperature variations for SRAM-PUF instances implemented on (a) Arduino UNO and (b) Arduino Zero boards. The BER was obtained across voltage fluctuations for SRAM-PUF instances and implemented on (c) Arduino UNO and (d) Arduino Zero boards.

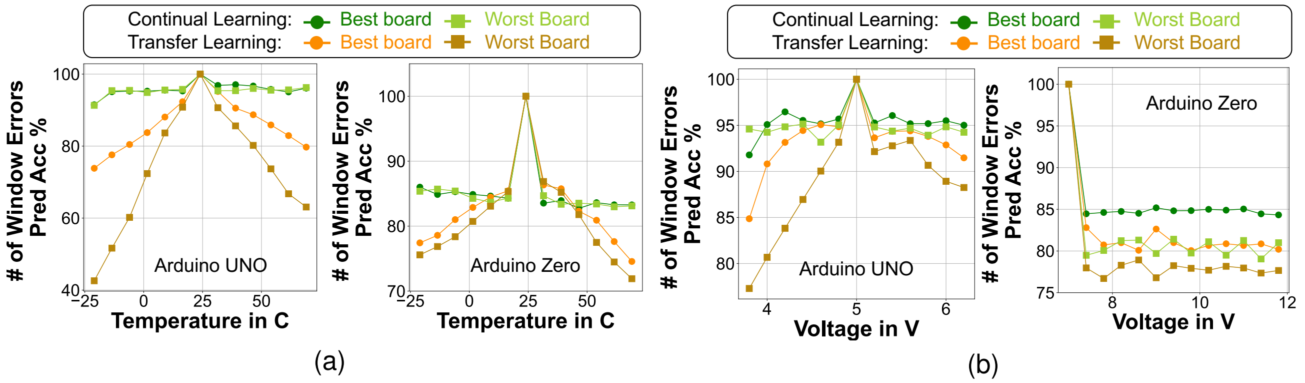


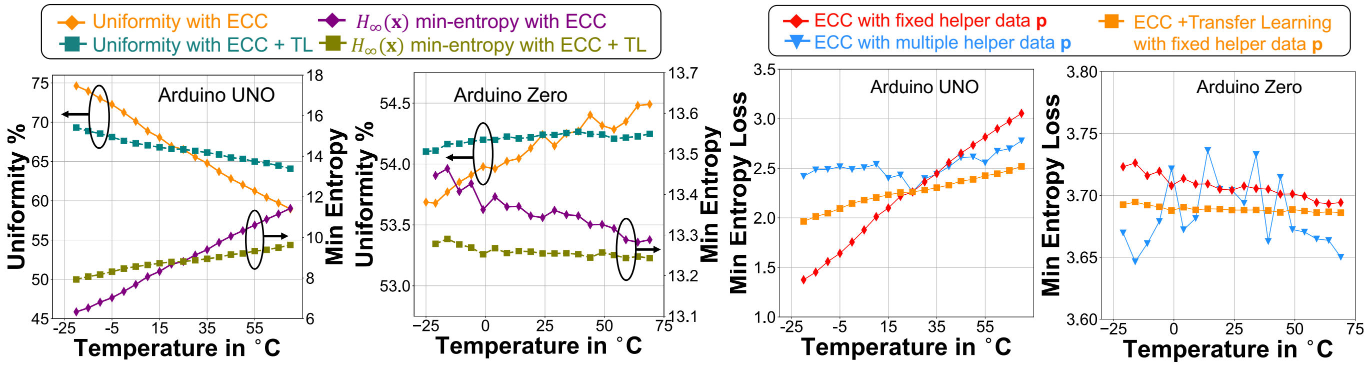
Fig. 11: Prediction accuracy of window errors using continual (CL) and transfer learning (TL) across the entire (a) temperature and (b) voltage sweep for Arduino Uno and Arduino Zero.

for UNO (approximately $> 82\%$ over boards) is similar for the scaled-up memory in Arduino Zero, signifying our ML approach's independent applicability. However, due to the associated training cost for each device, transfer learning (TL) emerges as a more attractive approach for generating a PUF-agnostic model. The increase in TL accuracy from $\approx$ 58% to 93% as we approach the ambient condition arises due to the high correlation in ambient PUF responses across multiple boards. Note that, our ML model does not overfit the training samples, as the accuracy remains consistent even with the application of regularization techniques such as L1 and L2 regularization. The accuracy obtained through 5-fold cross-validation, with similar accuracies across each fold, further indicates the absence of overfitting.

