scientific reports

Check for updates

OPEN

A flexible and fast digital twin for RRAM systems applied for training resilient neural networks

Markus Fritscher[1,2]✉, Simranjeet Singh[3], Tommaso Rizzi[1], Andrea Baroni[1], Daniel Reiser[4], Maen Mallah[5], David Hartmann[5], Ankit Bende[6], Tim Kempen[6], Max Uhlmann[1], Gerhard Kahmen[1,2], Dietmar Fey[7], Vikas Rana[3], Stephan Menzel[6], Marc Reichenbach[5], Milos Krstic[1,8], Farhad Merchant[9,10] & Christian Wenger[1,2]

Resistive Random Access Memory (RRAM) has gained considerable momentum due to its non-volatility and energy efficiency. Material and device scientists have been proposing novel material stacks that can mimic the "ideal memristor" which can deliver performance, energy efficiency, reliability and accuracy. However, designing RRAM-based systems is challenging. Engineering a new material stack, designing a device, and experimenting takes significant time for material and device researchers. Furthermore, the acceptability of the device is ultimately decided at the system level. We see a gap here where there is a need for facilitating material and device researchers with a "push button" modeling framework that allows to evaluate the efficacy of the device at system level during early device design stages. Speed, accuracy, and adaptability are the fundamental requirements of this modelling framework. In this paper, we propose a digital twin (DT)-like modeling framework that automatically creates RRAM device models from device measurement data. Furthermore, the model incorporates the peripheral circuit to ensure accurate energy and performance evaluations. We demonstrate the DT generation and DT usage for multiple RRAM technologies and applications and illustrate the achieved performance of our GPU implementation. We conclude with the application of our modeling approach to measurement data from two distinct fabricated devices, validating its effectiveness in a neural network processing an Electrocardiogram (ECG) dataset and incorporating Fault Aware Training (FAT).

Keywords RRAM, Modelling, GPU, Digital twin, ANN, FAT

Resistive Random-Access Memories (RRAMs) are promising due to their built-in non-volatility and since they can perform Vector-Matrix-Multiplication (VMM) of arbitrarily large inputs with constant computational complexity $\mathcal{O}(1)$. Large crossbars can be constructed, implementing both memory and also serving as a compute element, breaching the boundaries of traditional von-Neumann architectures. Additionally, multiple devices can be stacked vertically, further increasing overall design density[1]. These attributes render them an ideal candidate for the implementation of Artificial Neural Networks (ANNs), which involve an ever-increasing computational complexity. Unfortunately, the current generation of devices suffers from device variation, reliability issues are still being addressed at the fabrication level. A device which was programmed to store a given value a might actually store a different value. This field of research is moving fast, new generations and types of devices are proposed frequently and researchers try to optimize the corresponding programming algorithms and surrounding circuits. This poses a challenge for system-level RRAM research which strives to develop systems which can cope with the underlying device variation since these efforts require models which are both fast to evaluate and true to device behaviour.

Surprisingly, despite attempts to address this challenge, we found that there is a lack of models which are (a) fast to simulate, (b) accurate and (c) easy to adapt to new devices and/or programming algorithms. While various modelling approaches exist, these tend to be lacking in one or more of these criteria, hindering system-

[1]IHP - Leibniz Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany. [2]BTU Cottbus-Senftenberg, Cottbus, Germany. [3]Indian Institute of Technology Bombay, Mumbai, India. [4]University of Rostock, Rostock, Germany. [5]Fraunhofer IIS, Erlangen, Germany. [6]Forschungszentrum Jülich GmbH, Jülich, Germany. [7]FAU Erlangen-Nürnberg, Erlangen, Germany. [8]University of Potsdam, Potsdam, Germany. [9]Newcastle University, Newcastle upon Tyne, UK. [10]Cognigron and Bernoulli Institute, University of Groningen, Groningen, The Netherlands. ✉email: fritscher@ihp-microelectronics.com

level development and - subsequently - adoption. These efforts are fundamental for RRAM research since recent research indicates that e.g. individual instances of artificial neural networks react quite differently to device variations[2]. A given device variation might be acceptable for network *a* while a given network *b* might struggle to maintain reasonable accuracy. Furthermore, different programming algorithms might have severe impact on the overall fraction of erroneously stored data in a memristive crossbar[3–5]. Fault-Aware Training (FAT)[6] (sometimes also referred to as Hardware Aware Training (HAT)) aims to enhance ANN resilience by injecting device faults into the ANN during the training procedure, but is reliant on accurate variability modelling. Unfortunately, existing frameworks have encountered performance issues, requiring significant amounts of compute time on recent networks[7].

In this work, we propose a solution to the before-mentioned challenge. We implemented an end-to-end modelling flow for the generation and usage of Digital Twins (DTs) in a fully automated manner, as depicted in Fig. 1. A new model for a modified fabrication approach, individual wafer or programming algorithm can be created effortlessly, with the push of a button, by inputting measurement data into the DT generation block (Fig. 1A). These models are application-agnostic and can be integrated into various environments (Fig. 1B) to evaluate an application and architecture. Additionally, we developed a GPU backend to support the needs for applications such as FAT. We have taken great care to ensure that this model can undergo simulations at high speed, enabling the evaluation of large systems at an effective simulation speed of multiple $\mathrm{GB\,s^{-1}}$ when run on a GPU. This modeling process encompasses both the RRAM devices and the surrounding circuitry.

We showcase the versatility and effectiveness of our proposed model through its application to three distinct domains. Firstly, leveraging the inherent non-volatility of RRAMs, we explore their potential application as memory, where ongoing research focuses on diverse error correction approaches. Secondly, we illustrate the suitability of RRAMs for real-time applications at the Edge, particularly in tasks like Edge detection. Lastly, we investigate the suitability of two distinct different device technologies for the implementation of ANN. We derived measurement data for $\mathrm{HfO_2}$ based devices fabricated at IHP (Leibniz-Institut für innovative Mikroelektronik) and a $\mathrm{TaO_x}$ based device fabricated at FZJ (Forschungszentrum Jülich) respectively and apply our proposed modeling approach to both to emphasize its flexibility. Subsequently, we use the proposed approach to implement an ANN processing an ECG dataset and incorporate FAT to render the network implementations resilient to individual device fault characteristics. Both network implementations ultimately yield reasonable classification accuracy, indicating the relevance and importance of this work.

This is not the first work attempting to model memristive devices, significant efforts have been spent on creating simulation environments for both memory and compute applications of memristive devices at different levels of abstraction: At the lower levels of abstraction researchers have devised atomistic[8], monte carlo/finite element mthod[9] and compact circuit models[10]. However, adequately defining the involved partial differential equations has proven to be difficult[11]. This ultimately yields simulation times prohibiting a reasonable investigation when attempting large-scale simulations. At the higher levels of abstraction researchers have devised many (partially domain specific) simulators. NVSIM[12], which extends the Cacti[13] cache simulator with Memristor support, is a tool to evaluate many different memristive technologies. However, the possibilities to fit models to a given device (including variation) are both rather limited and work-intensive. DNN+NeuroSim[14], which has been created for the simulation of ANN embedding RRAM, is an adaptation of Neurosim[15] to the PyTorch framework[16]. However, two issues hinder its usage: Firstly, significant parts of the Neurosim code run on CPU, leading to training times in the tens of hours[7]. Secondly, adapting the simulation model to a new wafer of devices is a tedious process. Mnsim[17] allows for detailed architecture investigations, but is limited regarding RRAM device variation, since it only supports a single uniform distribution to model different variations. DL-RSIM[18] supports sophisticated mapping strategies but only supports inference simulations and is limited regarding variance modelling. Additionally, it does not support modelling of the surrounding circuitry. Memtorch[19] extends the PyTorch NN framework with Memristor models. However, it is implemented as a "wrapper" to traditional RRAM models such as VTEAM[20] or the Stanford PKU model[10], which involve a multitude of device parameters which need to be adapted to new devices manually. While data-driven models such as the implementation developed by Messaris et al.[21] have been integrated recently, manual fitting is still required.

