nature communications

Article

https://doi.org/10.1038/s41467-025-56319-6

Layer ensemble averaging for fault tolerance in memristive neural networks

Received: 23 April 2024

Accepted: 15 January 2025

Published online: 01 February 2025

Check for updates

Osama Yousuf[1,2,3], Brian D. Hoskins[2], Karthick Ramu[2], Mitchell Fream[2], William A. Borders[2], Advait Madhavan[2], Matthew W. Daniels[2], Andrew Dienstfrey[4], Jabez J. McClelland[2], Martin Lueker-Boden[3] & Gina C. Adam[1]

Artificial neural networks have advanced due to scaling dimensions, but conventional computing struggles with inefficiencies due to memory bottlenecks. In-memory computing architectures using memristor devices offer promise but face challenges due to hardware non-idealities. This work proposes layer ensemble averaging—a hardware-oriented fault tolerance scheme for improving inference performance of non-ideal memristive neural networks programmed with pre-trained solutions. Simulations on an image classification task and hardware experiments on a continual learning problem with a custom 20,000-device prototyping platform show significant performance gains, outperforming prior methods at similar redundancy levels and overheads. For the image classification task with 20% stuck-at faults, accuracy improves from 40% to 89.6% (within 5% of baseline), and for the continual learning problem, accuracy improves from 55% to 71% (within 1% of baseline). The proposed scheme is broadly applicable to accelerators based on a variety of different non-volatile device technologies.

The increasing demand for large-scale neural network models has prompted extensive research on approaches to optimize model efficiency and accelerate computations. Quantized neural networks, which utilize reduced-precision representations for model parameters and activations, have emerged as a promising avenue for achieving significant computational gains without compromising performance. Recent works advance this approach to the extreme demonstrating effective 1-bit[1] and 1.58-bit[2] (ternary) quantization of the parameter space with minimal performance loss. In addition to precision reduction, memory-based hardware accelerators are emerging as another frontier to enhance neural network efficiency. In particular, in-memory computation on a physical chip of memory devices called memristors offers a synergistic solution that can complement efficiency gains achieved by network quantization[3].

Memristors are two-terminal non-volatile memory devices that exhibit unique programmable resistive switching behavior. Their intrinsic characteristics enable co-location of computation and memory, mimicking aspects of synaptic functionality in biological systems[4,5]. By arranging memristors over a two-dimensional array such that the devices are placed at intersection points, an architecture commonly referred to as a crossbar[6–8], underlying device physics can be exploited to implement parallelized vector-matrix multiplication—a critical operation in artificial neural networks—in the analog domain. Since memristor crossbars are programmable and non-volatile[9,10], they can be utilized to build dedicated hardware accelerators for deep neural networks. Diverse technologies including resistive random-access memory (ReRAM) and phase change memory are being considered as promising crossbar candidates to implement the multiply and accumulate operations representing the standard synaptic weights model used in most neural networks[11–19]. Hardware accelerators based on these technologies can overcome von Neumann architecture limitations as they minimize data movement and energy

[1]Department of Electrical and Computer Engineering, George Washington University, Washington, DC, USA. [2]National Institute of Standards and Technology, Gaithersburg, MD, USA. [3]Western Digital Technologies, San Jose, CA, USA. [4]National Institute of Standards and Technology, Boulder, CO, USA.
e-mail: ginaadam@gwu.edu

consumption, both of which are key bottlenecks for large-scale neural network workloads. For these reasons, memristive neural network accelerators have the potential to transform capabilities of machine learning systems and thereby usher in a neuromorphic era of artificial intelligence computing at the edge. A comprehensive exploration of the interplay between quantized neural networks, dedicated hardware accelerators, and memristive technologies becomes imperative for advancing this capability with the goal to unlock unprecedented efficiency gains in real-world deep learning applications.

The study of memristive neural network accelerators faces several challenges. A purely experimental approach is unfeasible since commercial tape-outs have long timelines and significant design and fabrication costs[4,20,21]. Before hardware prototyping can be practically motivated, results from hardware-aware[22–24] simulations—simulations that take hardware characteristics into account—are needed to reliably predict the performance of these systems. The critical issue is that these devices can exhibit complex non-idealities such as cycle-to-cycle variability (inconsistency in performance from one switching cycle to the next), device-to-device variability (variation in behavior between individual devices), and even tuning failure (where the resistance state remains stuck at a certain level). Operating arrays of these devices introduces system-level non-idealities as well, such as noise and precision/resolution limitations arising from analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and trans-impedance amplifiers. These non-idealities manifest as deviations in the outputs of the underlying vector-matrix multiplications, rendering memristive neural networks incapable of achieving software-equivalent accuracies without some mechanism for fault tolerance in place.

There have been numerous advancements in fault tolerance schemes for bridging this performance gap in inference accuracy of non-ideal memristive networks compared to their software counterparts[6,25–30], with more recent works focusing on challenging continual learning settings[31,32]. Existing literature on the subject can be classified into two broad categories. The first focuses on circuit-level and device-level optimizations[6,25] such as alterations to the crossbar configuration and circuitry or the device material stack, and the second on network-level algorithmic optimizations[26–30,33] such as advanced programming or weight-to-device mapping and encoding schemes. Algorithmic investigations can be further divided into two sub-categories. The first utilizes information about hardware defects and attempts to train defect or hardware-aware solutions for the memristive hardware, while the second attempts to transfer a pure-software solution on the memristive hardware intelligently. Both approaches share the goal of maximizing the performance of the memristive network. The first achieves this by training redundancy into the solution, while the second achieves this by averaging out induced currents from multiple mapped instances of one or more solutions on the crossbar.

We offer a detailed comparison of prominent works from literature that propose fault tolerance solutions for memristive neural networks in Supplementary Table 1. Most works are simulation-oriented and are not validated with system-level neural network results from a hardware prototype. They make assumptions on non-idealities that do not fully represent a practical memristor crossbar system. For example, a common assumption about stuck-at faults is that stuck-low and stuck-high devices take on conductances G_{OFF} and G_{ON} (commonly formulated in terms of resistances as high resistance state and low resistance state respectively), while all operable devices can be tuned within this range to a relatively high precision (typically ≥ 6 bits). Additionally, some studies overlook the fact that defect maps can evolve over time due to repeated switching cycles, or due to the application of damaging voltages which can vary from one device to another.

With the prototyping system and memristive devices used in this work (presented later in the *Methods: Experimental Setup* section), we found that devices with stuck-at faults have conductance values that are significantly beyond the conductance range of non-stuck, operable devices. Although this is a common occurrence in array characterization studies[34,35], few existing works on fault tolerance schemes account for this, as indicated by the stuck-at state formulation column in Supplementary Table 1. We also found that tuning an operable device to the conductance of a stuck device tends to permanently damage the device due to irreversible dielectric breakdown, limiting device endurance[9,36]. This leads to an outdated defect map and, if unaddressed, a considerable reduction in the overall usability of the chip for any computational workload. This means that one must exercise caution when tuning operable devices, ensuring that the limited dynamic range of $[G_{OFF}, G_{ON}]$ is strictly respected across the array. Under these limitations, traditional fault tolerance schemes based on redundant rows or columns of memristors lose their effectiveness as a single stuck device in the positive conductance matrix can no longer be compensated by a single operable device in the negative conductance matrix (or vice versa).