The lower accuracy in transfer learning at extreme conditions primarily stems from the failure to predict a 1-bit error in windows that always had zero errors in the training set of its reference devices. However, an ECC with an error correcting capacity t consistently corrects these singular window errors, ultimately resulting in a low BER $\lessapprox 1\%$ and KER of $\lessapprox$ 1e-6. In the case of the worst board with transfer learning accuracy ranging from 55%-90% (Fig. 11a for UNO), the raw BER reduces from 13% to 6%, which is thereafter reduced to $\lessapprox$1% using a fixed ECC. Our ECC parameters are BCH[127,71,9] and BCH[63,30,6] for UNO and Zero, respectively, which meet the BER requirement for all boards.

2) Entropy Loss Estimation: The min-entropy loss is computed as $H_\infty(X) - \tilde{H}_\infty(X|P)$, following the relationship between helper-data, coset, and standard array as defined in [26]. Figure 12a shows that for Arduino, UNO min-entropy increases with temperature as the PUF uniformity reduces, becoming maximum when uniformity becomes closer to 50%. However, the uniformity-temperature trend for the Zero board (54% average uniformity) is reversed (Fig. 12a). Nonetheless, in both cases, the change in min-entropy is lower for our approach than standard ECC schemes as our method strives to narrow the gap between uniformity w.r.t. the ambient conditions. Note that we use transfer learning results (TL) for min-entropy analysis due to its generality.

Next, we evaluate the min-entropy loss for a BCH[15,11,1] code with maximum loss of $n - k = 4$. Following the min-entropy loss calculation in [26] we observe lower loss values, which increase gradually with temperature for our ML-assisted ECC as opposed to standard ECC with fixed helper data in case of UNO (see Fig. 12b). Additionally, our method achieves lower min-entropy loss at all temperatures than the multiple enrollment scheme [8]. On the other hand, the min-entropy loss with multiple helper data shows an erratic behavior for Arduino Zero with close-to-ideal PUF uniformity, compared to the relatively constant min-entropy loss of 3.69 in our case



(a) (b)

Fig. 12: Temperature dependence of (a) uniformity and min-entropy for SRAM-PUF implementation and (b) min-entropy loss estimation for Arduino UNO, and Arduino Zero.

		Code: BCH[n,k,t], Rep[n,1], or both	Raw BER(%)	Final BER(%)	KER	Volt Range(V)	Temp Range(°C)	Max Leakage*	Helper Bits	Raw Bits	Post-processing Instruction Count	Latency (Cycles)	Energy$ (μJ/decode)	Test PUF Platform
Concatenated Code [9]		Rep[9,1] + BCH[121,86,5]	15	-	6.54e-5	-	-	1003	1089	2178	8.9K	257	1.28	-
Standard ECC [27]		BCH[511,19,119]	15	-	2.97e-7	-	-	492	511	4599	205K	1141	132	-
Standard ECC [27]		BCH[1023,46,219]	15	-	1.85e-8	-	-	977	1023	4092	378K	2265	486	-
Our Work	**CL**	**Concatenated: Rep[5,1] + BCH[127,99,4]**	**13.02**	**0.32**	**1.33e-9**	[3.8,6.2]	[-16,70]	**536**	**635**	**1270**	**13K#**	**268**	**2.35**	**Arduino UNO**
	TL			**0.85**	**2.52e-6**									
	CL	**BCH[127,71,9]**		**0.75**	**6.07e-7**			**56**	**127**	**381**	**21.7K#**	**273**	**3.75**	
	TL			**0.98**	**5.2e-6**									
	CL	**Concatenated: Rep[5,1] + BCH[63,45,3]**	**7.93**	**0.11**	**8.97e-11**	[7,12]	[-25,66.5]	**270**	**315**	**1260**	**11.3K#**	**139**	**1.08**	**Arduino Zero**
	TL			**0.15**	**4.73e-10**									
	CL	**BCH[63,30,6]**		**0.021**	**6e-15**			**33**	**63**	**378**	**16.5K#**	**142**	**1.52**	
	TL			**0.055**	**2e-12**									
Soft Decoding [28]		Rep[3,1] + RM[2,6]	15	-	<1e-6	-	-	170	192	1536	3.7K	1680	3.52	-
Soft Decoding with all reliabilities		Rep[3,1] + RM[2,5]	7.12	1.29	2.25e-9	[3.8,6.2]	[-1,70]	80	96	1056	1.67K	650	1.1	**Arduino UNO**
			17.2	10.55	1.67e-2		[-25,70]							
Our Work	**CL**		**17.2**	**1.29**	**2.25e-9**		**[-25,70]**				**6.5K#**	**672**	**2.3**	
	TL			**1.29**	**2.25e-9**									
Soft Decoding with all reliabilities		Rep[3,1] + RM[2,4]	4.48	0.52	8.5e-7	[7,12]	[1.5,59]	37	48	768	1K	364	0.9	**Arduino Zero**
			9.00	3.09	9.25e-4		[-25,70]							
Our Work	**CL**		**9.00**	**0.67**	**2.16e-6**		**[-25,70]**				**6K#**	**384**	**1.4**	
	TL			**0.7**	**2.7e-6**									

