Security Enhancements for Approximate Machine Learning

He Li
hl556@cam.ac.uk
University of Cambridge
Cambridge, UK

Yaru Pang
yaru.pang.17@ucl.ac.uk
University College London
London, UK

Jiliang Zhang*
zhangjiliang@hnu.edu.cn
Hunan University
Changsha, China

ABSTRACT

Approximate computing techniques for error-tolerant machine learning applications are gaining interest, promising a energy-accuracy balance for modern digital computing systems. As an ubiquitous step in machine learning, iterative solvers have been widely used for training neural networks and accelerating feedforward computations. In this paper, we provide three novel information hiding techniques that use properties of redundant number systems, most-significant digit-first arithmetic and the forward error analysis of stationary iterative methods, to secure approximate computing systems. We demonstrate that three different security signatures are encoded through redundant representation, function-equivalence arithmetic replacement and algorithmic optimisation. Our illustrative security enhancement countermeasures can be used to prevent potential attacks, such as privacy leakage, out-of-control systematic error and error injection in approximate computing.

CCS CONCEPTS

• **Mathematical analysis** → Numerical analysis; • **Security and privacy** → *Security in hardware*; • **Computing methodologies** → Machine learning.

KEYWORDS

Computer arithmetic, approximate computing, machine learning, iterative methods

ACM Reference Format:
He Li, Yaru Pang, and Jiliang Zhang. 2021. Security Enhancements for Approximate Machine Learning . In *Proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI'21).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3453688.3461753

1 INTRODUCTION

The past decade has witnessed the growing interest in approximate computing techniques for machine learning (ML) applications. ML applications are data-rich, and their high degree of error tolerance promises resource-efficient implementations through the investigation of approximation at algorithm, architecture or circuit levels [29]. The scaling of digital computing systems necessitates the growing demand of arithmetic intellectual property (IP) cores [19], thereby causing frequent reconsideration of the trustworthy issues behind these arithmetic operators [24]. In 2020, over four hundred vulnerabilities have been found in the digital signal processor (DSP) of Qualcomm's Snapdragon chip, which was estimated to affect nearly half of the global mobile phone market [2]. Through this event, we realised that system designers or mobile vendors rarely access the design details of arithmetic operators [2]. It is necessary to provide security enhancement techniques, such as embedding designers' signatures in arithmetic operations, to build secure digital systems. Interestingly, approximate computing paradigms have demonstrated possibilities for secure information hiding [47, 48].

Even though many approximate adders and multipliers have been proposed [19], they produce their output digits in the order of increasing significance, *i.e.* least-significant digits (LSDs) first. This results in a fundamental problem: if a datapath contains a chain of approximate adders or multipliers, the error generated by each operation will accumulate due to carry propagation, causing uncontrollable errors in the final results. Therefore, most of these approximate operators will often require an overestimated precision, and cannot realize the full potential of the approximate computing techniques [11].

Use of most-significant digit (MSD)-first arithmetic avoids the need to consider the aforementioned problem entirely, due to its attractive properties for variable precision operation [15, 22, 51]. One can simply terminate an operation once the desired number of accuracy have been generated. MSD-first arithmetic requires redundant number systems to avoid propagation of carries. Adopting arithmetic of this form enables operators to be deeply pipelined and to produce application-level outputs of high significance more quickly than through the use of LSD-first, or a mixture of LSD- and MSD-first, arithmetic. While MSD-first operations, *e.g.* online arithmetic operations, are usually used in a digit-serial fashion, unrolled digit-parallel implementations of these functions also exist [26].

In this paper, we first provide a brief overview of the principles of redundant number representations, MSD-first arithmetic operations, and detail their implementations as well as many machine leaning inferences based thereupon. We then investigate three security protection techniques using redundant number representation, MSD-first arithmetic, and algorithmic approximation analysis. Since iterative methods play an important role in machine learning training [28] and feedforward computation parallelization [37], an exemplary stationary iterative solver is used to demonstrate information hiding abilities following our proposals.

*Corresponding author

GLSVLSI'21, June 22–25, 2021, Virtual Conference and Exhibition

ACM ISBN 978-1-4503-8393-6/21/06…$15.00
https://doi.org/10.1145/3453688.3461753

2 BACKGROUND

2.1 Redundant Number Representation

MSD-first arithmetic requires redundant number systems. As a consequent, propagation of carries is avoided. There are two main redundant digit sets: carry-save (CS) and signed-digit (SD) [9]. The value x is represented in CS form by two bit-vectors called the carry vector vc and sum vector vs with $vc = (vc_{\text{n-1}}, \ldots, vc_1, vc_0)$ and $vc = (vs_{\text{n-1}}, \ldots, vs_1, vs_0)$. Therefore, $x = (vc + vs) \mod 2^n$ [41]. For radix-2 carry-save number representation, redundant digits can be selected from a digit set {0,1,2}. For radix-2 signed-digit number representation, wherein the i^{th} digit of a number x, x_{i}, lies in {-1, 0 ,1}. Each x_{i} corresponds to a pair of bits, x_{i}^{+} and x_{i}^{-}, selected such that $x_{\text{i}} = x_{\text{i}}^{+} - x_{\text{i}}^{-}$ [26].