This problem is further exacerbated by system-imposed limitations on device tunability. For example, even though we can tune individual devices to multi-bit precision (presented later in the *Results: Device and Array Characterization* section), operating contiguous blocks in parallel and realizing parallel vector-matrix multiplication for inference imposes new constraints on conductance states due to limited precision and noise from system components such as ADCs, DACs, transimpedance amplifiers, and other transient sources. To compensate, we artificially limit target conductance states for our devices to a lower precision of 1-bit. From a device perspective, this regime is relevant beyond memristors, as certain device technologies—such as magnetic tunnel junctions[37]—only support two conductance states. Just as a unified prototyping system is essential for the development of emerging memory devices[38], a device-agnostic fault tolerance algorithm that can fairly compare the neural network performance of different technologies is equally critical.

These practical considerations form the foundation of our work on layer ensemble averaging. We propose layer ensemble averaging (summarized in Fig. 1, see *Methods: Layer Ensemble Averaging* for more details), a hardware-correction fault tolerance scheme designed for improving the performance of memristive neural networks irrespective of the underlying device technology. It leverages the concept of redundant devices and eliminates the need for high device and system tunability, making it a versatile, assumption-free, and universally applicable solution for fault tolerance in these networks. It also does not require re-training and is thus particularly suitable in the context of a fixed prototyping system designed for evaluating device technologies or parameters against one another. It comprises of block-by-block weight mapping and encoding algorithms that operate together to directly correct vector-matrix multiplication outputs in hardware at the level of individual neural network layers. Contrary to existing related literature[26,30] where neural network outputs are obtained by polling outputs of an ensemble of neural networks, not necessarily mapping the same solution, here ensemble outputs are polled at the level of each layer by mapping the same solution multiple times. This key difference makes our approach suitable for cases where the availability of software solutions is limited (perhaps due to high training costs or long training times) and one wishes to deploy a pre-trained model directly to memristive hardware for acceleration with minimal performance loss. It also leads to a straightforward way to investigate hardware outputs relative to software outputs of the vector-matrix multiplication operation for any given layer. Consequently, the proposed approach has the added benefit of being useful for domains beyond deep learning that require accurate multiply-and-accumulate or vector-matrix multiplication operations such as digital signal processing, image and video processing, scientific computing, and financial modeling.

https://doi.org/10.1038/s41467-025-56319-6

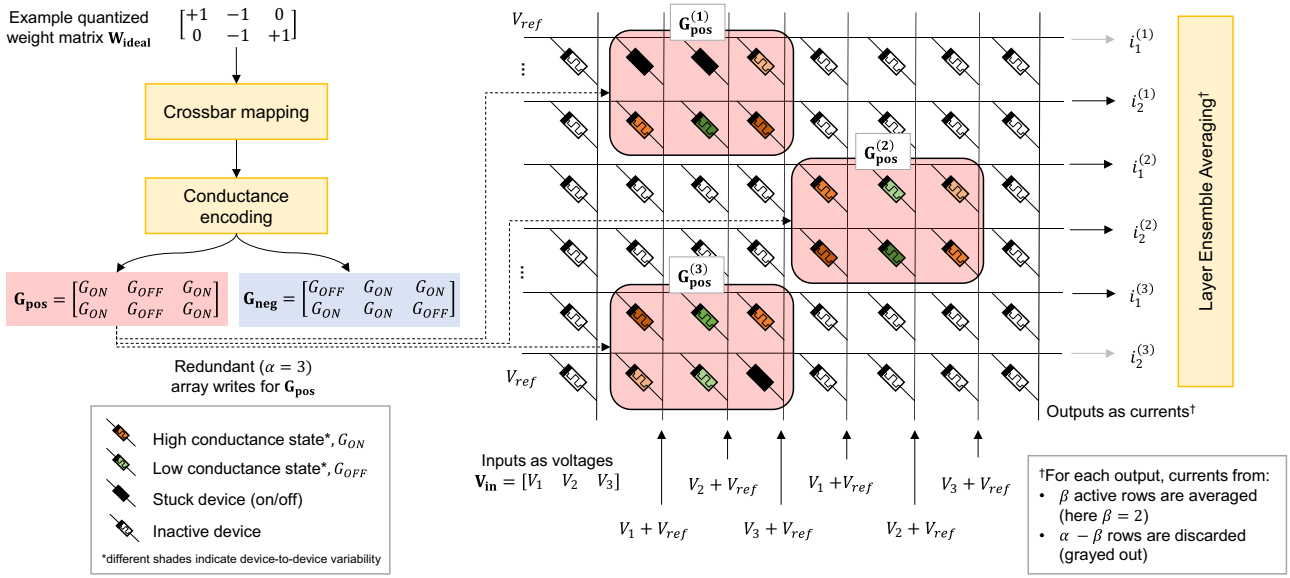

Fig. 1 | Layer ensemble averaging. Demonstration of the proposed layer ensemble averaging technique for mapping an example weight matrix $\mathbf{W_{ideal}}$ to a crossbar of non-ideal memristive devices. The first step is to determine appropriate locations on the crossbar for mapping (see Supplementary Algorithm 1), followed by an encoding step (see Table 1 and Supplementary Algorithm 2) that converts the weight matrix to device conductance matrices $\mathbf{G_{pos}}$ and $\mathbf{G_{neg}}$. Each encoded conductance matrix is written to contiguous blocks of devices on the non-ideal chip according to respective mappings that contain additional information about row-wise defects based on the summed conductance variation metric from Eq. (1). This is utilized during inference to suppress currents from highly defective rows from participating in the averaging process (see Eq. (3)). Green and orange coloring represents devices in low and high conductance states respectively, with varying shades indicating device-to-device variability, while black and white coloring represents devices that are stuck/faulty or inactive respectively. For simplicity, only the layer ensemble mapping for $\mathbf{G_{pos}}$ is shown. Redundancy parameters shown: $\alpha = 3$, indicating that the weight matrix (represented by conductance matrices $\mathbf{G_{pos}}$) is mapped on to the crossbar three times with $\mathbf{G_{pos}^{(i)}}$ representing the i-th mapping of $\mathbf{G_{pos}}$ (similarly for $\mathbf{G_{neg}}$), and $\beta = 2$, indicating that from this layer ensemble of size 3, currents from 2 active rows (based on the SCV metric from Eq. (1)) will be averaged for each output.