# instruction count of our proposed architecture including ML model inference (5.5K) using a B-spline with 10 knots and cubic spline, and ECC (see Fig. 8); & Key size of 24 as opposed to 128 for all other works; $ energy cost of post-processing implementation on Arduino Nano Sense board for all works; M() - Majority Voting.; $*$ $(n-k)$ is the upper bound for the helper data leakage

Table II: Comparison with state-of-the-art works during key-regeneration phase.

(Fig. 12b). Interestingly, in both cases, our method fares better in terms of change in min-entropy loss. We attain a min-entropy loss between 2-2.5 bits for Arduino UNO, performing at par with the standard ECC techniques with an average loss of 2 bits and multiple helper data methods with an average of 2.5 bits for a 15-bit codeword for UNO board. One may note that similar min-entropy loss trends are also observed for large ECC codes such as BCH[127,29,21], as shown in [26].

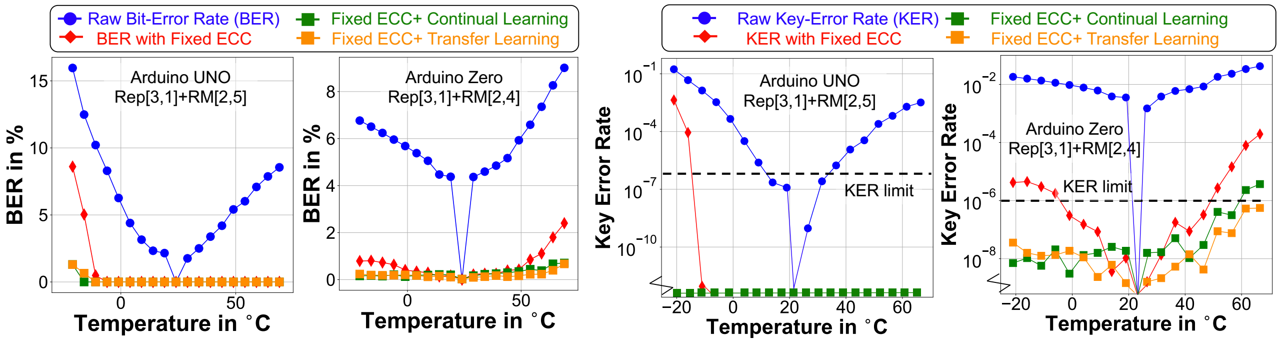
D. Comparison with Other ECC techniques

Key Generation: A key generation function is a part of HDA (e.g., hash function), which takes PUF response as input to generate the cryptographic key. The relationship between the key error rate (KER) and the bit error rate (BER) for a $[n, k, t]$ linear code is given by $KER = 1 - \left(\sum_{i=0}^{t} \binom{n}{i} (1 - BER)^{(n-i)} BER^{i}\right)$ where, n is the codeword length, k is the message length, and t is the maximum error-correcting capability of the linear code. A conservative KER of $\lessapprox 10^{-6}$ [5] dictates the requirement of high error correcting capability ($t > 100$) for a raw BER of 15% [9]. From Table II, we see that our ML-assisted ECC reduces the error correcting capacity t of the fixed ECC required for achieving an effective KER of $\lessapprox 10^{-6}$ over a wide range of temperature/ voltages.