MSD-first arithmetic would have been possibly employing carry-save representation of numbers [31], however, the number systems most frequently used in the literature are signed-digit due to the flexible choices of redundancy [9]. Avizienis presented a generalization of signed-digit representation and algorithms for addition, multiplication and division [5].

For a signed-digit number, digits are selected from the set

$$D := \left\{-\rho_{\text{low}}, \cdots, -1, 0, 1, \cdots, \rho_{\text{high}}\right\}, \tag{1}$$

where $\rho_{\text{low}} + \rho_{\text{high}} + 1 \geq r$ and $\rho_{\text{low}}, \rho_{\text{high}} \geq 0$. D can be either *symmetric* or *asymmetric*, with different levels of redundancy. In particular, one can have (i) symmetric minimal redundancy with $\rho_{\text{low}} = \rho_{\text{high}} = r/2$, (ii) symmetric maximal redundancy ($\rho_{\text{low}} = \rho_{\text{high}} = r-1$), and (iii) asymmetric redundancy ($\rho_{\text{low}} \neq \rho_{\text{high}}$). The optimal symmetry and redundancy will be dependent on the characteristics and requirements of the target application. In particular, the use of minimal redundancy limits overheads but adds complexity to the conversion of values between nonredundant and redundant forms and the generation of result digits. Maximal redundancy has the opposite features. Maximum redundancy simplifies the selection of result digits, and there is no conversion from conventional to redundant numbers, however, it makes the generation of an arithmetic algorithm harder. The use of symmetric, maximally redundant signed-digit sets is a standard practice for online arithmetic [9].

2.2 MSD-first Arithmetic Operation

Implementation of online and MSD-first algorithms has been considered in many papers, including fixed-point [13, 18, 42, 44], floating-point [43, 49, 50], algorithms in complex number system [30, 38], and MSD-first Coordinate Rotation DIgital Computer (CORDIC) [14, 39], *etc.* Recently, hardware acceleration of MSD-first arithmetic algorithms have become increasingly popular, in particular those with digit-parallel implementations targeting FPGAs [35, 36], multiple operands [21, 45], high radices [20] and variable precision [26, 40]. Shi *et al.* presented an efficient digit-parallel implementation of MSD-first addition and multiplication [35, 36]. Multi-operand multiplication was designed recently, resulting in savings in resource usage and interconnection complexity versus an LSD-first arithmetic equivalent [21]. Moreno *et al.* presented a multi-operand online addition with an efficient conversion of two's complement, carry-save, and signed-digit data [45]. Joseph and Devanathan built a high-radix online multiplier and demonstrated throughput boost and online delay reduction versus classical radix-2 online multipliers, with the sacrifice of a growing power-delay product [20]. Li *et al.* [26] proposed hardware architectures for arbitrary-precision online multiplication and division through hardware reuse. Ercegovac and McIlhenny presented a combined decimal division/square root operation using limited-precision multipliers and adders [10]. Pineiro *et al.* presented an architecture for the computation of logarithm ($\log(x)$), exponential (2^x), and powering (x^y) operations based on an optimized high-radix composite iterative algorithm [33]. Huang and Ercegovac developed a composite online 3-D vector normalization algorithm for computing $v_x/d, v_y/d, v_z/d$ [17], where

$$d = \sqrt{(v_x)^2 + (v_y)^2 + (v_z)^2} \tag{2}$$

For a radix-2 implementation, the overall online delay of a composite scheme of (2) is 5, while a conventional design with a network of standard online operators has an online delay of 11 [17]. Other composite algorithms have been used in implementing rotation factors for matrix transformations [8].

2.3 MSD-first Arithmetic-based ML Inference

With the adaptability of numerical precision for DNN inferences, bit-serial arithmetic operations trivially permit flexibility of precision at runtime for mixed-precision neural networks [1, 4, 7, 15, 23, 34, 46]. Most bit-serial DNN inference engines use conventional fixed or floating arithmetic [4, 7, 23, 34, 46], where the LSDs are computed first. As a consequence, the accumulation of the least significant parts remains insignificant, leading to computational wastage. Alternatively, use of the online arithmetic allows the computations to benefit from early computation of MSDs. Hassan *et al.*'s fine-grained bit-serial DNN inference, employs online arithmetic to generate bits for weights and activations in an MSD-first version. The early generated MSDs can be propagated to the connected online convolution operation [15]. Moreover, Abdelhadi and Shannon focused on online arithmetic-based multi-layer perceptron (MLP) [1]. Online adder trees are used in the ReLU function to detect silent neurons early [1]. Girau and Tisserand presented an implementation of MLP back propagation, where Horner's scheme is used for online computation of $tanh(x)$ [12].