In this work, we demonstrate that pre-trained and quantized neural network solutions written to a non-ideal memristive crossbar can attain near-software inference performance using the proposed layer ensemble averaging fault tolerance scheme. We present simulations informed by experiments where we compare the performance of layer ensemble averaging with hardware correction methods from literature (bold entries in Supplementary Table 1) for classifying the MNIST handwritten digit dataset[39]. Realistic device and system non-idealities are modeled in these simulations. We offer comparisons with two existing state-of-the-art hardware-oriented fault tolerance schemes *Mapping Algorithm with inner fault tolerant ability (MAO)*[27,33] and the *Committee Machines (CM)* algorithm[26]. We do not offer comparisons with software-oriented schemes such as those based on error correcting codes and architectures[40–43], because they are orthogonal solutions that can be paired with any device redundancy-based fault tolerance scheme. Moreover, we do not offer comparisons with schemes involving re-training, because one of our goals is to have a fault tolerance scheme that is independent of the underlying technology, and solutions involving re-training inherently utilize device-specific properties such as tunability. Additionally, we provide an end-to-end system demonstration where we validate the overall effectiveness of layer ensemble averaging as a fault tolerance approach on a continual learning problem based on the Yin-Yang dataset[44] using an array of ReRAM devices with our custom-built mixed-signal hardware prototyping platform called Daffodil[38] (summarized in Fig. 2, see *Methods: Experimental Setup* for more details). This platform consists of an integrated chip with 20,000 memristive devices, a mixed-signal printed circuit board (PCB), a Zynq-based (certain commercial processes and software are identified in this article to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the processes and software identified are necessarily the best available for the purpose) field programmable gate array (FPGA) development board, and an accompanying software suite guided by principles of hardware-software co-design for hardware experiments as well as accurate system simulations.

Results

Device and array characterization

We summarize the ReRAM device and crossbar array characterization results gathered using our mixed-signal prototyping platform in Fig. 3. Figure 3a shows a micrograph of the 20,000-device chip. The first step for characterization involves observing current vs. voltage (*I-V*) characteristics of these devices. The objective is to identify a set of conductance states to which a maximal percentage of devices on the chip can be written. For our devices, the optimal set was found to comprise of four conductance states, [133, 167, 200, 233] µS, indicating that our devices can be tuned to 2-bit precision. For neural network experiments, the two extreme states from this optimal conductance set, 133 µS and 233 µS, were utilized as G_{OFF} and G_{ON} to maximize the separation for the ternary neural network weights and minimize state overlap, which is a key requirement for implementing a neural network based on these devices.

Figure 3b shows retention properties with the four identified conductance states for a single representative device from the chip. The process involves initial tuning or programming of a device to a requested conductance state followed by successive measurements over a specified timeframe. An iterative margin-based programming scheme was utilized where a device is considered successfully programmed if its conductance is read back as $G_{req} \pm \theta$, where G_{req} is the target or requested conductance and θ is a user-specified margin. A smaller θ can lead to more precise tuning at the expense of increased difficulty to program as the resolution limitations and noise present in the system (primarily arising from the ADCs and DACs) restrict very precise measurements. A smaller θ requires more iterations for programming and thus bears a higher chance of wearing out devices. For

https://doi.org/10.1038/s41467-025-56319-6

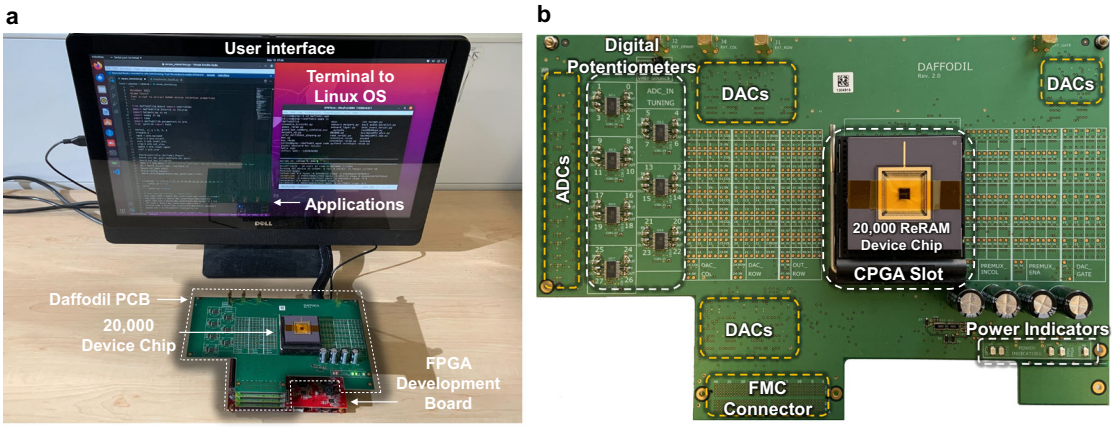

Fig. 2 | Overview of the experimental setup. a Image of the complete working setup. The FPGA development board interfaces with the Daffodil printed circuit board housing the 20,000 device ReRAM chip (outlined in white) via a FPGA mezzanine card (FMC) connector. The development board also connects to a host system over a serial terminal through which high-level applications can be run. **b** A zoomed-in view of the outlined mixed-signal Daffodil board showing major hardware components present on-board. ADCs, DACs, and the FMC connector are located under the board.