To the best of the authors knowledge none of the existing works can achieve simulation speeds which are competitive with the results reported in this work while being as easy to fit to new measurement data and/or a different application domain. Furthermore, limiting RRAM models to the mere device itself lacks substantial information about the surrounding circuit, both regarding area and power draw. Subsequently, we integrated the capability to model surrounding circuitry. This approach has been used successfully to design and evaluate a

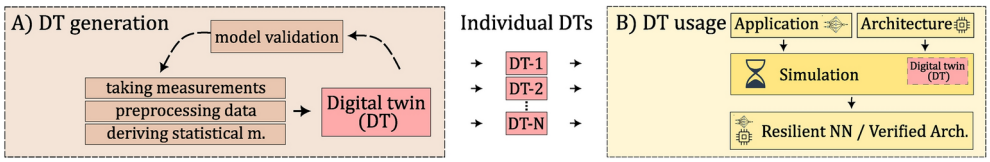


Fig. 1. Proposed approach. Generating Digital Twins from measurement data is fully automated (brown box). This can be used to create many individual Digital Twins, each covering a different variability aspect (middle). These can be integrated seamlessly in various simulation environments in order to validate a given application or system architecture (yellow box).

RISC-V[22] based ANN-accelerator platform embedding RRAM-crossbars[3] and could easily be adapted to other applications such as stochastic computing[23] or random number generation[8].

Results

Overview—fully automated flow

One of the key contributions of the proposed modelling technique lies within the fully automated flow from raw measurements to system-level evaluation; A new generation of RRAM devices, a different wafer or a novel programming algorithm can be modelled by pressing a single button (Fig. 1A), allowing for a fast evaluation within an application or architecture (Fig. 1B). We have been able to successfully use the proposed flow to create individual digital twins for each combination of programming algorithm and type of RRAM device (HfO_2, TaO_x). Individual sources of device variation can be included by providing corresponding measurements. We used both measurements of 4096 devices and the parametrization of an implemented ADC as the ground truth for these modelling efforts. Creating 20 individual DTs for the 20 steps of a given programming algorithm from raw data for a given device technology is fully automated and takes about 5 minutes using a single CPU core of an Intel i9-12900K CPU. This can be parallelized easily if more performance is needed. These models do not need any specific adaption to be integrated into the different environments outlined later. The remainder of this section will provide details regarding utilized devices, our modelling approach and the achieved performance.

Generation of digital twins

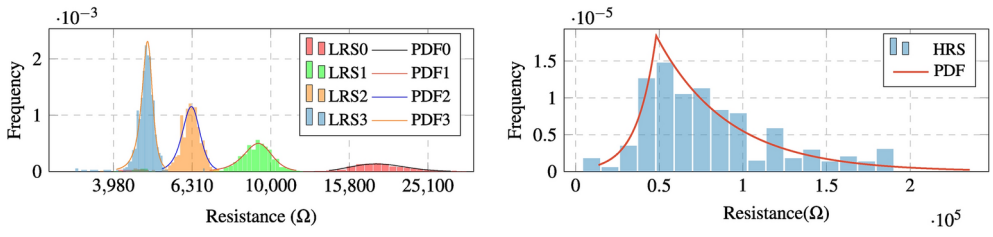
Electrical characterization of HfO_2 *devices*

A significant number of devices have to be characterized to adequately capture the device behavioural model. To ease this measurement phase, the HfO_2 devices have been put in series with a transistor, and the resulting 1T1R cell has been deployed in a 4 kbit array. The transistor serves a twofold purpose: Firstly, selecting the cell inside the crossbar and secondly fixing the compliance current during the SET operations. The array has been divided into 4 batches of 1024 cells each. Each of these sub arrays has been programmed to a different nominal conductance level (three low resistive states (LRS) and one high resistive state (HRS)) leading to currents of $10\,\mu A$ (LRS0), $20\,\mu A$ (LRS1), $30\,\mu A$ (LRS2) and $40\,\mu A$ (LRS3). The devices have been cycled for 1000 times between that state and a HRS with a target conductance of $5\,\mu A$. The cells have been programmed with an Incremental Step Pulse with Verify Algorithm (ISPVA) scheme[5]. This algorithm has been selected to serve as an example for creating a digital twin, future work will be to use this approach to select the best writing algorithm for an individual task. The ISPVA was performed using the following parameters:

- Forming voltage sweep: 2.0 V to 5.0 V using steps of 0.01 V and a pulse duration of $10\,\mu s$.
- Reset/set voltage sweep: 0.5 V to 2.0 V using steps of 0.1 V and a pulse duration of $1\,\mu s$.
- Read operation: A single 0.2 V pulse. The results of this operation are depicted in Fig. 2. Albeit each device which was meant to be programmed to the same target state was programmed using the same current target the actual implementation yields a distribution of currents. The D2D and C2C variability of HfO_2 devices have been investigated and compared in[24,25] (e.g. Ref.[25], Fig. 6b). The authors found that the C2C variation might play a larger role than the D2D variation for this type of device.

Electrical characterization of TaO_x *devices*

We fabricated and characterized 35 devices at FZJ, using the TaO_x stack structure in 0T1R (passive) configuration. We use a different set of measurements as described in Fig. 9 (later section) to model this device to emphasize our approach's flexibility. A compliance current of $500\,\mu A$ is enforced externally in order to make up for the missing transistor. We derive the current state of a given device by computing the inverse of the I-V curve slope at low voltages (between 0.1 V and 0.3 V) of the SET hysteresis; The inverse slope of the positive part yields the HRS resistance while the inverse slope of the negative part yields the LRS resistance. The results of these measurements are depicted in Fig. 4a. These measurement campaigns allow extracting information about two stochastic effects for this type of devices, namely D2D and C2C variabilities for both devices. A thorough investigation of the D2D and C2C variability of the TaO_x devices has been done in[26]. The authors found that the D2D and C2C variations follow the same order of magnitude.



(a) LRS0-LRS3 **(b)** HRS

Fig. 2. This figure depicts one individual set of underlying measurement data (histograms) and a corresponding fitted PDF for the HfO_2 device after using the ISPVA algorithm to program a 1T1R crossbar of 4096 devices to five distinct states.

Model for the analog digital conversion

A simple Flash-ADC has been designed for IHP SG13S as depicted in Fig. 3a. This circuit serves as an example how circuit parameters can be integrated into the Digital Twin. It consists of a voltage divider between the RRAM devices and a shunt resistor. Five operational amplifiers in comparator configuration compare the current voltage at $V_{divider}$ to a reference voltage and switch their respecting output to 1 when this threshold is reached. This ADC can be calibrated to a given device by appropriately modifying $V_{ref(1..5)}$. We perform DC simulations to relate the resistance to currently-read-out state and export this information for later usage within the Digital Twin (Fig. 3b). We relate the ADC input voltage $V_{divider}$ which is present at the voltage divider between R_{mem} and R_{ref} (dotted line) for a given RRAM device resistance to the different state conversion outputs. The outputs out_1 to out_5 switch to a logical one when $V_{divider}$ is above $V_{ref(1..5)}$ (other lines).