MSD-first arithmetic has been widely used in function evaluation. Bayesian networks can be represented by a characteristic multinomial, and the *probabilistic inference*—a key operation of Bayesian Networks—also requires the function evaluation of multinomial [3]. Algirdas Avizienis and Milos Ercegovac have employed online arithmetic to evaluate Bayesian network multinomial (BNM), converting a multinomial function to a network of online modules on hardware [6]. Online arithmetic modules are chained serially and performed in an overlapped fashion, thereby achieving low interconnect bandwidth and computation latency [3].

3 SECURITY ENHANCEMENT METHODS

To guarantee the trustworthiness of arithmetic IP core reuse when building digital circuits and systems, we investigate security enhancement methods by hiding designers' signature or verification keys in the algorithm and arithmetic implementations. We propose

three information hiding techniques to protect approximate computing systems. The first method takes advantage of the redundant representations inherently existing in the number systems by enforcing them to have specific values corresponding to designer's signatures. A separate digit-detection module can be easily built to retrieve the hidden information. The second method utilizes the fact that many arithmetic operators can be implemented in MSD-first and LSD-first fashions. Instead of always using LSD-first arithmetic algorithms, we investigate functionally identical MSD-first arithmetic operations, which can be substituted in any digital computing applications. We then selectively instantiate these duplicated modules for information hiding. The third technique considers information hiding in the algorithm level. Since iterative algorithms have been widely used in machine learning training and feedforward computation parallelization, we investigate how to perform information hiding in the approximate iterative calculations, while still able to converge to the required accuracy.

3.1 Information Hiding Using Redundant Number System

Consider a finite-precision number $x = 0.996093$ represented in a nonredundant radix-10 number system, we can represent the same number by using a redundant radix-10 number system, where digits are able to be selected from a *redundant digit set* $\{-9, -8, \cdots, 8, 9\}$. In this paper, -9 and $\overline{9}$ are represented equally. Therefore, x can be also represented as $1.00\overline{4}1\overline{1}3$. Herein, LSDs can be used to correct errors that were previously introduced by digits of higher significance, further allowing the computation or data representation to be more precise by producing more MSDs [27].

Consider a 16-bit adder, since every operand digit $x_i,\ i \in [0, 15]$, does not contribute equally to the accuracy of the sum, LSDs could be neglected to save latency and power, given a small error tolerance. When using redundant number systems, digits with less significance can be represented in multiple ways. At the number format conversion stage, we can add a digit-selection module to insert authors' signatures. Taking signed-digit representation as an example, we have the following options.

- Case one: $x_{15} = 0$, where $x_{15} = x_{15}^{+} - x_{15}^{-}$. We have two representation options for $\{x_{15}^{+}, x_{15}^{-}\}$, *i.e.* $\{0, 0\}$ and $\{1, 1\}$. Both options are functionally equivalent. However, the control signal to determine the actual representation can be used as a designer's signature. This can be embedded in the inherent logic of redundant arithmetic operators.
- Case two: $x_{14} = 0$ and $x_{15} = 1$. The choices of $\{x_{14},\ x_{15}\}$ can be either $\{0, 1\}$ or $\{1, \overline{1}\}$. Both representations correspond to the same value and do not introduce any error. We can substitute these representations without requiring additional cost for information hiding.
- We can also consider different representations of a few more LSDs. This leads to additional representation choices that can be used for information hiding. We remark that one or two LSDs can be sufficiently enough to embed simple designers' signatures for security enhancement.

3.2 Information Hiding Using Functionally Equivalent Arithmetic

We employ the inherent logic between functionally-equivalent arithmetic cores to deliberately replace a conventional LSD-first arithmetic core in the datapath with an MSD-first one, and vice versa. For LSD-first arithmetic, fixed-point or floating-point, two's complement representations are popular number systems implemented. To eliminate overflow issues, LSD-first fixed-point arithmetic operations should control its p-bit outputs within the interval of $[-2^{p-1}, 2^{p-1} - 1]$. However, a p-digit MSD-first arithmetic operation can calculate the results in an extended interval of $[-2^{-p}, 2^{-p}]$ [24]. Given the same p-digit operation, use of MSD-first arithmetic calculates several additional values compared to its LSD-first equivalence. Designers can employ this fundamental difference to insert security signatures of the digital computing system.