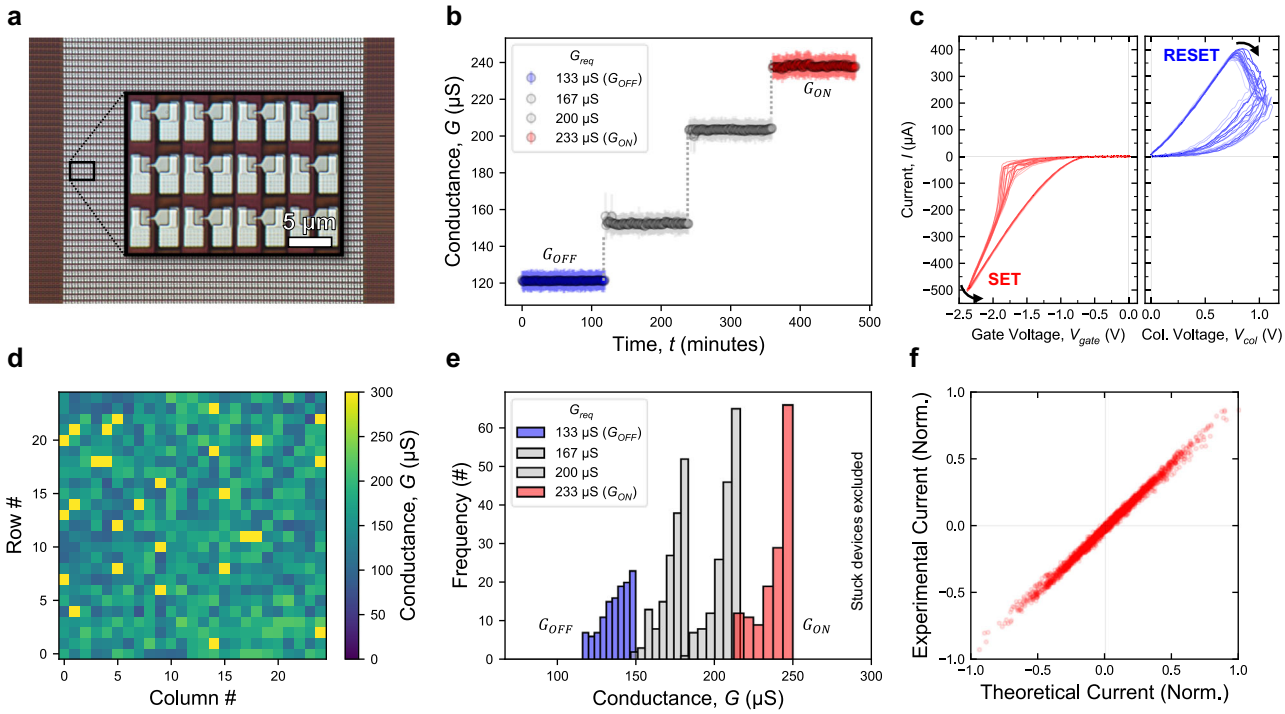

Fig. 3 | ReRAM device and array characterization. a Zoomed-in device micrograph of a small subarray within the 20,000-device chip. **b** Device retention data showcasing effective 2-bit tuning of a representative device. The error bars indicate three standard deviations. States G_{ON} and G_{OFF} are utilized for the neural network implementation. **c** Current *vs.* voltage (*I-V*) sweep curves over 20 cycles of a single representative device from the chip showing low cycle-to-cycle variability. **d** Conductance map and **e** corresponding conductance distributions of devices in a kernel randomly programmed to one of four conductance states from (**b**). The color map is clamped at 300 µS for clarity; actual stuck-at high conductances are significantly higher, averaging 500 µS. **f** Comparison of theoretical accumulated currents and experimental accumulated currents on the kernel for 100 randomly generated input voltage vectors repeated for a total of 20 iterations showing effective multiply-and-accumulate and vector-matrix multiplication operations. All experimental results are gathered using the Daffodil mixed-signal prototyping platform.

this reason, a relaxed margin of $\theta = 16.66$ µS was used. The retention data was gathered by tuning the device to a specified state via this scheme, and then reading back its conductance once per second over a period of 2 h. As can be seen, the devices exhibit relatively stable states, which is a crucial requirement for any neural network. We note here that we found numerous devices that could be tuned to a much larger set of conductance states using our margin-based programming scheme. An example of one such device tuned to a total of 10 unique conductance states is presented in Supplementary Fig. 1. Figure 3c shows *I-V* sweep curves for a single representative device. The data was gathered over a total of 20 cycles to capture the underlying cycle-to-cycle variability. The SET and RESET phases of the device are highlighted. For the SET phase, we fixed the column voltage at 2.0 V and swept the gate bias. This proved more reliable with our source-drain configuration for the SET operation, with the source bias floating between the memristor and ground. For RESET, we used a traditional

column-row voltage sweep with a gate voltage of 3.3 V. The obtained curves show that the devices exhibit acceptable cycle-to-cycle variability. The dynamic range and precision of the current measurements were tuned directly within high-level application code to maximize the effective signal-to-noise ratio.

The specified conductance states were utilized to realize a general vector-matrix multiplication operation on the crossbar where the matrix entries take on one of four values. A single kernel represents the highest degree of physics-based parallelism attainable for our system. For a given kernel, the algorithm for our vector-matrix multiplication experiment proceeds as follows: first, each device is randomly assigned to one of the four conductance states and is tuned to that using our margin-based tuning algorithm. Secondly, a kernel-level map of device conductances is extracted. This information is utilized to directly calculate the theoretical accumulated current for a given vector of input voltages. Finally, a sequence of input voltage vectors (with 25 dimensions corresponding to inputs for the kernel) is generated, with each voltage in each vector selected randomly from the interval $[-0.3, 0.3]$ V (corresponding to $[-V_{read}, V_{read}]$). This random vector of voltages is then applied onto the kernel columns as inputs and the accumulated experimental current outputs are measured on the rows in parallel. These currents are compared with corresponding theoretical outputs. Figure 3d shows the conductance map extracted for a representative kernel after the write step of the vector-matrix multiplication operation, and Fig. 3e shows corresponding device conductances as a histogram. After programming, the devices have two kinds of non-idealities. The first non-ideality arises from the programming scheme. Since the programming scheme is based on a margin, two devices tuned successfully to the same state still have variability. As a result, there is a small degree of overlap between device conductances for neighboring states. The second non-ideality arises from unavoidable physical characteristics of the chip where devices can either be stuck in a high or low conductance state altogether or be electrically shorted. The tuning algorithm fails for these devices. For the kernel reported in Fig. 3d, the tuning succeeds for ~ 96 % of devices and a total of 27 devices with stuck-at faults were found. Across different kernels, we found that the average conductance of devices with such stuck-at high faults was 500 μS, while the average conductance of devices with stuck-at low faults was 10 μS, both of which are significantly beyond the operating range of operable devices (G_{OFF} = 133 μS and G_{ON} = 233 μS). Figure 3f compares the experimentally measured accumulated currents and the theoretical or expected accumulated currents from this kernel for the vector-matrix multiplication operation. A total of 100 random input vectors were generated and applied, and each accumulated output was measured 20 times to capture read noise and approximate the true mean. The experimental measurements agree very well with theoretical estimates, demonstrating that the vector-matrix multiplication operation is implemented successfully within crossbars in our system. The limited precision of the ADCs and DACs on the prototyping system contribute to the underlying noise observable in the accumulated currents, but the system allows trading off between the dynamic range and precision of the induced currents via a simple resistance tuning process summarized in the Supplementary Material. While this agreement between theoretical and experimental accumulated currents from read-back device states showcases correctness and fidelity of memristive arrays in our system, it does not capture the nuanced disagreements and deviations arising from device defects or variabilities possibly tied to the limited precision and the employed margin-based programming scheme. These disagreements are directly related to the performance gaps between memristive neural networks and their software counterparts. We developed the layer ensemble averaging fault tolerance scheme to address this critical issue and discuss this next.