Figure 3c depicts the current consumption of the entire circuitry for each given state. It becomes apparent that the higher resistance states should be used predominantly to save on overall power consumption. This data can be extracted as well to allow for a "weight storage power consumption" estimate for a given NN. An actual implementation would most likely embed a more sophisticated ADC circuit to mitigate the comparatively large power consumption.

We extract the results of the simulations depicted in Fig. 3b and c for integration it into the DT. This enables our DT to relate a given resistance of a device (or the outputs of a crossbar) to a digital ADC output as it would occur for a given implementation. Modeling a different ADC merely requires rerunning the beforementioned simulations.

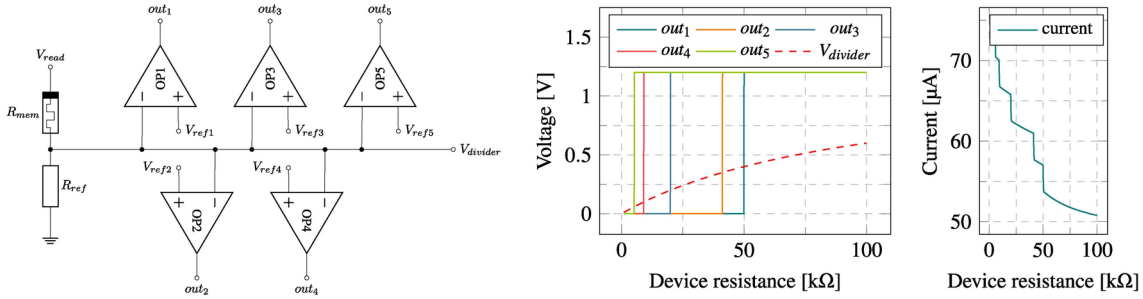
Statistical model for HfO_2 *based devices*

Firstly, we prepared scripts to pre-process the measurement data coming from the utilized measurement equipment. Subsequently, we derive the individual DT as follows; The percentage of stuck-at-HRS devices can be derived automatically by counting the occurrences of high-resistive (e.g. x$\geq 200\,\mathrm{M\Omega}$) measurements and dividing it by the total number of devices. The percentage of stuck-at-LRS3 devices can be derived by doing the reset measurement procedure and counting the number of shorted devices. Doing the before-mentioned measurements and using SciPy to fit about a hundred different distributions to the LRS0 state leads to a set of distributions approximating the raw data. While some distributions appear to be a good fit, others do not manage to approximate the data well. This set of fittings serves as the basic block for our modeling efforts; Subsequently, the best fit is selected by calculating and selecting the fit with the smallest Root Mean Squared Error (RMSE) and its parameters are exported. This is combined with the ADC model we derived as described in the previous section and can be imported within an arbitrary simulation framework by writing an appropriate import wrapper. This can be achieved with a few lines of code since the modelling logic itself is already built into the model. We provide appropriate wrappers for C/C++, Python running on CPU, Python running on a NVIDIA GPU and PyTorch. Creating a wrapper for a specific programming language takes little effort since we put emphasis on using as few language-specific features as possible.

Statistical model for combined TaO_x *based devices*

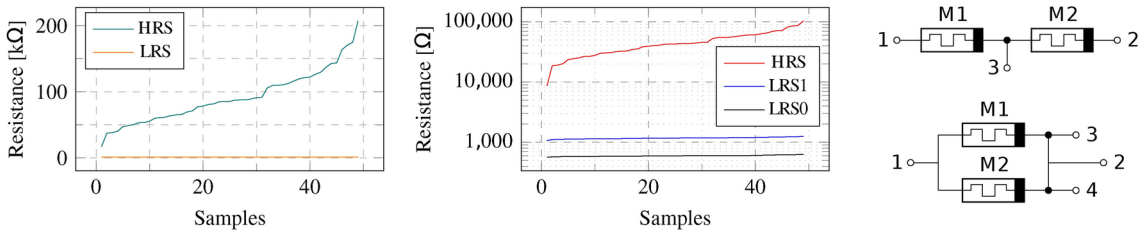
The measurements taken for the TaO_x devices are shown in Fig. 4a. Each device has been programmed to its LRS and HRS (see previous section for details). As one can see, the HRS shows significant variance, rendering the definition of two separate LRS states difficult. Such a high HRS variance in TaO_x devices has previously been reported and investigated by others[27,28]. However, when dealing with NNs, providing an uneven number of states eases the training procedure. Subsequently, we use two devices to increase the number of individual states which can be stored reliably (Fig. 4c). Similar to resistors, the equivalent resistance of serial memristors is given by Eq. (1) while the equivalent resistance of parallel devices is given by Eq. (2).

$$R_{M1+M2} = R_{M1} + R_{M2} \tag{1}$$



(a) Simple ADC circuit. **(b)** DC characterization: input/states **(c)** Current consumption

Fig. 3. (**a**) Depicts a simple non-optimized Flash-ADC circuit. (**b**) Depicts the result of running DC simulations for this schematic, ranging the resistance of a theoretical RRAM device from $0\,\mathrm{k\Omega}$ to $100\,\mathrm{k\Omega}$. (**c**) Depicts the corresponding current, indicating that the power consumption strongly depends on the device's resistance.



(a) HRS/LRS measurements **(b)** Derived states **(c)** Serial/Parallel configuration

Fig. 4. HRS/LRS characteristics of TaO_x devices (**a**) and simulation of three states by using two devices in parallel (**b**). The serial and parallel configuration of memristors (**c**) allows the programming of more than two states. The additional pins 3 and 4 are required to facilitate the forming and programming of individual devices.

$$R_{M1||M2} = \frac{R_{M1} \times R_{M2}}{R_{M1} + R_{M2}} \tag{2}$$

Given that the HRS of these devices is spread significantly, only the parallel implementation yields reasonably separable states. Subsequently, we selected the parallel version for our evaluations. The derived states are depicted in Fig. 4b, they are realized by combining both *LRS* states for LRS1, one *HRS* and one *LRS* state for LRS0 and by setting both devices to *HRS* for the HRS state. Integrating these approaches into our proposed modelling methodology was achieved by pre-processing the measurement data accordingly. Since these devices are ultimately combined on a simulative level we performed measurements of parallel devices in order to validate these results (details outlined in Supplementals/Fig. 3).

Integrating digital twins into application-specific simulators

This section describes key aspects of rendering the proposed model fast enough to support both large crossbars and the evaluation of large systems. The model itself is application agnostic, a given Digital Twin can be seamlessly evaluated across application domains. More details are described in the Methodology section.

Fundamental concept

The fundamental concept can be summarized as follows: Firstly, the to-be-written state is selected by the application. Next, the model samples the "baseline" from the first to-be-used distribution which has been found for a given variation type (typically device-to device variation), including the mean. Further variations are added arithmetically by ignoring the mean of the corresponding distribution. The resulting energy usage (provided by the model annotations) is provided to the application. Secondly, during the read operation, the simulated resistance value is compared to the parametrized ADC, determining which digitized state this would correlate to, again providing the energy usage for the system-level evaluation. This procedure has to be repeated for each individual device. We found that this can be implemented on GPU without major data dependencies by spawning an individual CUDA thread per RRAM device. This ultimately leads to impressive overall simulation performance while only moderately taxing GPU memory (up to about 1.25×10^9 devices fit into the $80\,\mathrm{GB}$ of HBM provided by a recent GPGPU). This approach does not directly transfer to CPUs (since they do not allow for spawning massive amounts of threads), but constructs such as the *map* operator can be used to have strong control over thread scheduling.