Helper Data Leakage: The helper data leakage (upper bounded by $(n - k)$) drops as the redundancy of ECC [10] or the error correction capacity t reduces. Our ML-based PUF re-calibration obtains $\approx 1\%$ worst-case final BER in all conditions simultaneously with a low helper data leakage compared to conventional ECC techniques. The lower bound of the SRAM memory size is determined by the number of PUF raw bits required for key generation, which is proportional to t. We only need 381 and 378 raw PUF bits to generate a 128-bit key using a BCH code for UNO ($t = 9$) and Zero ($t = 6$) boards. This is significantly lower compared to previous works [27] with $t \geq 18$ requiring > 1000 raw PUF bits.

Energy and Latency: To ensure fair comparison in terms of energy cost and latency incurred for post-processing, we have implemented all the previous ECC techniques ([9], [27], [28]) on the Arduino Nano 33 BLE Sense board. Note that the Nano board implementing the ML-based re-calibration facilitates testing different SRAM-PUFs, multiple UNO, and Zero boards with identical post-processing, allowing us to evaluate our approach against others. It is apparent from Table II that the proposed ML-assisted BCH[127,71,9] achieves a 35.2× lesser energy, 4.17× lower latency, and 9.44× lower instruction count compared to BCH[511,19,119] [27] at iso-KER.

Comparison with Soft Decoding based ECC: Soft-decoding-based decoders use reliability information to reduce the ECC resource requirements while achieving the same



(a)

(b)

Fig. 13: Temperature dependence of (a) BER after post-processing and (b) the corresponding KER obtained for Arduino UNO using Rep[3, 1]+RM[2, 5] and Arduino Zero using Rep[3, 1]+RM[2, 4] PUF implementations.

post-ECC BER levels. We highlight that the key difference between our proposal and soft decoding is that we also factor in the voltage-temperature dependence of bit errors with the reliability information. When *using soft-decision information for different environmental conditions* i.e., augmenting the soft decision information at the ambient conditions with reliabilities at varying temperatures/voltages, we observe that the soft-decoding scheme in conjunction with the Rep[3,1] + RM[2,5] achieves the desired key-error rate (KER) of $<$ 1e-6 only in the temperature range of $[-1^\circ, 70^\circ]$ C. In contrast, our proposed post-processing scheme with Rep[3,1] + RM[2,5] achieves the desired KER of <1e-6 as shown in Fig. 13b across all operating conditions, even correcting a worst-case BER of 17.2% (see Fig. 13a). The success of our methodology is attributed to the fact that it learns the voltage/temperature dependence of the number of errors per window both for reliable and unreliable cells, thereby achieving the desired KER with reduced ECC parameters. Furthermore, we obtain a massive 52% reduction in the maximum leakage value and a 30% drop in the number of raw PUF bits required when compared to [28] with comparable latency and energy values. Similar trends are observed for Arduino Zero (ARM Cortex M0+) with lower worst-case BERs, requiring a smaller concatenated code, i.e., Rep[3,1] + RM[2,4].

E. Impact of both temperature and voltage variations:

The BER worsens at extreme temperature and voltage variations (maximum of 20% at -20° C and 3.8 V) compared to only temperature variations (15% at -20° C) requiring an ECC of Rep[5,1] + BCH[127,64,10]. However, the spatial relationship between the errors still persist, which is utilized in our ML-assisted error correction scheme to reduce the ECC complexity. Using all the training data from an SRAM-PUF of the same family, our transfer learning approach reduces the ECC requirement to Rep[5,1] + BCH[127,86,6], even correcting a worst-case BER of 20% (see Fig. 14). At the same temperature, our method also reduces the 3-σ (standard deviation) of BER caused by voltage variation from 2.23% in the case of standard ECC to 0.97%.

V. Discussion

ML Model Leakage: Our ML model is trained using the number of window errors under variable operating conditions, ensuring raw PUF responses remain undisclosed. This methodology inherently limits the information to differential error counts rather than exact responses, rendering any attempt to predict the raw PUF responses merely a random guess. Notably, adversaries cannot access raw PUF data in SRAM-PUFs, a category of weak-PUFs designed to protect challenge-response pairs [11]. Additionally, the model does not retain ambient reliability data beyond the PUF enrollment phase, eliminating the risk of reliability-dependent data exposure. Lastly, given the ML model output, i.e., the error locations at given operating condition e, the probability of PUF response x being one, is close to the PUF's ambient uniformity U, i.e., $P(x = 1|e = 1) = P(x = 1) \approx U$. One can only predict response $\boldsymbol{x} \in \{0, 1\}^n$ with a probability of $1/\binom{n}{U\cdot n}$, which is negligible (<1e-25) for a typical raw PUF response size of $n > 100$ and $0.45 \le U \le 0.55$. As a result, we do not see the formulation of an imminent attack vector by exploiting the proposed ML model.