Simulation validation of layer ensemble averaging

To gauge the effectiveness of layer ensemble averaging in improving the performance of non-ideal hardware neural networks, we designed a simulation framework using PyTorch. This framework allows us to map pre-trained neural network solutions to a defective memristive system consisting of a specified number of crossbar arrays, stuck-at faults, read and write noise, input voltage and output current quantization, and operable device conductance ranges. In our simulations, we implement our proposed layer ensemble averaging (LEA) algorithm as well as two related algorithms from literature, *Mapping Algorithm with inner fault tolerant ability (MAO)*[27,33] and the *Committee Machines (CM)* algorithm[26]. We compare the algorithmic performance of these fault tolerance schemes against one another at similar levels of non-idealities for the same network architecture and pre-trained solutions. Although MAO has multiple variants, we compare against the version with redundant crossbars because it has the highest network performance gains compared to others in the original works. For clarity, we only include LEA results with the simple encoding algorithm here, but present complete results with the reduced mapping error encoding algorithm in Supplementary Fig. 2.

Figure 4 showcases comparative results on these algorithms on the MNIST handwritten digit classification task (networks detailed in Table 2). In our simulations, stuck-at high and stuck-at low faults are modeled as being equally likely, and the overall distribution of stuck-at defects is modeled as a uniform random distribution according to a specified percentage. Stuck-low and stuck-high devices have conductance 10 μS and 500 μS respectively, while devices that are not stuck can be written to one of two possible conductance states, 133 μS and 233 μS, in line with our experimental measurements from Fig. 3. To mimic system non-idealities, all reads are injected with a uniform random noise of ±10 μS, all writes are injected with zero-mean Gaussian noise with a standard deviation of 16.66 μS, and all input voltages and output current measurements are quantized to 12-bit fixed-point precision. As a result of these non-idealities, the mapping error between a software weight matrix $\mathbf{W_{ideal}}$ and its corresponding effective mapping $\mathbf{W_{mapped}}$ grows. To quantify algorithmic performance, we thus measure the inference performance of the neural network at increasing levels of redundancy (α) and the percentage of stuck devices on the simulated array, as well as the average mapping error from Eq. (2) between the ideal and mapped matrices for each layer. For MAO and LEA, the same solution is used for all redundant α mappings, while for CM, α unique solutions from the pre-trained pool of software solutions (Table 2) are mapped. The entire simulation is repeated across 10 independent cycles to sufficiently capture variations due to randomness. In addition, we choose $\beta = \alpha$ for LEA to ensure that LEA uses the same number of devices for mapping network parameters as the other algorithms for a given redundancy level α (for mapping a single layer of size $m \times n$, each scheme requires $2\alpha mn$ devices). To further ensure that all comparisons are direct and fair, we fix the strategy used for determining mapping locations for each fault tolerance scheme to the random strategy from Supplementary Algorithm 1.

Figure 4a summarizes how the MNIST handwritten digit classification dataset is passed to the neural network architecture. Figure 4b shows the network accuracy alongside the average mapping error as a function of increasing levels of device stuck-at faults and redundancy for each scheme. The software baseline accuracy is also included, which is the average performance of software solutions from Table 2. This baseline can alternately be interpreted as the accuracy that a perfect system with no non-idealities would attain, since such a system would not have any mapping error. Without any redundancy i.e., at $\alpha = 1$, the schemes exhibit very similar performances. As the percentage of stuck devices increases, the mapping error grows, causing a decline in test accuracy. This is expected since there are no redundant devices that could be used to compensate for non-idealities. At $\alpha = 3$,

https://doi.org/10.1038/s41467-025-56319-6

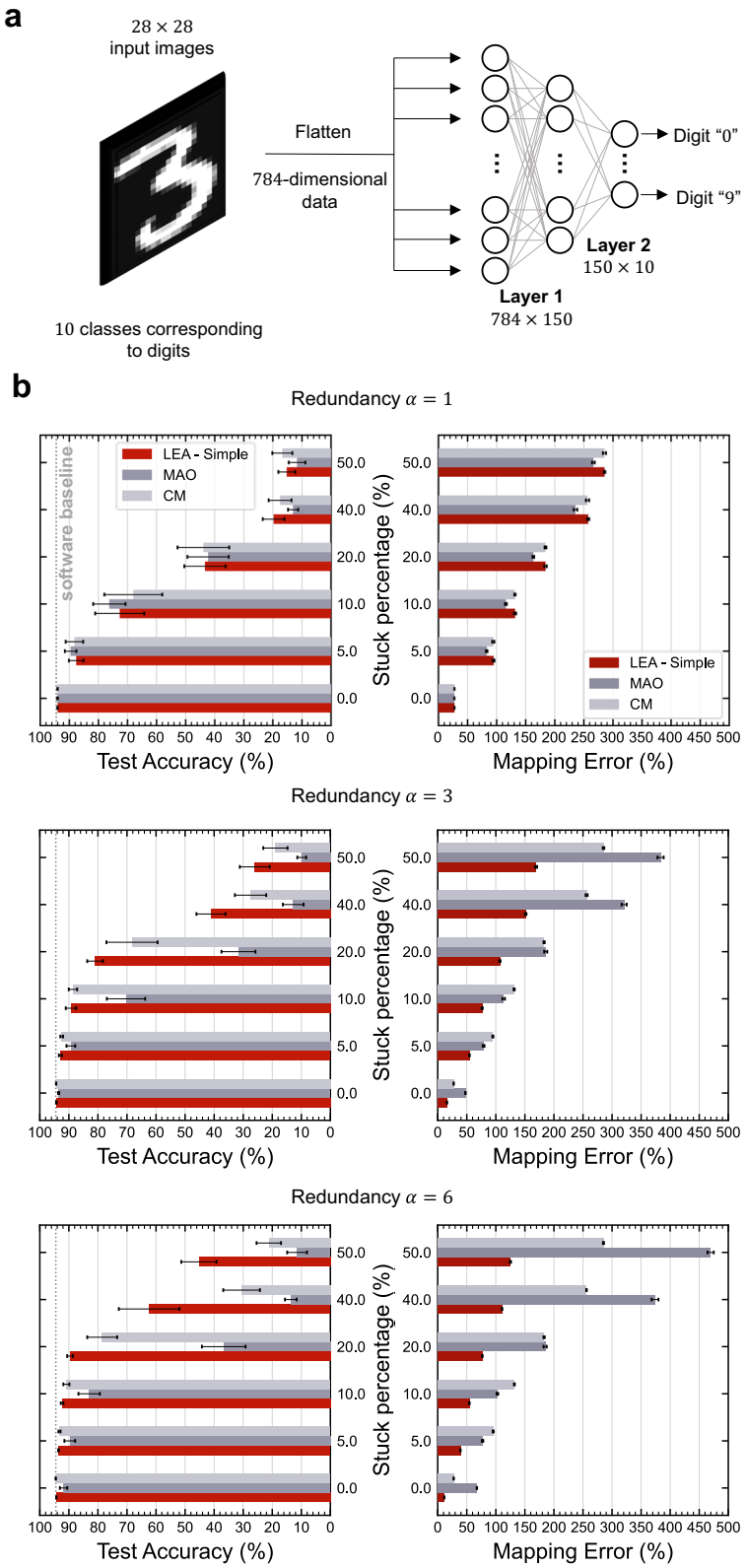

Fig. 4 | Comparative simulation results of hardware fault tolerance schemes on the MNIST handwritten digit classification task. a Overview of the MNIST dataset and employed architecture of the 2-layer perceptron network used for image classification. **b** Network test accuracies and mapping errors averaged across network layers at increasing levels of device stuck-at faults and the redundancy parameter α for each fault tolerance scheme (see Supplementary Table 1 and corresponding discussion for details). Bar heights correspond to averages and error bars correspond to standard deviations across 10 independent cycles of the entire simulation process. The proposed layer ensemble averaging scheme boasts a higher degree of fault tolerance compared to other hardware fault tolerance schemes, evident by lower mapping errors and higher test accuracies at similar levels of stuck-at faults and redundancy.