Variation modules

There are two fundamental approaches to integrate a digital twin containing multiple variation modules into a simulation environment. Firstly, one could sample each kind of distribution for each kind of device and each individual variation module and store the result in memory. This is advisable when a lot of operations are required since the computational complexity of retrieving the current state of the memristor is $\sim \mathcal{O}(1)$; Unfortunately, the amount of required memory increases exponentially with the number of variation modules, it is equivalent to N^m, where N depicts the required number of memristors while m depicts the number of variation modules. This will quickly drain the memory of a recent GPU, even for moderately sized networks. Fortunately, pre-initializing all these variation modules is unnecessary. In contrast, it is imperative to pre-initialize some of the variation modules (like stuck-at and D2D variations); Others can be sampled on access, vastly reducing the memory footprint. Since the D2D and the stuck-at matrices can be combined, this reduces the overall memory requirements to $m \times N$ while only moderately increasing the computational complexity to $\sim \mathcal{O}(NlogN)$, which resembles the worst-case for sampling discrete distributions. This results in a vastly reduced memory footprint, which is well within the limits of current GPUs. A graphical representation of the corresponding memory footprints is provided in the supplemental (Fig. 4).

We implemented a prototype to evaluate whether this holds for an actual simulation. We evaluated its performance by programming to 1×10^6 RRAM devices using an increasing number of variation modules running on a single CPU core. This leads to the real-world relative simulation performance as shown in Fig. 5c.

Adding multiple variation modules leads to a moderate decrease in overall performance, which aligns with the earlier theoretical estimations. Subsequently, this approach allows for the investigation of large systems.

Performance and accuracy of the proposed model

Memory simulator

We have implemented a memory simulator using the proposed concept. A memory block is accessed randomly, setting individual RRAM devices to individual states. In order to achieve sufficient performance we ported our model to CUDA and ran it on a NVIDIA H100 GPU. The achieved equivalent simulation speed for differently sized memory blocks is depicted in Fig. 5a. We ran 100 individual simulations per matrix size and depict the deviations as error bars. The simulation performance is lower for smaller memories, probably as a result of these not fully utilizing the GPU. However, the simulation speed scales linearly with the number of devices until the GPU is fully utilized. This simulation speed is similar to the performance of a recent SSD, enabling thorough application evaluations.

Real-time application

We have implemented an application which uses individual (simulated) HfO_2-based MVM blocks to perform edge detection using Sobel filter kernels[29]. The input is given by a video camera. A different number of HfO_2 based RRAM crossbars is utilized, splitting up the input image into differently sized partitions. We use this setup to evaluate both differently sized crossbars and different steps of the ISPVA programming algorithm regarding the quality of the edge detection. The output of the calculations are shown on the laptop display. The resulting Frames Per Second (FPS) when using two CPU cores on a recent laptop are depicted in Fig. 5b. For a small number of crossbars the performance is capped by the 30 FPS provided by the camera. Apparently the implemented model is fast enough to evaluate real-time applications.

Simulation performance impact of combining multiple variation modules

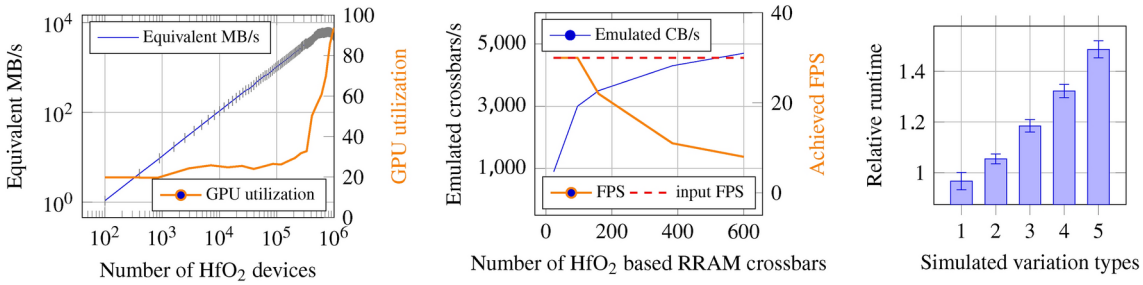
RRAM devices tend to be affected by multiple types of variability (such as e.g. device-to-device and cycle-to-cycle variations) simultaneously, which need to be taken into consideration for system-level evaluations. Subsequently, multiple types of device variability need to be simulated simultaneously, ultimately slowing down the achieved simulation speed. We performed multiple simulations which program 1×10^6 RRAM devices using an increasing number of variation modules running on a single CPU core to evaluate the performance impact. This leads to the real-world relative simulation performance as shown in Fig. 5c. Simulating 5 different types of device variation (e.g. D2D (1), C2C (2), drift (3), retention (4) and read disturb (5)) simultaneously leads to a moderate increase of simulation time of about $1.5\times$.

Validating the model

This section describes the validation of the D2D module using a separate set of data received from HfO_2 based devices.

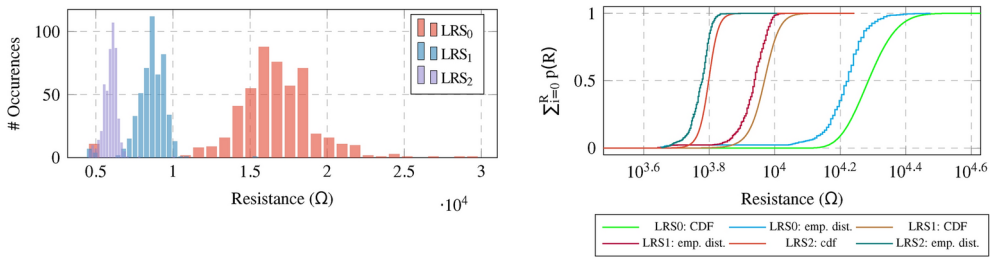
Firstly, a model has been created using measurement data from a specific wafer of HfO_2 devices "wafer a"; A multitude of devices has been programmed, data has been extracted, and a Digital Twin has been derived. Subsequently, in order to validate this model, 500 weights each, containing the states-*1*, *0* and *1*, have been programmed to a total of 1500 devices on a different wafer, "wafer b". This equals to programming LRS0, LRS1 and LRS2 states to individual devices. To avoid any overfitting of the model, this data has only been made available *after* the modeling process was finalized.

The histogram drawn from these write operations on wafer b is depicted in Fig. 6a. Stuck devices have been pre-filtered to avoid intermixing different variation types. Subsequently, we evaluated whether the fitted distributions fit the resistances as they are observed for devices on wafer b. In order to achieve this, the cumulative density function (CDF) of each individual resistance distribution as simulated by the DT (as created for wafer



(a) Memory benchmark (GPU) **(b)** Real-time application (CPU) **(c)** Variation modules

Fig. 5. (**a**) Illustrates the achieved throughput when emulating differently sized memories on a single Nvidia H100 GPU. Apparently the performance scales linearly with the number of devices, indicating an efficient implementation. (**b**) Depicts the achieved FPS for a real-time application embedding different counts of crossbars on a CPU. For less than 180 HfO_2 RRAM crossbars the CPU performance is limited by the 30 FPS provided by the camera (dashed line). (**c**) Depicts the performance impact of evaluating different variation types simultaneously as indicated by the relative runtime (see text for more details).