3.3 Information Hiding Using Algorithmic Approximation Analysis

Feedforward computation in machine learning can be interpreted as solving a triangular system of equations [37], and stationary iterative methods, *e.g.* Jacobi, are used to find the solution [32]. Solving a system $\boldsymbol{Ax} = \boldsymbol{b}$ is to transform it into a linear fixed-point iteration of the form

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{G}\boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}\boldsymbol{b}, \tag{3}$$

with iteration matrix $\boldsymbol{G}$ and $\boldsymbol{M}$ non-singular [27]. An approximant of an iterative method at iteration $k \in \mathbb{N}_{>0}$ is denoted $\boldsymbol{x}^{(k)}$, while its exact result is $\boldsymbol{x}^{*}$.

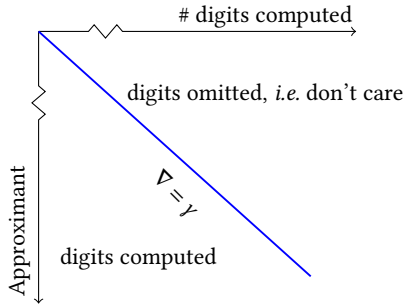

Figure 1: A sketch of LSDs omission, where $\chi = 0$, and the gradient of the don't-care line is controlled by γ.

Recently, a theorem of identifying the number of LSDs to be calculated per iteration within stationary iterative methods has been proposed [25]. This is proven to have no bearing on the chosen method's ability to reach a solution of any accuracy [25]. However, Li *et al.*'s analysis relied on the assumption of an available "budget" of total digits for computation [25]. We now give a more general analysis of LSD growth rate per iteration within stationary iterative methods, thereby enabling the preclusion of don't-care LSD computation, as shown in Fig. 1. We define χ as the number of digits represented in $\hat{\boldsymbol{x}}^{(0)}$, and γ as the number of additional digits allocated per approximant. γ can be used as designers' signature embedded in the approximate computing.

Theorem 3.1 (LSDs omission). *Given a stationary iterative method, if the conditions*

$$\chi \geq 0 \text{ and } \gamma > 0$$

hold, then (3) *will converge with the omission of don't-care digits computation.*

Proof. Let rounding error be $\boldsymbol{\epsilon}_k$, as we propose to via truncation of each approximant, (3) becomes

$$\hat{\boldsymbol{x}}^{(k+1)} = \boldsymbol{G}\hat{\boldsymbol{x}}^{(k)} + \boldsymbol{M}^{-1}\boldsymbol{b} + \boldsymbol{\epsilon}_{k+1}$$

or, expressed per approximant,

$$\hat{\boldsymbol{x}}^{(k+1)} = \boldsymbol{G}^{k+1}\hat{\boldsymbol{x}}^{(0)} + \sum_{i=0}^{k} \boldsymbol{G}^i\boldsymbol{M}^{-1}\boldsymbol{b} + \sum_{i=0}^{k} \boldsymbol{G}^i\boldsymbol{\epsilon}_{k+1-i} \tag{4}$$

from some finite-precision initial guess $\hat{\boldsymbol{x}}^{(0)}$.

Let computation error be $\boldsymbol{e}^{(k)} = \boldsymbol{x}^* - \hat{\boldsymbol{x}}^{(k)}$, which captures errors due to the finiteness of both the iteration count *and* the precision [25],

$$\boldsymbol{e}^{(k+1)} = \boldsymbol{G}^{k+1}\boldsymbol{e}^{(0)} - \sum_{i=0}^{k} \boldsymbol{G}^i\boldsymbol{\epsilon}_{k+1-i}.$$

We wish to minimise this value. Since we cannot minimise $\boldsymbol{e}^{(k)}$ directly, we seek to minimise its upper bound instead. Taking norms, we ensure that $\|\boldsymbol{\epsilon}_{k+1-i}\|_\infty \leq r^{-d_{k+1-i}}$ by controlling the precision of each approximant's computation

$$\begin{aligned}\left\|\boldsymbol{e}^{(k+1)}\right\|_\infty &\leq \left\|\boldsymbol{G}^{k+1}\boldsymbol{e}^{(0)}\right\|_\infty + \sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, \|\boldsymbol{\epsilon}_{k+1-i}\|_\infty \\ &\leq \left\|\boldsymbol{G}^{k+1}\boldsymbol{e}^{(0)}\right\|_\infty + \sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, r^{-d_{k+1-i}}. \end{aligned} \tag{5}$$

Further assume that the inequality $\|\boldsymbol{G}\|_\infty < 1$ holds [16], then

$$\lim_{k\to\infty} \left\|\boldsymbol{G}^{k+1}\boldsymbol{e}^{(0)}\right\|_\infty = \boldsymbol{0}, \tag{6}$$

as proved by Higham [16]. Therefore, the first term of the right-hand side (RHS) in (5) is analysed.