differences in the schemes are more apparent. As can be seen, the performance of the MAO algorithm degrades significantly, and while CM performs better than MAO, the proposed LEA algorithm exhibits the best performance with consistently lower mapping errors and higher network accuracies. This trend becomes even more apparent as we increase redundancy further. At $\alpha = 6$, our LEA fault tolerance scheme successfully tolerates even 20% devices with stuck-at faults, attaining a competitive accuracy of 89.6 ± 1.0% with a significantly lower mapping error than other schemes. Even at the extreme end where 50% of devices have stuck-at faults, our proposed LEA is capable of significantly outperforming both MAO and CM in terms of mapping error and as a result, the network performance. Since the mapping strategy is fixed, the differences in fault tolerance performance are a direct consequence of differences in how each scheme maps network parameters and polls output currents for subsequent layers.

The primary reason why MAO struggles to reduce the mapping error is the stuck-at fault formulation. Since stuck devices have conductances that are significantly beyond the conductance range of operable devices, the algorithm can no longer cancel out the contribution of a single stuck device in one conductance matrix by a single operable device in the other conductance matrix. The limited device conductance states also contribute to this. As a result, the fault tolerance performance of MAO can be erratic at low levels of redundancy. For example, with 10% defects, the network accuracy drops going from $\alpha = 1$ to $\alpha = 3$ instead of improving. It is only later when we allow for more redundancy that the algorithm recovers some fault tolerance performance, as can be seen at $\alpha = 6$. On the other hand, although the performance of CM is better than MAO, it still lags behind LEA. This is evident especially in cases where the stuck percentage is high. We attribute this to differences in the averaging process in the two schemes. In CM, outputs are averaged only at the final layer. Defective, uncorrected outputs from preceding layers, when sent directly to respective final layers as inputs would yield defective final predictions. Averaging final layer outputs does resolve this to some extent, but there is certainly a limit to this as particularly evident from the growth of the mapping error with CM at $\alpha = 3$ as well as $\alpha = 6$.

By correcting outputs via averaging for each layer, LEA outperforms both schemes. For a given stuck percentage and redundancy level, all inputs for network layers produced by LEA are significantly closer to the ground truth compared to inputs produced by MAO and CM. In general, we found that for a fixed redundancy level α>1, the rate at which the mapping error of LEA grows with increasing levels of stuck percentages is significantly lower than the respective rates of MAO and CM. This indicates that our layer-wise averaging scheme induces a much higher degree of fault tolerance in the memristive neural network compared to the other investigated schemes. We cover this in more detail in the *Discussion* section.

Hardware demonstration of layer ensemble averaging

Hardware results on layer ensemble averaging gathered using our mixed-signal prototyping platform are presented in Fig. 5. We study all possible combinations of mapping and encoding schemes at increasing values of redundancy parameters $\alpha \in [1, 4]$ and $\beta \in [1, \alpha]$. The multi-task Yin-Yang dataset and the 3-layer perceptron network architecture is showcased in Fig. 5a. For simplicity, a single solution from the pre-trained pool of solutions (see Table 2) is mapped according to a chosen mapping scheme and converted into device conductances according to a chosen encoding scheme. Then, the encoded solution is written to the 20,000-device array at appropriate physical locations using our margin-based switching algorithm presented previously in Fig. 3. Finally, multi-task test samples from the dataset are passed to the network as voltages, and the quality of layer ensemble averaging is evaluated by measuring the network's accuracy as well as layer-wise mapping errors for each configuration. The entire process is repeated for a total of 100 iterations to sufficiently capture variability.

Figure 5b compares the network test performance of all mapping and encoding configurations. We find that a higher degree of redundancy benefits network performance in all cases. When there is no averaging in the ensemble (i.e., $\alpha = \beta = 1$) and output currents from only a single row are used, the inference performance of the network is worse than even a linear solver (< 63.8%). However, as α increases, the performance increases, approaching the software baseline of 72% at (α, β) = (4,4) (which means that for each output of each layer, 4 rows are considered for averaging). At this highest investigated level of redundancy, all configurations achieve accuracies that are very close to the software baseline, verifying that our proposed layer ensemble averaging can successfully tolerate faults by mitigating the impact of device variabilities and system non-idealities that assist the network in performing better despite the challenging continual learning formulation of the problem. Corresponding mapping errors for each configuration are presented in Supplementary Fig. 3, where we show that the increased redundancy reduces the overall mapping error in line with our simulation results from Fig. 4. At a given α, the β parameter can be tuned to implement a trade-off between the overall network performance against the total number of devices that participate in the inference process.

The greedy mapping approach clearly yields solutions with more favorable and tighter distributions than random mapping, as indicated by the higher median accuracy of 71% compared to 69%. This can be attributed to the fundamental difference in how the search phase differs between the two. The greedy search is exhaustive and searches through all possible candidate locations for mapping each encoded conductance matrix on the physical chip. On the other hand, the random search is limited by the number of sampling iterations, which is fixed at 1000 for the results in Fig. 5b. Since the greedy search is exhaustive, it outputs the same mapping for each independent run, leading to tightly distributed mapping errors and network accuracies. By comparison, since random mapping only explores a fraction of possible mapping locations, it outputs differing mappings in each independent run with higher variabilities. This explains the larger distribution spread in mapping errors as well as network accuracies. Despite this lower search space size and thus faster algorithmic run time compared to greedy mapping, random mapping still produces solutions that have near-software inference performance at high levels of redundancy. If the number of sampling iterations for the random mapping are increased, the distribution of network accuracies would theoretically match that of greedy mapping, since a higher number of sampling iterations directly increases the search space. We offer experimental proof of this in Supplementary Fig. 4, where we showcase how the performance of random mapping improves as the number of sampling iterations increase.