(a) (b)

Fig. 6. Results: (**a**) depicts the resulting histogram after programming the $LRS_{0,1,2}$ states to 500 devices each on a separate wafer (wafer b). (**b**) depicts the CDF using the DT we created using data from wafer a and the empirical distributions we observed on wafer b.

a) has been compared to the observed actual empirical resistance distribution (see Fig. 6b). The observed distributions seem to match the simulations done using the DT well. A slight statistical shift could be explained by using different wafers.

Our efforts to validate the usage of parallel TaO_x devices are part of the supplementals.

Integration of the digital twin model into a NN framework performing fault aware training

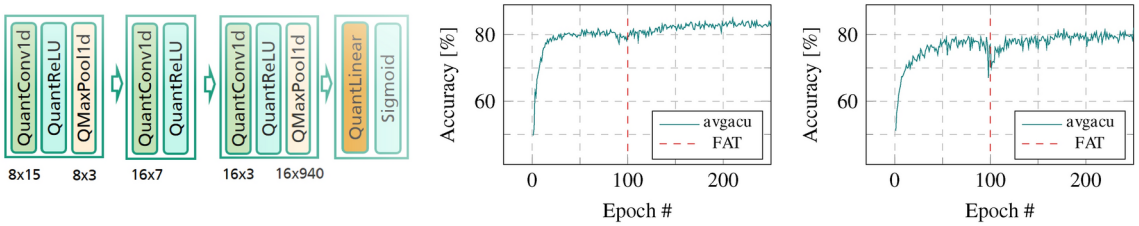
Given the notable speed of the proposed model, it seamlessly integrates into widely used NN frameworks. This integration facilitates the introduction of device faults to the network during the training phase, empowering the model to enhance its resilience to such faults. While the concept itself is not new, the rapid processing capabilities of our model allow its application across NNs of varying sizes. We exemplify this by employing the model to implement FAT in an NN designed for detecting cardiovascular diseases using ECG data sourced from the "China Physiological Signal Challenge 2018 (CPSC)" dataset[30]. The utilized network architecture is illustrated in Fig.7a, and the training process for two distinct RRAM device stacks is presented in Fig.7b and c. Our training setup introduces device faults starting from epoch 100, revealing that both networks swiftly recover, maintaining reasonable accuracy. More details regarding the dataset and the utilized network architecture are given in the first section of the Supplemental.

Discussion

Within this work we have demonstrated a modelling approach which helps mitigate two crucial aspects: Firstly, it outperforms existing modelling works by multiple orders of magnitude (as a result of being fully GPU compatible). Secondly, it resembles a push-button approach to modelling, being easy to fit to new devices or individual programming algorithms. While this model cannot predict the behaviour of hypothetical devices due to the fact that it is not a physics-based model it can emulate existing devices with high accuracy. Furthermore, we have illustrated its portability by integrating it into quite different application domains. Our memory emulation benchmark achieved multiple GB/s of effective simulation speed on a HPC GPU, which - to the best of the authors knowledge - outperforms any previous work. We illustrated that this model can be ported to a CPU implementation seamlessly, allowing for the evaluation of a real-time application. Lastly, our model enabled us to apply Fault Aware Training to a neural network, allowing the training procedure to adapt to two different types of RRAM devices. More implementation details are outlined in the next section.

Methodology

The implemented approach is summarized in Fig. 8. It consists of multiple steps, ranging from creating the Digital Twin (red box) to Validation (blue box). This section provides an introduction to these individual parts. Further details are given in the supplemental.



(a) NN architecture **(b)** FAT on the HfO_2 device **(c)** FAT on the TaO_x device

Fig. 7. (**a**) Illustrates the utilized network architecture, consisting of multiple convolutional layers and a single linear layer. (**b**) Depicts the first 250 epochs of the FAT training procedure for the HfO_2 device. (**c**) Depicts the same procedure done for the TaO_x device. Device variations were included starting from epoch 100 as indicated by the dashed line.

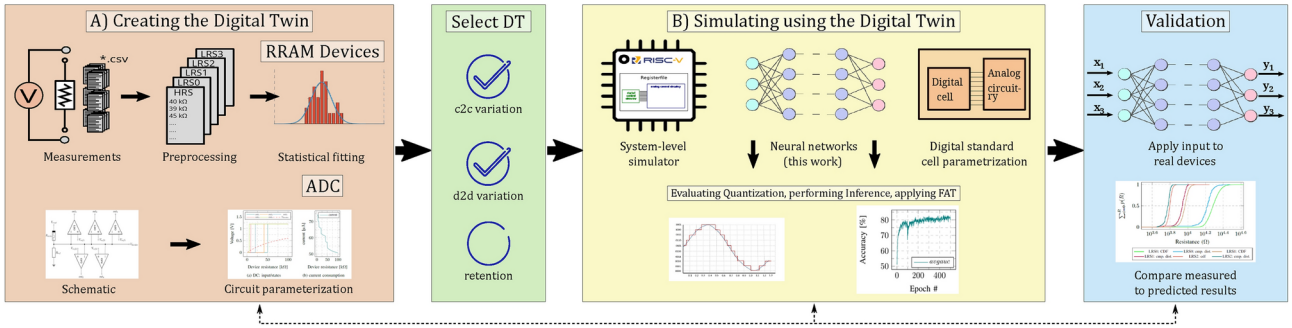


Fig. 8. Proposed framework: The evaluation begins with taking measurement data of RRAM devices (covering a given variation type) and combining them with circuit parametrizations to create a Digital Twin (red box). After selecting variation types (green box) these can be integrated into different applications to evaluate and/or mitigate the effects of device variations (yellow box). Lastly, the individual DT can be validated with additional measurement data (blue box).

RRAM device fabrication and measurements

Device fabrication

Although a materials exploration is beyond the scope of this work, we selected two different RRAM devices in order to evaluate the technological independence and flexibility of our framework. This section describes the fabrication of the two types of devices.

The HfO_2 RRAM device has been realized using a Metal-Insulator-Metal (MIM) stack consisting of a $150\,nm$ TiN top electrode, a $7\,nm$ Ti layer (under the TiN TE), and an $8\,nm$ HfO_2 switching layer grown by atomic layer deposition at IHP[31]. The MIM cell area is $0.4\mu\,m^2$. This device is fabricated within a 4 kbit 1T1R crossbar.

The TaO_x device also is an oxide-based memristor, but the stack is different and is composed of a $25\,nm$ Pt top electrode, a $15\,nm$ Ta layer, a $10\,nm$ TaO_x switching layer, and a $30\,nm$ Pt bottom electrode fabricated at FZJ[32]. We used a small number of individual MIM devices for characterizing this device.

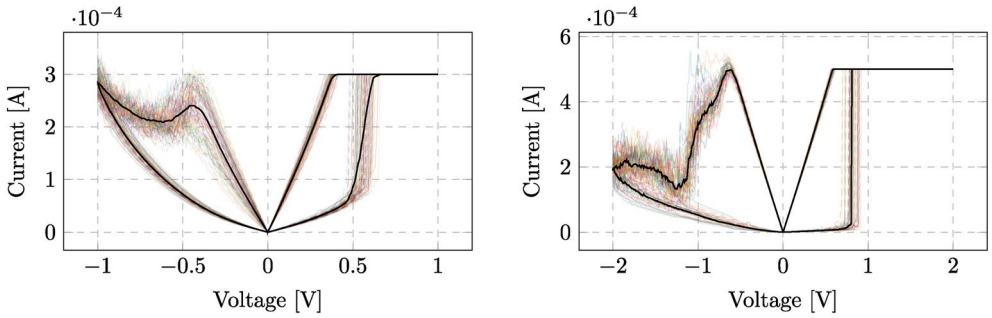
Figure 9 shows the I-V characteristics of both devices. One device each has been programmed to their respective conductance states for 100 (HfO_2) and 50(TaO_x) cycles. The I-V characteristics for the HfO_2 device are generated using a triangular-shaped DC voltage measurement. The input ranged from $\pm 1\,V$ with a step size of 0.1 V and current compliance set to $300\,\mu A$ during the SET operation. A triangular pulse ranging from $\pm 2\,V$ with a slew rate of 0.65 v/s and current compliance of $500\,\mu A$ during the SET operation has been used for thÿe TaO_x device.