Memory Overhead: The proposed ML technique implemented on the Arduino BLE 33 Nano Sense board requires only 512 KB of memory including the storage overhead of the dark bits mask needed to isolate cells with less than 90% reliability during the reconstruction phase (*Note: We use 50% of the Arduino BLE 33 Nano Sense board's on-chip flash memory of 1MB*). Since the model parameters depend on the memory size, it remains consistent across all ECCs, despite varying computation overheads, as indicated by the post-processing instruction count in Table II. It should be noted that our ML approach is functionality-driven, demonstrating efficacy across a wide range of temperatures and voltages. To optimize the memory footprint, one potential approach is to design a fixed-point digital implementation with sparsity awareness, storing only the non-zero B-spline parameters and sharing the same parameter value across windows. This optimization would make our approach competitive with FPGA/ASIC implementations of SRAM-PUFs equipped with circuit-level reliability enhancement schemes.

VI. Conclusion

This work proposes an ML-based PUF re-calibration scheme to enhance the reliability of SRAM-PUFs, reducing their worst-case BER from 17.02% to 1.29% thereby, meeting the practical KER requirement of $< 10^{-6}$ for stable key generation over a wide operating range. The proposed post-

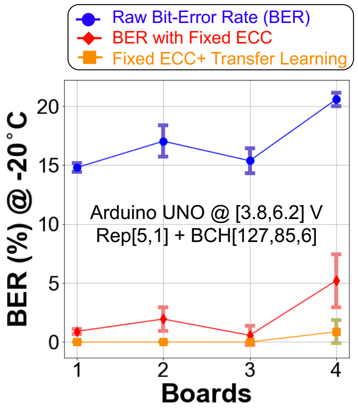


Fig. 14: Error correction at -20° C with 3-σ deviation due to voltage variation.

processing scheme achieves high reliability without exposing helper data (single enrollment at ambient conditions) across all operating conditions and re-configuring the ECC. Our method is based on inferring the reference point at adverse operating conditions via learning from the PUF responses 1) at past operating conditions and 2) transferring the learned model across devices, aided with combinatorics unit and reliability information. Our extensive experiments on multiple instances of SRAM-PUFs in popularly used Arduino UNO and Zero boards are obtained with a temperature sweep in the range of $[-25, 70]^\circ$ C and a voltage sweep in the range $[3.8, 6.2]$ V and $[7, 12]$ V respectively, demonstrating the efficacy of our ML-based approach on real-world SRAM-PUFs. Furthermore, in the presence of both voltage (3.8V) and temperature (-20°C) variation leading to a worst-case BER 20% our proposed approach corrected it to $< 2\%$. This novel approach provides low leakage and efficient SRAM-PUF reliability enhancement over a broad operating regime.

References

[1] IntrinsicID, “Protecting a Device’s Root Secrets With SRAM PUF,” http://www.is.gd/intrinsicidPUF, 2020.

[2] M.-Y. Wu *et al.*, “A PUF scheme using competing oxide rupture with bit error rate approaching zero,” in *ISSCC*. IEEE, 2018.

[3] D. Fainstein *et al.*, “Dynamic intrinsic chip ID using 32nm high-K/metal gate SOI embedded DRAM,” in *2012 VLSIC*. IEEE.

[4] S. Taneja *et al.*, “PUF architecture with run-time adaptation for resilient and energy-efficient key generation via sensor fusion,” *JSSC*, 2021.

[5] ——, “Fully synthesizable PUF featuring hysteresis and temperature compensation for 3.2% native BER and 1.02 fJ/b in 40 nm,” *JSSC*, 2018.

[6] Y. Dodis *et al.*, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” in *EUROCRYPT*. Springer, 2004.

[7] Y. Gao *et al.*, “Building secure SRAM PUF key generators on resource constrained devices,” in *IEEE PerCom Workshops*, 2019, pp. 912–917.

[8] Y. Gao, Y. Su, L. Xu, and D. C. Ranasinghe, “Lightweight (reverse) fuzzy extractor with multiple reference PUF responses,” *TIFS*, 2018.

[9] C. Bösch *et al.*, “Efficient helper data key extractor on FPGAs,” in *CHES*. Springer, 2008.