Unlike our simulation results in Fig. 4, differences in the two encoding algorithms are much more subtle. Although there are cases where the reduced mapping error encoding algorithm leads to lower average mapping errors compared to the simple encoding algorithm (see discussion on Supplementary Fig. 3), the improvement does not manifest in terms of network performance, and the overall distributions are largely similar. We find that this is a result of our mapping algorithms being able to consistently find locations for redundant mappings that do not contain devices with stuck-at faults for the small network layers, which is made possible because our mapping algorithms rely on the SCV metric from Eq. (1), and stuck-at faults have a higher contribution to the SCV metric compared to other non-idealities such as device-to-device variability and read/write noise. If layer ensemble averaging mappings do not contain these stuck-at faults, the reduced mapping error encoding algorithm can only compensate for other non-idealities, which we know are tolerable by direct

https://doi.org/10.1038/s41467-025-56319-6

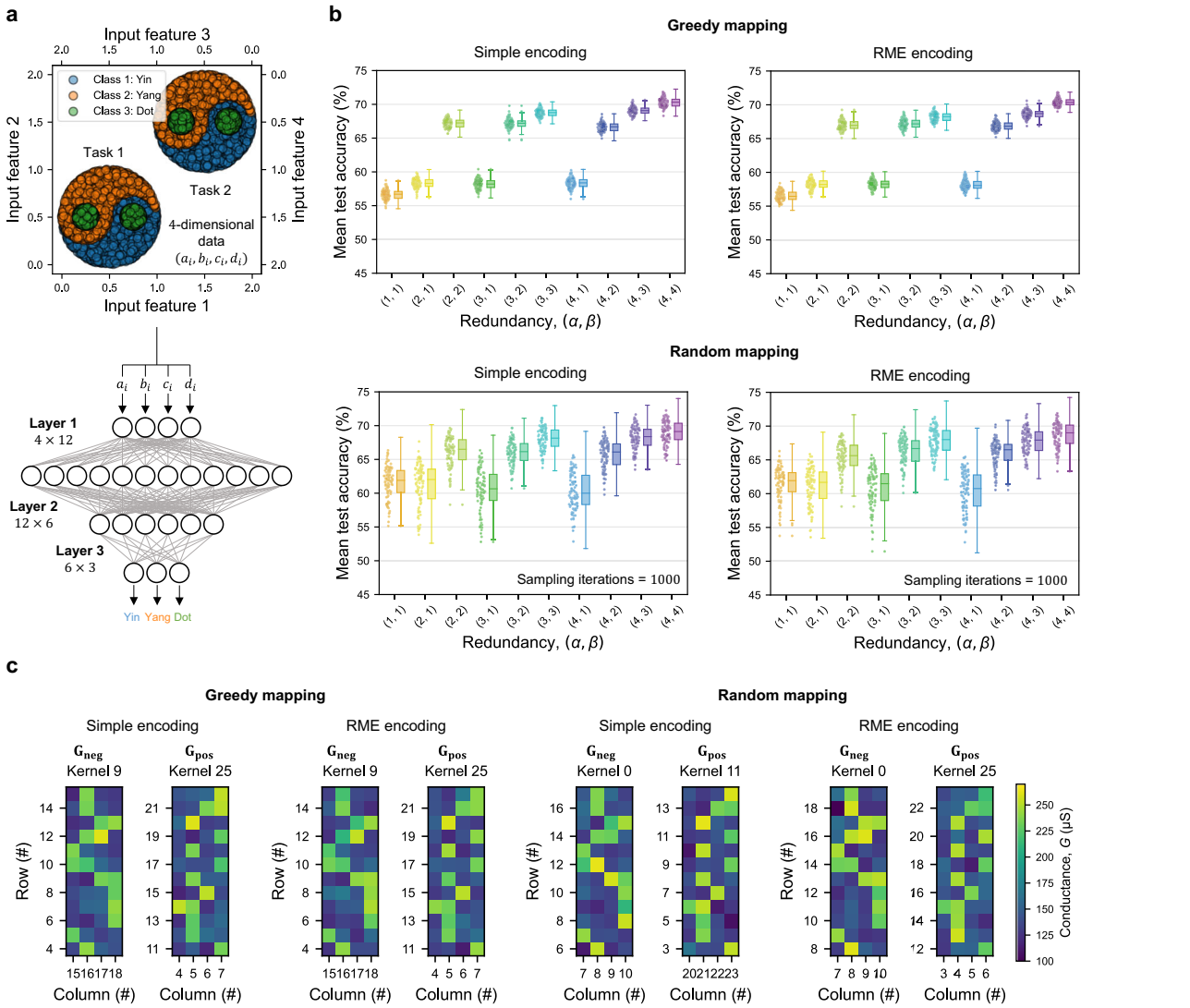

Fig. 5 | Hardware demonstration of layer ensemble averaging for the Yin-Yang classification problem. a Overview of the dataset and employed architecture of the 3-layer perceptron network used for multi-task classification. **b** Mean test accuracies for layer ensemble averaging at increasing values of redundancy parameters $\alpha \in [1, 4]$ and $\beta \in [1, \alpha]$ under different combinations of the greedy and random mapping algorithms with the simple and reduced mapping error (RME) encoding algorithms, where α indicates the total number of redundant mappings of each conductance matrix and β indicates how many rows (out of α) contribute to the current averaging process for each output. We include individual data points from 20 iterations as well as summary boxes to highlight the underlying distribution. Each box follows Tukey's rule, where the middle line indicates the median, box boundaries indicate first and third quartiles, and error bars indicate points at ± 1.5 inter-quartile range. **c** A subset of hardware mappings $\mathbf{G_{pos}}$ and $\mathbf{G_{neg}}$ of the first fully connected network layer (with dimensions 4 × 12) from a single cycle of the inference process for each configuration. The kernel, column, and row labels correspond to physical locations on the chip. All experimental results are gathered using the Daffodil mixed-signal prototyping platform.

averaging in the simple encoding scheme from results in Fig. 4. This also explains why the distribution of mapping errors with the reduced mapping error encoding is significantly better than that of simple encoding when sampling iterations are kept small (Supplementary Fig. 4).

Figure 5c presents a subset of hardware mappings $\mathbf{G_{pos}}$ and $\mathbf{G_{neg}}$ of the first fully connected layer for each configuration from a single iteration at the highest investigated redundancy level. The different configurations vary in where the matrices are mapped, as shown by the kernel, column, and row labels corresponding to physical locations on the 20,000-device array. They also differ in the encoded device conductance values due to the encoding algorithm, system-level non-idealities like read/write noise, and device-level non-idealities such as stuck-at faults and device-to-device variability. We include the full hardware mapping of each configuration separately in Supplementary Fig. 5, where these differences are much more evident.