Let the number of radix-r digits represented in approximant k be $d_k = \chi + \gamma k$. Now applying $d_k = \chi + \gamma k$ to the second term of RHS in (5) leads to

$$\begin{aligned}\sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, \|\boldsymbol{\epsilon}_{k+1-i}\|_\infty &\leq \sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, r^{-d_{k+1-i}} \\ &= \sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, r^{-[\chi+\gamma(k+1-i)]} \\ &= r^{-\chi} r^{-\gamma(k+1)} \sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, r^{\gamma i} \\ &= r^{-\chi} r^{-\gamma(k+1)} \sum_{i=0}^{k} (\|\boldsymbol{G}\|_\infty \, r^{\gamma})^i. \end{aligned} \tag{7}$$

Given the definition of χ, $r^{-\chi}$ is a constant. Firstly, if $\|\boldsymbol{G}\|_\infty \, r^{\gamma} = 1$, then the limit of (7) is

$$\begin{aligned}\lim_{k\to\infty} \sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, \|\boldsymbol{\epsilon}_{k+1-i}\|_\infty &\leq \lim_{k\to\infty} r^{-\chi} r^{-\gamma(k+1)} \sum_{i=0}^{k} (\|\boldsymbol{G}\|_\infty \, r^{\gamma})^i \\ &= \lim_{k\to\infty} r^{-\chi} r^{-\gamma(k+1)} \sum_{i=0}^{k} 1^i \\ &= r^{-\chi} \lim_{k\to\infty} (k+1) r^{-\gamma(k+1)} \\ &= r^{-\chi} \lim_{k\to\infty} {}^{(k+1)}/_{r^{\gamma(k+1)}} \qquad (8) \\ &= 0 \text{ (if } \gamma > 0). \qquad (9) \end{aligned}$$

Secondly, if $\|\boldsymbol{G}\|_\infty \, r^{\gamma} \neq 1$, then applying the summation formula of geometric series to (7) results in

$$\begin{aligned}\sum_{i=0}^{k} \|\boldsymbol{G}\|_\infty^i \, \|\boldsymbol{\epsilon}_{k+1-i}\|_\infty &\leq r^{-\chi} r^{-\gamma(k+1)} \sum_{i=0}^{k} (\|\boldsymbol{G}\|_\infty \, r^{\gamma})^i \\ &= r^{-\chi} r^{-\gamma(k+1)} \frac{1 - (\|\boldsymbol{G}\|_\infty \, r^{\gamma})^{k+1}}{1 - \|\boldsymbol{G}\|_\infty \, r^{\gamma}}. \end{aligned} \tag{10}$$

If $\chi > 0$, $\gamma > 0$, and $\|\boldsymbol{G}\|_\infty \in (0, 1)$, and $\frac{r^{-\chi}}{1-\|\boldsymbol{G}\|_\infty r^{\gamma}}$ is a constant, then

$$\lim_{k\to\infty} r^{-\gamma(k+1)} = 0, \tag{11}$$

and

$$\begin{aligned}&\lim_{k\to\infty} r^{-\gamma(k+1)} (\|\boldsymbol{G}\|_\infty \, r^{\gamma})^{k+1} \\ &= \lim_{k\to\infty} \|\boldsymbol{G}\|_\infty^{k+1} \\ &= 0. \end{aligned} \tag{12}$$

From (6), (9), (11) and (12),

$$\lim_{k\to\infty} \left\|\boldsymbol{e}^{(k)}\right\|_\infty = 0.$$

□

Theorem 3.1 indicates iterative calculations following the don't-care line will obtain arbitrary minimised algorithm error. Therefore, we can choose an arbitrary growth rate γ as the security signature embedded in the design of approximate iterative solvers.

4 EMPIRICAL ANALYSIS OF SECURITY APPLICATIONS

In order to investigate how the proposed information hiding techniques can be used to embed security signatures in approximate digital computing systems, we experimented with Jacobi iteration for solving a linear system of the following form as a case study,

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}, \quad \boldsymbol{x}^{(0)} = \boldsymbol{0}, \tag{13}$$

with b_0 and b_1 randomly selected. The termination criterion is $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 < \eta$, with $\eta \in (0, 1]$.

Experimental results show that the proposed security enhancement countermeasures can be used to prevent potential attacks, such as privacy leakage, out-of-control systematic error and error injection in approximate computing [47].

4.1 Security Signature Encoded within Redundant Number Representation

As discussed in Section 3.1, with appropriate redundant number systems, the representation associated with a value is not unique, *i.e.,* the same value can be encoded in more than one way. Table 1 shows the computation of approximants of $\boldsymbol{Ax} = \boldsymbol{b}$ using Jacobi method, along with approximants' binary redundant and non-redundant representations. Use of different number systems, we can produce different security signatures based on the LSDs in approximants. In the common case of radix-2 number systems, non-redundant and redundant representations result in completely different signatures as demonstrated in Table 1.

Table 1: Number representation strategies for the solution of (13) to produce different security signatures.