Modelling—creating the digital twin

This section describes how to create an individual DT from raw measurement data (Fig. 8A). The measurements are implemented as described it in the previous section. This procedure is repeated for each individual metric (number of programming pulses, variation type, programming algorithm, ...) which is of relevance to the researcher. Since this is a fundamental aspect of this work the details have already been introduced in the results section and will not be repeated here.

Application integration—using the digital twin

Within this section we will explain how we rendered the proposed approach universal and ensured it meets performance requirements (Fig. 8B). The DT does not require application-specific adaption, the same device model can be used seamlessly across application domains.



(a) HfO_2 device **(b)** TaO_x device

Fig. 9. I-V characteristics of the utilized memristive devices. The black line depicts the median of individual measurements.

Memory simulator—GPU implementation

GPUs use the Single Instruction Multiple Threads (SIMT) approach to process many threads in parallel. A prominent example, namely ANN applications, have mostly moved to GPU processing since these provide significantly more compute capacity (about 67 TFLOP/s on a recent NVIDIA H100[33] GPGPU vs about 5 TFLOP/s on a recent AMD EPYC 9754 CPU with 128 Cores[34]). Unfortunately, as of now, this compute capacity is not yet available for most simulation models embedding RRAM, so either the computations need to be performed on the CPU core or the data needs to be moved between CPU and GPU constantly, yielding a substantial bottleneck.

Fortunately, RRAM devices are largely independent from each other, which allowed us to spawn threads without data dependencies without being bottlenecked by data transfers. Subsequently, we spawn one thread per RRAM device and spread them on the SMX units containing CUDA cores using the grid interface provided by the CUDA API. Locality effects can be covered using either the grid-thread mapping interface or by setting a cutoff distance (as typical for MD simulation frameworks) accordingly, preventing data movement bottlenecks. However, this needs to be easily accessible from an implementation-agnostic interface - the RRAM researcher should not have to deal with implementation details. Subsequently, we implemented this by initializing a PyTorch tensor per memory block accordingly and using CUDA kernels to act on them. This combination allows for eased integration into a given framework while still providing the full control of using CUDA directly. This GPU implementation can be ported to any language which is interoperable with CUDA without the need to rewrite the model itself.

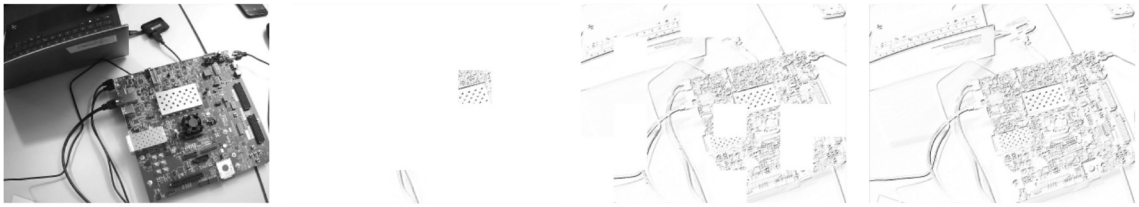
We used this implementation to perform a memory simulation using RRAM-blocks as storage simulating differently sized memories. This yields the performance as reported in Fig. 5a when run on a NVIDIA H100. Since a given number of threads is required to fully utilize the GPU the performance is lower for small matrices and saturates for larger ones. The performance scales linearly with problem size (using an increasing fraction of the GPU), indicating an efficient implementation not yielding major bottlenecks. The performance slightly decreases by a few % when fully utilizing the GPU, possibly since at this point threads need to be rescheduled within a single memory operation. The total number of simulated devices is limited by the GPU memory, up to about 1.25×10^9 devices can be simulated in this manner when using the 80 GB of HBM provided by this GPU. Multiple GPUs can be combined if this is insufficient for a given simulation.

Real-time application—CPU

While a CPU is fundamentally easier to program than a GPU, creating as many threads as on a GPU is not feasible. Nevertheless, a model needs to be compatible since a lot of devices such as laptops or embedded devices might not have a dedicated GPU. Fortunately, a lot of effort has been spent on rendering CPU implementations of iterator patterns within programming languages reasonably fast[35,36]. Subsequently, unlike for GPUs, where we would manually assign threads, we found that using iterator patterns such as numpy.enumerate in Python, the std::map operator in C++ or the "@" map operator in functional languages such as Mathematica yield performance advantages over manual thread scheduling.

We used this approach to evaluate the suitability of HfO_2 RRAM crossbars for a real-time application, namely edge detection filter kernels. Firstly, we gathered input images from a video camera at 30 FPS. Subsequently, we cut the input image into smaller sub-images and used simulated RRAM-crossbars to apply a Sobel edge-detection filter. The simulated results are displayed in real-time to allow for a reasonable investigation. We repeated this process with sub-images of different sizes, which allows to apply more parallelism to the edge-detection task since more RRAM-crossbars can be utilized in parallel. Using this approach we could evaluate whether we could stop a programming algorithm (consisting of 20 programming steps when done completely) early while still maintaining reasonable edge detection performance. Exemplified results for 48 crossbars and 5 and 15 steps respectively are depicted in Fig. 10. We set an entire crossbar to zero when an individual weight contains an error to ease interpreting the results. Apparently the RRAM crossbars are suitable for the task when applying 15 programming steps, but struggle to meet requirements when merely five programming steps have been applied to the individual crossbars.

The CPU needs to simulate an increasing number of crossbars while still meeting real-time requirements. Fortunately, as depicted in the performance graph in Fig. 5b the effective simulation speed of our model is fast enough for these evaluations.



(a) Input image **(b)** 5 pulses **(c)** 15 pulses **(d)** Ideal output

Fig. 10. Results: We use the DT to evaluate the suitability of HfO_2 RRAM-crossbars for an edge-detection task. (**a**) Depicts the input image, consisting of an FPGA board and the laptop running the simulation. (**d**) Depicts the desired edge detection result. (**b**) and (**c**) depict the results when 5 and 15 pulses have been applied to the individual crossbars.

Implementing fault aware training for an artificial neural network running on ECG data
The concept of FAT has been proposed by Zahid et al.[6] as a means to mitigate faults caused by radiation faults within an FPGA performing ANN inference. The approach involves presenting device faults/inaccuracies to the model during the training phase to enable the model to train itself to be resilient to device faults. We implemented this approach for the training of ANNs storing their weights using (possibly faulty) RRAM devices. We use the Xilinx Brevitas framework (which builds on PyTorch) as basis for our RRAM-ANN implementation. PyTorch provides a powerful ANN environment which has already proven usefull for custom extensions[37]. Brevitas extends Pytorch with fine-grained quantization support. We have extended this by adding a custom quantizer, a custom device variation module and our own gradient handling.