[10] J. Delvaux *et al.*, “Helper data algorithms for PUF-based key generation: Overview and analysis,” *TCAD*, 2014.

[11] T. McGrath *et al.*, “A PUF taxonomy,” *Applied physics reviews*, vol. 6, no. 1, 2019.

[12] B. Karpinskyy *et al.*, “8.7 physically unclonable function for secure key generation with a key error rate of 2E-38 in 45nm smart-card chips,” in *ISSCC*. IEEE, 2016.

[13] S. Satpathy *et al.*, “A 4-fJ/b delay-hardened physically unclonable function circuit with selective bit destabilization in 14-nm trigate CMOS,” *JSSC*, 2017.

[14] M. Bhargava *et al.*, “Reliability enhancement of bi-stable PUFs in 65nm bulk CMOS,” in *HOST*. IEEE, 2012.

[15] A. Rodrigues *et al.*, “SmartFusion2 SoC as a security module for the IoT world,” in *ACM CCF*, 2022, pp. 270–278.

[16] N. Semiconductors, “AN12324 LPC55Sxx usage of the PUF and Hash Crypt to AES coding,” *NXP Semiconductors*, 2019.

[17] T. Lu *et al.*, “Secure device manager for Intel Stratix 10 devices provides FPGA and SoC security,” *Intel, Santa Clara, CA, USA, White Paper WP-01252-1.2*, 2018.

[18] G.-J. Schrijen and C. Garlati, “Physical unclonable functions to the rescue,” *Proceedings of the Embedded World*, 2018.

[19] R. Suragani *et al.*, “Identification and classification of corrupted PUF responses via machine learning,” in *IEEE HOST*, 2022, pp. 137–140.

[20] O. Millwood *et al.*, “PUF-phenotype: A robust and noise-resilient approach to aid group-based authentication with DRAM-PUFs using machine learning,” *IEEE TIFS*, 2023.

[21] A. Roelke and M. R. Stan, “Controlling the reliability of SRAM PUFs with directed NBTI aging and recovery,” *TVLSI*, 2018.

[22] K. Xiao *et al.*, “Bit selection algorithm suitable for high-volume production of SRAM-PUF,” in *HOST*. IEEE, 2014.

[23] D. Nedospasov *et al.*, “Functional integrated circuit analysis,” in *HOST*. IEEE, 2012.

[24] P. H. Eilers and B. D. Marx, “Flexible smoothing with B-splines and penalties,” *Statistical science*, 1996.

[25] F. Lin *et al.*, “Certified space curve fitting and trajectory planning for CNC machining with cubic B-splines,” *CAD*, 2019.

[26] J. Delvaux *et al.*, “Efficient fuzzy extraction of PUF-induced secrets: Theory and applications,” in *CHES*. Springer, 2016.

[27] M. Hiller *et al.*, “Review of error correction for PUFs and evaluation on state-of-the-art FPGAs,” *Journal of Cryptographic Engineering*, vol. 10, no. 3, pp. 229–247, 2020.

[28] R. Maes *et al.*, “Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs,” in *CHES*. Springer, 2009.

Kuheli Pratihar is a PhD student at the Department of Computer Science and Engineering, IIT Kharagpur under the supervision of Prof. Debdeep Mukhopadhyay and Prof. Rajat Subhra Chakraborty. Her primary research work involves design, lightweight implementation, and analysis of hardware security primitives like Physically Unclonable Functions (PUFs), True Random Number Generators (TRNGs) etc.

Soumi Chatterjee is pursuing her PhD under the supervision of Prof. Debdeep Mukhopadhyay in the Department of Computer Science and Engineering at the Indian Institute of Technology, Kharagpur. Her primary research focuses on the Side Channel Analysis of Cryptographic Algorithms and the design of lightweight countermeasures for the development of secure systems.

Rajat Subhra Chakraborty is a professor with the Department of Computer Science and Engineering, IIT Kharagpur. His area of research is hardware security, VLSI design and digital content protection through watermarking. He is a senior member of IEEE and ACM.

Debdeep Mukhopadhyay received his PhD degree from IIT Kharagpur. Currently, he is a full professor with the Department of Computer Science and Engineering, IIT Kharagpur. His research interests include hardware security, micro-architectural attacks, cryptography, VLSI. He is a senior member of IEEE and ACM, and the recipient of the prestigious Shanti Swarup Bhatnagar prize 2021.