# Iterations	Approximants in radix-10	Non-redundant representation in radix-2	Redundant representation in radix-2
1	0.5	0.**1**	0.**1**
2	0.25	0.0**1**	0.1$\bar{\mathbf{1}}$
3	0.375	0.01**1**	0.1$\bar{\mathbf{1}}$**1**
4	0.3125	0.010**1**	0.1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$
5	0.34375	0.0101**1**	0.1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$**1**
6	0.328125	0.01010**1**	0.1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$
7	0.3359375	0.0101011	0.1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$**1**
8	0.33203125	0.01010101	0.1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$
9	0.333984375	0.010101011	0.1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$**1**
10	0.3330078125	0.0101010101	0.1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$
Signature based on the lest-significant bit		**1111111111**	**1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$1$\bar{\mathbf{1}}$**

4.2 Security Signature Hidden with Functionally Equivalent Arithmetic

As discussed in Section 3.2, we can substitute functionally equivalent MSD-first and LSD-first arithmetic IP cores, resulting in completely inaccurate computed results to serve as a unique security signature. Table 2 shows the computed results of (13) with different $\|\boldsymbol{b}\|_\infty$ values. We assume that $\|\boldsymbol{b}\|_\infty < 1$ is the input specification to guarantee a working Jacobi solver. If an extreme case $\|\boldsymbol{b}\|_\infty \geq 1$ occurs, it will violate the specification, thus leading to an overflow termination. However, consider that the function-equivalence substitution has been quietly performed at the design stage, an MSD-first operator with a greater overflow tolerance allows us to calculate an inaccurate result, *i.e.* the hidden security signature in our security-enhanced Jacobi solver.

4.3 Security Signature Generated at Approximate Algorithm

As discussed in Section 3.3, the rate of growth in the number of digits allocated per iteration can be used to generate a security signature. Fig. 2 shows the relationship between the growth rate γ per approximate and the required iteration counts for convergence, using Jacobi iteration as a case study. Herein, γ can be used as the designer's signature embedded as the security specification for those who employ this iterative solver for numerical analysis [16] and machine learning applications [37]. Experimental results demonstrate that the less γ is specified at the input, the more iterations will be used to obtain the convergent results. As an iterative solver designer, he/she can randomly specify a γ (*e.g.* 0.5) at the design stage. At the user end, one can only solve a particular linear system, by verifying the trustworthiness of the solver with the correct iteration count (*i.e.* 43).

Table 2: Overflow error for the solution of (13).

Arithmetic algorithm	$\|\boldsymbol{b}\|_\infty \geq 1$	$\|\boldsymbol{b}\|_\infty < 1$
LSD-first	Termination (overflow)	0.3333333333 (convergence)
MSD-first	Inaccurate results (**signature**)	0.3333333333 (convergence)

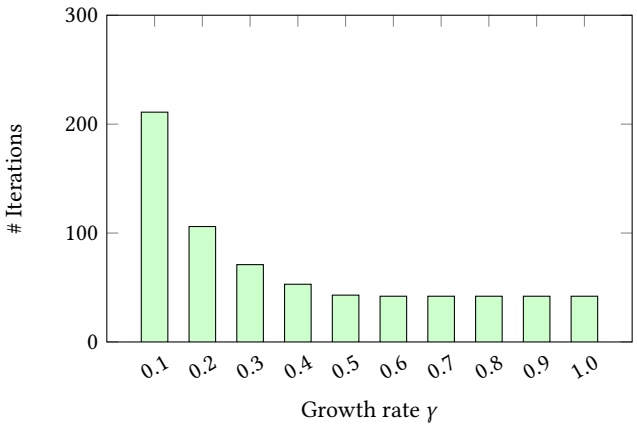

Figure 2: How the growth rate γ affects # iterations required to converge for the solution of (13).

5 CONCLUSION

We propose the first set of non-traditional protection mechanisms for potential machine computing inferences. Our proposals are evaluated using the Jacobi method implemented in different number systems, arithmetic algorithms and LSD-preclusion strategies. Three kinds of signatures are embedded into approximate computing systems for security protection. The proposed techniques can be used to protect both existing and potential machine learning applications, with a particular focus on iterative solvers. In the future, we will attempt to evaluate our principle to a complete machine learning systems, such as neural networks and K-means clustering. We are also particularly keen to develop EDA tools for approximate computing protection.

6 ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China under Grant U20A20202, 61874042, and the the Hunan Natural Science Foundation for Distinguished Young Scholars under Grant No. 2020JJ2010.

REFERENCES

[1] Ameer MS Abdelhadi and Lesley Shannon. 2019. Revisiting deep learning parallelism: Fine-grained inference engine utilizing online arithmetic. In *IEEE International Conference on Field-Programmable Technology*. 383–386.

[2] Achilles. [n.d.]. https://blog.checkpoint.com/2020/08/06/achilles-small-chip-big-peril/.

[3] Pavan Adharapurapu and Miloš Ercegovac. 2005. A linear-system operator based scheme for evaluation of multinomials. In *IEEE Symposium on Computer Arithmetic*. 249–256.