PyTorch provides the "tensor" data structure, yielding objects that can be stored and evaluated fast on a GPU. However, this proved not flexible enough to cover all aspects of RRAM models. Subsequently, we have split up our implementation into parts which can be implemented using Tensors and parts which need to be implemented with CUDA and combined them into one coherent implementation. The performance characteristics are similar to the memory simulation described earlier.

Quantized ANNs with implemented RRAM device variabilities Since not all networks can be quantized appropriately, the proposed flow allows for evaluating the quantizability of a given network. We have written a custom quantizer block for Brevitas to implement different quantization techniques. Additional modules, such as noise or device variations, can be added before and after this module. While the approach to gradient calculation implemented within Brevitas (straight-through estimator[38]), works well for traditional architectures, maintaining a continuous gradient while being true to the device variations is a bit more difficult since errors such as stuck-at errors are - by definition - non-continuous. For others, such as D2D variation, an analytical description of the error might help to maintain the gradient. Keeping with the separation offered by the straight-through estimator does not seem reasonable since the training algorithm would keep trying to change weights which ultimately can never be changed (since the device is stuck).

This approach tries to find a middle ground: For the forward pass, the device faults are applied in a way that is similar to the quantization. To do so, a wrapper for the Digital Twin has been integrated into a custom pre/postquantizer block utilized within the custom Brevitas implementation. A matrix correlating to the stuck devices is constructed for the backward pass. Firstly, each element is initialized with ones (Eq. 3).

$$\mathbf{G}_{\mathbf{modstuck}} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{3}$$

Subsequently, each weight gradient update that is to be stored within a stuck RRAM device is multiplied by $0 < x < 1$; The precise number is an optimization problem between maintaining continuity and staying true to the devices. Empirically $0.5 < x < 0.7$ has yielded good results. This yields a matrix as shown in Eq. (4) which is reused during the training procedure.

$$\mathbf{G}_{\mathbf{modstuck}} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} = \begin{pmatrix} 0.7 & 1 & 1 \\ 1 & 0.7 & 0.7 \end{pmatrix} \tag{4}$$

During each backpropagation step, each gradient is element-wise multiplied with this matrix, in an attempt to encourage the training procedure to not depend too much on individual weights. This can be further enforced by repeatedly reinitializing the devices during the training procedure, causing different devices to be stuck. Other types of device errors are integrated similarly, by introducing further modifications to the gradient in a cautious manner to ensure trainability of the ANN.

Performance of fault aware training on an ANN using ECG data This implementation was used to train a network on the CPSC dataset[30], detecting eight individual cardiovascular diseases and a healthy patient (denoted as SNR) using ECG data. We started from a floating point implementation, added quantization aware training and lastly performed fault aware training for both the HfO_2 and TaO_x devices. The training procedure when using floating point numbers is depicted in Fig. 11a, which indicates that the proposed network architecture is mostly suitable for this task. Unfortunately the network fails to detect Premature Atrial Contraction (PAC) reliably. This result serves as a baseline - apparently using this network architecture and merely using three input leads (see supplementals for more information) prevents detecting this morbidity. Subsequently, the network performing poorly regarding PAC when applying quantization and device variations would not be a result of the former, but a limitation of the utilized architecture.

As described in a previous section coupling two RRAM devices to form a single weight was neccesary when training for the TaO_x device since these devices could not store more than two states reliably by themselves. A single RRAM device was used to store the weights when using the HfO_2 devices.

Subsequently, we applied both quantization aware training and FAT using the proposed DT approach. We configured the training procedure in such a way that device variation effects would start occuring from epoch 100. The resulting training procedure for the first 150 epochs when using the TaO_x device is depicted in Fig. 11b. The accuracy drops shortly after introducing device variations but recovers within a few epochs. Averaging the individual accuracies leads to the plots as depicted in Fig. 11c and e. Apparently both networks recover when presented with device variations, yielding acceptable application performance which is comparable to the floating point implementation. More details regarding dataset and network architecture are provided as part of the supplemental.

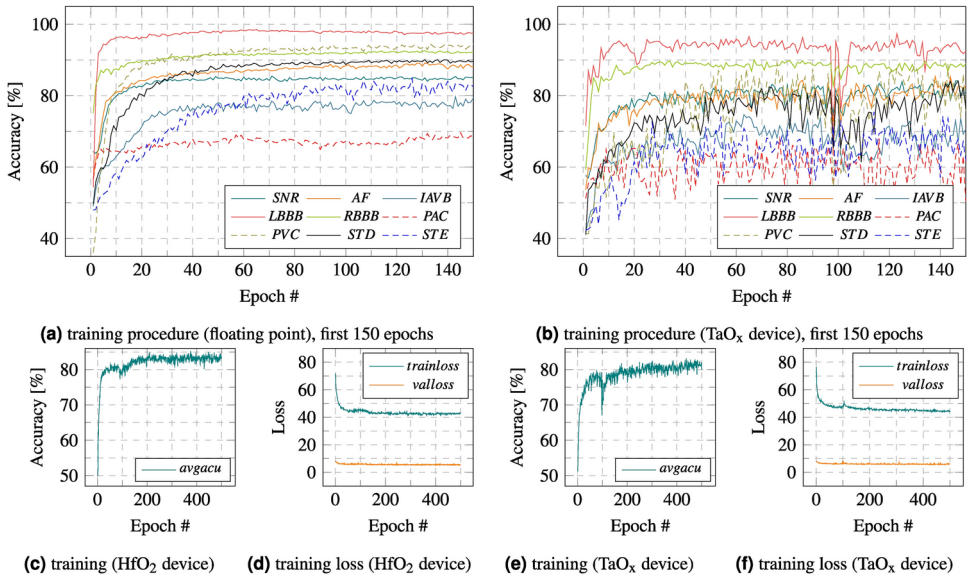


Fig. 11. This figure depicts the individual training procedures when training neural networks to detect the morbidities as prevalent in the CPSC dataset. (**a**) Depicts the first 150 epochs of the training procedure when using floating point numbers, serving as a baseline. Apparently all morbidities but PAC are detected reasonably well. The remaining Figures depict the results when applying both quantization and FAT. (**b**) Depicts the first 150 epochs of the same procedure when applying FAT to the TaO_x device. The network looses performance when the FAT procedure begins (at epoch 100), but recovers succesfully. A more corse-grained representation using the corresponding averaged accuracies are depicted in (**c**) (HfO_2 device) and (**e**) (TaO_x device). The network using HfO_2 devices also trains sucesfully. The drop in accuracy when enabling FAT appears to be a little smaller compared to the TaO_x equivalent. The corresponding training/validation loss is depicted in (**d**) and (**f**), indicating that the training procedure did not overfit on the training data.

Data availability

The data that support the findings of this study are available from the corresponding author on reasonable request. The CPSC dataset containing the utilized ECG data has been published in an open access database by its creators[30].