[4] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. 2017. Bit-pragmatic deep neural network computing. In *IEEE/ACM International Symposium on Microarchitecture*. 382–394.

[5] Algirdas Avizienis. 1961. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on electronic computers* EC-10, 3 (1961), 389–400.

[6] Pouya Dormiani, David Omoto, Pavan Adharapurapu, and Milos D Ercegovac. 2005. A design of online scheme for evaluation of multinomials. In *Advanced Signal Processing Algorithms, Architectures, and Implementations XV*. 59100S.

[7] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. 2018. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *ACM/IEEE International Symposium on Computer Architecture)*. 383–396.

[8] M.D. Ercegovac and T. Lang. 1988. On-line scheme for computing rotation factors. *Journal Parallel and Distributed Computing* 5, 6 (1988), 209–227.

[9] Milos D Ercegovac and Tomas Lang. 2004. *Digital arithmetic*. Elsevier.

[10] Miloš D Ercegovac and Robert McIlhenny. 2010. Design and FPGA implementation of radix-10 combined division/square root algorithm with limited precision primitives. In *IEEE Asilomar Conference on Signals, Systems and Computers*. 87–91.

[11] Mingze Gao, Qian Wang, Akshaya S Kankanhalli Nagendra, and Gang Qu. 2017. A novel data format for approximate arithmetic computing. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 390–395.

[12] B. Girau and A. Tisserand. 1996. On-Line arithmetic-based reprogrammable hardware implementation of multilayer perceptron back-propagation. In *IEEE International Conference on Microelectronics for Neural Networks*. 168–175.

[13] A. Gorji-Sinaki and M.D. Ercegovac. 1981. Design of a digit-slice on-line arithmetic unit. In *IEEE Symposium on Computer Arithmetic*. 72–80.

[14] Robert Hamill, John V McCanny, and Richard L Walke. 2000. Online CORDIC algorithm and VLSI architecture for implementing QR-array processors. *IEEE Transactions on Signal Processing* 48, 2 (2000), 592–598.

[15] Abdus Sami Hassan, Tooba Arifeen, and Jeong-A Lee. 2020. Data footprint reduction in DNN inference by sensitivity-controlled approximations with online arithmetic. In *IEEE International Conference on Digital System Design*. 534–541.

[16] Nicholas J Higham. 2002. *Accuracy and Stability of Numerical Algorithms*. SIAM.

[17] Zhijun Huang and Milos D Ercegovac. 2001. FPGA implementation of pipelined on-line scheme for 3-D vector normalization. In *IEEE Symposium on Field-Programmable Custom Computing Machines*. 61–70.

[18] Mary Jane Irwin. 1977. *An arithmetic unit for on-line computation*. PhD Dissertation. University of Illinois at Urbana-Champaign.

[19] Honglan Jiang, Francisco Javier Hernandez Santiago, Hai Mo, Leibo Liu, and Jie Han. 2020. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proc. IEEE* (2020).

[20] Georgina Binoy Joseph and R Devanathan. 2016. Design and analysis of online arithmetic operators for streaming data in FPGAs. *International Journal of Applied Engineering Research* 11, 3 (2016), 375–390.

[21] Georgina Binoy Joseph and R Devanathan. 2018. Algorithms for multiplierless multiple constant multiplication in online arithmetic. *Circuits, Systems, and Signal Processing* 37, 11 (2018), 5127–5142.

[22] Georgina Binoy Joseph and R Devanathan. 2018. Computation of design metrics in online arithmetic networks. In *IEEE International Conference on Emerging Trends and Innovations In Engineering And Technological Research*. 1–4.

[23] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *IEEE/ACM International Symposium on Microarchitecture*. 1–12.

[24] He Li, Ameer Abdelhadi, Runbin Shi, Jiliang Zhang, and Qiang Liu. 2021. Adversarial Hardware with Functional and Topological Camouflage. *IEEE Transactions on Circuits and Systems-II: Express Briefs* (2021).

[25] He Li, James J Davis, John Wickerson, and George A Constantinides. 2018. Digit elision for arbitrary-accuracy iterative computation. In *IEEE Symposium on Computer Arithmetic*. 107–114.

[26] H. Li, J. J. Davis, J. Wickerson, and G. A. Constantinides. 2019. ARCHITECT: Arbitrary-precision hardware with digit elision for efficient iterative compute. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 2 (2019), 516–529.

[27] He Li, Ian McInerney, James J. Davis, and George A. Constantinides. 2020. Digit stability inference for iterative methods using redundant number representation. *IEEE Trans. Comput.* (2020).

[28] Qiang Liu, Jia Liu, Ruoyu Sang, Jiajun Li, Tao Zhang, and Qijun Zhang. 2018. Fast neural network training on FPGA using quasi-newton optimization method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 8 (2018), 1575–1579.

[29] Weiqiang Liu, Chongyan Gu, Máire O'Neill, Gang Qu, Paolo Montuschi, and Fabrizio Lombardi. 2020. Security in Approximate Computing and Approximate Computing for Security: Challenges and Opportunities. *Proc. IEEE* 108, 12 (2020), 2214–2231.

[30] Robert McIlhenny. 2002. *Complex number on-line arithmetic for reconfigurable hardware: Algorithms, implementations, and applications*. PhD Dissertation. University of California Los Angeles.

[31] J-M Muller. 1994. Some characterizations of functions computable in on-line arithmetic. *IEEE Trans. Comput.* 43, 6 (1994), 752–755.

[32] James M Ortega and Werner C Rheinboldt. 2000. *Iterative solution of nonlinear equations in several variables*. SIAM.

[33] J-A Pineiro, Milos D Ercegovac, and Javier D Bruguera. 2004. Algorithm and architecture for logarithm, exponential, and powering computation. *IEEE Trans. Comput.* 53, 9 (2004), 1085–1096.

[34] Sayeh Sharify, Alberto Delmas Lascorz, Kevin Siu, Patrick Judd, and Andreas Moshovos. 2018. Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks. In *ACM/ESDA/IEEE Design Automation Conference*. 1–6.

[35] K. Shi, D. Boland, and G. A. Constantinides. 2014. Efficient FPGA implementation of digit parallel online arithmetic operators. In *IEEE International Conference on Field Programmable Technology*. 115–122.

[36] Kan Shi, David Boland, Edward Stott, Samuel Bayliss, and George A Constantinides. 2014. Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs. In *ACM/EDAC/IEEE Design Automation Conference*. 1–6.

[37] Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. 2020. Nonlinear equation solving: A faster alternative to feedforward computation. *arXiv preprint arXiv:2002.03629* (2020).

[38] Milena Svobodova, Edita Pelantova, Marta Pavelka, and Christiane Frougny. 2019. On-line algorithms for multiplication and division in real and complex numeration systems. *Discrete Mathematics & Theoretical Computer Science* 21, 3 (2019).

[39] Naofumi Takagi, Tohru Asada, and Shuzo Yajima. 1991. Redundant CORDIC methods with a constant scale factor for sine and cosine computation. *IEEE Trans. Comput.* 40, 9 (1991), 989–995.

[40] AF Tenca and MD Ercegovac. 1997. A high-radix multiplier design for variable long-precision computations. In *IEEE Asilomar Conference on Signals, Systems and Computers*. 1173–1177.

[41] Alexandre F Tenca, Song Park, and Lo'ai A Tawalbeh. 2006. Carry-save representation is shift-unsafe: The problem and its solution. *IEEE Trans. Comput.* 55, 5 (2006), 630–635.

[42] P.K.-G. Tu. 1990. *On-line arithmetic algorithms for efficient implementation*. PhD Dissertation. University of California Los Angeles.

[43] P.L.-G. Tu and M.D. Ercegovac. 1991. Gate array implementation of on-line algorithms for floating-point operations. *Journal of VLSI Signal Processing* 3, 4 (1991), 307–317.

[44] D. Tullsen and M.D. Ercegovac. 1986. Design and implementation of an on-line algorithm. In *SPIE Conference on Real-Time Signal Processing*. 92–99.

[45] Julio Villalba, Tomas Lang, and Javier Hormigo. 2011. Radix-2 multioperand and multiformat streaming online addition. *IEEE Trans. Comput.* 61, 6 (2011), 790–803.

[46] Jingcheng Wang, Xiaowei Wang, Charles Eckert, Arun Subramaniyan, Reetuparna Das, David Blaauw, and Dennis Sylvester. 2019. A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing. *IEEE Journal of Solid-State Circuits* 55, 1 (2019), 76–86.

[47] Ye Wang, Jian Dong, Qian Xu, Zhaojun Lu, and Gang Qu. 2020. Is It Approximate Computing or Malicious Computing?. In *ACM International conferences on Great Lakes Symposium on VLSI*. 333–338.

[48] Ye Wang, Qian Xu, Gang Qu, and Jian Dong. 2019. Information hiding behind approximate computation. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. 405–410.

[49] Osaaki Watanuki and Milos D Ercegovac. 1981. Floating-point on-line arithmetic: Algorithms. In *IEEE Symposium on Computer Arithmetic*. 81–86.

[50] Osaaki Watanuki and Milos D. Ercegovac. 1983. Error analysis of certain floating-point on-line algorithms. *IEEE Trans. Comput.* C-32, 4 (1983), 352–358.

[51] Valerii Zhabin and Valentina Zhabina. 2020. Methods of on-line computation acceleration in systems with direct connection between units. In *IEEE International Conference on Dependable Systems, Services and Technologies*. 356–362.