Received: 9 May 2024; Accepted: 16 September 2024
Published online: 10 October 2024

References

1. Banerjee, W. et al. Design of CMOS compatible, high-speed, highly-stable complementary switching with multilevel operation in 3D vertically stacked novel $HfO_2/Al_2O_3/TiO_x$ (HAT) RRAM. *Adv. Electron. Mater.* **4**, 1700561 (2018).
2. Fritscher, M. et al. Simulating large neural networks embedding MLC RRAM as weight storage considering device variations. In *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*, 1–4 (IEEE, 2021).
3. Fritscher, M. et al. Prototyping reconfigurable RRAM-based AI accelerators using the RISC-V ecosystem and digital twins. In *International Conference on High Performance Computing*, 500–514 (Springer, 2023).
4. Le, B. Q. et al. Radar: A fast and energy-efficient programming technique for multiple bits-per-cell RRAM arrays. *IEEE Trans. Electron Devices* **68**, 4397–4403 (2021).
5. Pérez, G. A., Zambelli, C., Olivo, P. & Wenger, C. Impact of the incremental programming algorithm on the filament conduction in HfO_2-based RRAM arrays. *IEEE J. Electron Devices Soc.* **5**, 64–68 (2016).
6. Zahid, U., Gambardella, G., Fraser, N. J., Blott, M. & Vissers, K. Fat: Training neural networks for reliable inference under hardware faults. In *2020 IEEE Int. Test Conf. (ITC)*, 1–10 (IEEE, 2020).
7. Peng, X., Huang, S., Jiang, H., Lu, A. & Yu, S. DNN+ NeuroSim V2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **40**, 2306–2319 (2020).
8. Ielmini, D. & Milo, V. Physics-based modeling approaches of resistive switching devices for memory and in-memory computing applications. *J. Comp. Electron.* **16**, 1121–1143 (2017).
9. Tappertzhofen, S. et al. Modeling of quantized conductance effects in electrochemical metallization cells. *IEEE Trans. Nanotechnol.* **14**, 505–512 (2015).
10. Jiang, Z. et al. A compact model for metal-oxide resistive random access memory with experiment verification. *IEEE Trans. Electron. Devices* **63**, 1884–1892 (2016).
11. Panda, D., Sahu, P. P. & Tseng, T. Y. A collective study on modeling and simulation of resistive random access memory. *Nanoscale Res. Lett.* **13**, 1–48 (2018).
12. Dong, X., Xu, C., Xie, Y. & Jouppi, N. P. Nvsim: a circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **31**, 994–1007 (2012).
13. Muralimanohar, N., Balasubramonian, R. & Jouppi, N. P. Cacti 6.0: A tool to model large caches. *HP Lab.* **27**, 28 (2009).
14. Peng, X., Huang, S., Luo, Y., Sun, X. & Yu, S. DNN + Neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *2019 IEEE International Electron Devices Meeting (IEDM)*, 32–5 (IEEE, 2019).

15. Chen, P.-Y., Peng, X. & Yu, S. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **37**, 3067–3080 (2018).
16. Paszke, A. et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* Vol. 32, 8024–8035 (Curran Associates, Inc., 2019).
17. Zhu, Z. et al. Mnsim 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 83–88 (2020).
18. Lin, M.-Y. et al. Dl-rsim: A simulation framework to enable reliable reram-based accelerators for deep learning. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1–8 (2018).
19. Lammie, C., Xiang, W., Linares-Barranco, B. & Azghadi, M. R. Memtorch: An open-source simulation framework for memristive deep learning systems. *Neurocomputing* **485**, 124–133 (2022).
20. Kvatinsky, S., Ramadan, M., Friedman, E. G. & Kolodny, A. Vteam: A general model for voltage-controlled memristors. *IEEE Trans. Circuits Syst. II Express Briefs* **62**, 786–790 (2015).
21. Messaris, I. et al. A data-driven verilog-a reram model. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **37**, 3151–3162 (2018).
22. Waterman, A. et al. The risc-v instruction set manual. Volume I: User-Level ISA, version **2**, 1–79 (2014).
23. Gaba, S., Knag, P., Zhang, Z. & Lu, W. Memristive devices for stochastic computing. In *2014 IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 2592–2595 (IEEE, 2014).
24. Pérez, E. et al. Analysis of the statistics of device-to-device and cycle-to-cycle variability in tin/Ti/Al: HfO_2/tin RRAMS. *Microelectron. Eng.* **214**, 104–109 (2019).
25. Milo, V. et al. Multilevel hfo2-based rram devices for low-power neuromorphic networks. *APL Mater.* **7** (2019).
26. Bende, A. et al. Experimental validation of memristor-aided logic using 1t1r tao x rram crossbar array. In *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*, 565–570 (IEEE, 2024).
27. Wu, L. et al. Study on high-resistance state instability of TaOx-based RRAM. In *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 1–3 (2018).
28. Li, X., Wu, H., Gao, B., Deng, N. & Qian, H. Short time high-resistance state instability of TaO_x-based RRAM devices. *IEEE Electron Device Lett.* **38**, 32–35 (2017).
29. Sobel, I. An isotropic 3x3 image gradient operator. Presentation at Stanford A.I. Project 1968 (2014).
30. Liu, F. et al. An open access database for evaluating the algorithms of electrocardiogram rhythm and morphology abnormality detection. *J. Med. Imaging Health Inform.* **8**, 1368–1373 (2018).
31. Perez, E., Mahadevaiah, M. K., Quesada, E.P.-B. & Wenger, C. Variability and energy consumption tradeoffs in multilevel programming of RRAM arrays. *IEEE Trans. Electron Devices* **68**, 2693–2698 (2021).
32. Kempen, T., Waser, R. & Rana, V. 50x endurance improvement in TaOx RRAM by extrinsic doping. In *IEEE Int. Memory Workshop (IMW)*, 1–4 (2021).
33. NVIDIA. H100 datasheet. https://www.nvidia.com/en-us/data-center/h100/ (2023).
34. AMD. High performance computing (hpc) tuning guide for amd epyc 9004 series processors. https://www.amd.com/ (2024).
35. Graefe, G. Iterators, schedulers, and distributed-memory parallelism. *Softw. Pract. Exp.* **26**, 427–452 (1996).
36. Gibbons, J. & Oliveira, B. C. D. S. The essence of the iterator pattern. *J. Funct. Program.* **19**, 377–402 (2009).
37. Gao, X., Ramezanghorbani, F., Isayev, O., Smith, J. S. & Roitberg, A. E. TorchANI: a free and open source PyTorch-based deep learning implementation of the ANI neural network potentials. *J. Chem. Inf. Model.* **60**, 3408–3415 (2020).
38. Yin, P. et al. Understanding straight-through estimator in training activation quantized neural nets. arXiv preprint arXiv:1903.05662 (2019).

Acknowledgements

The work was supported in parts by the Deutsche Forschungsgemeinschaft (DFG) within the Projects MIMEC (Project No. 441921944) and Mem2CNN, which are part of the SPP MemrisTec (Project No. 422738993) and by the BMBF by the Federal Ministry of Education and Research (BMBF, Germany) in the Projects iCampus II (Project No. 16ES1128K), 6G-RIC (Project No. 16KISK026), LODRIC (Project No. 16ME0386) and NEURO-TEC (Project Nos. 16ME0398K and 16ME0399). The authors gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). The hardware is funded by the German Research Foundation (DFG).

Author contributions

M.F. conceived the concept and designed the experiments with support from T.R., D.R., D.F, M.R. and C.W. M.F. and T.R. implemented the data preprocessing steps. A.Ba. performed measurements on HfO_2 devices with support from C.W.; T.K., S.S., A.Be., S.M. and V.R. fabricated and performed measurements on TaO_x devices. M.M. and D.H. prepared the ECG dataset and implemented an ANN working on the input in FP representation. M.F. and M.M. implemented improved quantization support. M.U. and G.K. assisted with implementing parallel devices. M.F. implemented the automated modelling procedure and both CPU and GPU implementations. M.F and S.S. wrote the paper with help from F.M., S.M., M.K and C.W. All authors reviewed and discussed the manuscript.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-024-73439-z.

Correspondence and requests for materials should be addressed to M.F.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.