Discussion

In summary, we proposed and demonstrated layer ensemble averaging (LEA)—a hardware-correction fault tolerance scheme that can improve the inference performance of memristive neural networks up to some software baseline. The scheme does not require any retraining after deployment to crossbars of emerging memory devices and does not make any assumptions on multi-bit device tunability, making it a versatile, generally applicable, device-agnostic fault tolerance scheme. We demonstrated that layer ensemble averaging outperforms both the Mapping Algorithm with Inner-Fault tolerance (MAO) and Committee Machines (CM)—two existing state-of-the-art hardware-based fault tolerance methods—by simulating a 2-layer perceptron network for classifying the MNIST handwritten digit dataset (see Fig. 4). Additionally, we experimentally validated the fault tolerance capability of layer ensemble averaging by mapping a 3-layer perceptron network to tackle a multi-task classification problem (see

Fig. 5) on our end-to-end mixed-signal prototyping platform featuring an in-house fabricated chip with 20,000 ReRAM devices.

MAO, CM, and LEA are all fault tolerance schemes that rely on device-level redundancy. The first algorithm, MAO, operates by mapping network parameters to devices such that each parameter is modeled by the summed conductance of multiple devices. For some software weight matrix $\mathbf{W_{ideal}}$, this leads to the expression $\mathbf{W_{ideal}} \propto \sum_{i=1}^{\alpha} \mathbf{G_{pos}^{(i)}} - \sum_{i=1}^{\alpha} \mathbf{G_{neg}^{(i)}}$, where α expresses the number of redundant crossbars and $\mathbf{G_{pos}^{(i)}}$ expresses the i-th $\mathbf{G_{pos}}$ conductance matrix (similarly for $\mathbf{G_{neg}}$). The impact of devices with stuck-at faults can be compensated by tuning operable devices participating in the mapping of a given parameter directly. This scheme works well if stuck-at fault conductance ranges are within the conductance ranges of operable devices, but we find that it becomes ineffective when stuck-at fault ranges are beyond the range of operable devices (which themselves have low tunability, as summarized in Supplementary Table 1). The second algorithm, CM, operates by mapping α unique solutions to the crossbar, each of which is mapped 1 time, in contrast to MAO where a single solution is mapped α times. When the final prediction must be made, output currents are averaged from all networks in the committee. Mathematically, this leads to the representation $\mathbf{W_{ideal,\,final}} \propto \frac{1}{\alpha}\left(\sum_{i=1}^{\alpha} \mathbf{G_{pos}^{(i)}} - \sum_{i=1}^{\alpha} \mathbf{G_{neg}^{(i)}}\right)$ for the final layer, and $\mathbf{W_{ideal}^{(i)}} \propto \mathbf{G_{pos}^{(i)}} - \mathbf{G_{neg}^{(i)}}$ for the i-th mapping of preceding layers. This scheme works well when the trained solutions and devices have a high number of states, but we find that the performance degrades when the tunability is limited and stuck-at faults are high. Finally, our algorithm LEA operates by mapping the same solution α times to memristive crossbars, similar to MAO. Unlike CM, the averaging happens at each layer instead of just the final layer. The mathematical representation is thus $\mathbf{W_{ideal}} \propto \frac{1}{\beta}\left(\sum_{i=1}^{\beta} \mathbf{G_{pos}^{(i)}} - \sum_{i=1}^{\beta} \mathbf{G_{neg}^{(i)}}\right)$ for each layer, where $\beta \leq \alpha$ (see *Methods: Layer Ensemble Averaging* for a detailed discussion). Although this averaging incurs additional overhead compared to MAO and CM, we showed that our proposed algorithm LEA leads to a consistently higher degree of fault tolerance and is thus a suitable scheme for memristive neural network implementations where practical non-idealities are present.

The overheads in LEA are still comparable to MAO and CM. At similar levels of redundancy, all schemes require the same number of devices. For a mixed-signal system such as our Daffodil prototype that we utilize for experimental validation in this work, there is no additional hardware overhead. This is because ADCs and DACs are reconfigurable, and inference operations such as current summation and averaging can be done in software on the hard processor before being sent to subsequent analog layers. In such an implementation, the only overhead is in terms of time complexity. For an implementation where all operations are implemented in hardware, layer ensemble averaging would have a constant time overhead because of the scaling operation for averaging that must happen at each layer after current summation, compared to no averaging in MAO and averaging only at the final layer in CM. The scaling unit in hardware can be time-shared across various layers, meaning that the hardware overhead of layer ensemble averaging would still be the same as CM. We argue that this would be an acceptable overhead given the performance gains that are possible with layer ensemble averaging, as reported in our simulation results in Fig. 4 as well as experimental results in Fig. 5.

Furthermore, the redundancy parameters α and β can be used to implement a trade-off between the performance of the memristive neural network against the total number of devices that participate in mapping each weight matrix. Provided enough devices are available for the mapping to succeed, this trade-off can bring the performance of the memristive network on-par with its software counterpart. Although the exact level of redundancy required to attain near-software inference performance depends on the network architecture and task under investigation and layer-wise sensitivities to faults, layer ensemble averaging can be employed as a general hardware-correction technique with even a limited redundancy level to correct vector-matrix multiplication outputs and improve network performance. Additionally, it can be paired with software-correction techniques for additional fault tolerance if needed, since software-correction would be an orthogonal solution. By design, layer ensemble averaging assumes that hardware-redundancy is possible. However, pairing layer ensemble averaging with a software-correction scheme can assist in the case where the number of devices required to map parameters for a certain layer redundantly exceed the physical number of available devices. In an extreme situation where no block-wise redundancy can be implemented due to hardware limitations, one could relax the mapping phase to allow for row-wise duplication, as discussed in the *Methods: Layer Ensemble Averaging* section. In conjunction with a software-correction scheme, another workaround for added robustness can be to average currents from the same mapping multiple times per layer. This can be accomplished with minimal changes to the system controller.

Our results provide an experimental proof-of-concept demonstration of the possibility of performing accurate vector-matrix multiplications in non-ideal memristive crossbars. We showed that layer ensemble averaging is a versatile training-free fault tolerance scheme that can improve the performance of memristive neural networks irrespective of the underlying device technology. We presented several variants suitable for different situations. If optimizing network performance is the priority, greedy mapping is the better candidate. If device defects are known to be uniformly distributed or if fast algorithmic run time is a requirement, then random mapping is a better candidate as a single sampling iteration would produce mappings that have a similar statistical distribution of defects compared to the exhaustive greedy search. Similarly, simple encoding is generally the better candidate since it has a faster algorithmic run time compared to the reduced mapping error encoding, but the latter can be useful algorithm if the application would benefit from slightly better mapping errors. A multi-objective optimization framework for producing layer ensemble averaging mappings at redundancy levels proportional to layer-wise sensitivities could be devised in the future. While this work focused exclusively on utilizing layer ensemble averaging for fully connected neural network inference with ternary weights, the methodology is broadly applicable to a variety of architectures (such as convolutional layers, recurrent layers, transformer layers, etc.) and domains requiring accurate multiply-and-accumulate and vector-matrix multiplication operations. Dedicated memristive hardware accelerators for such cases could be envisioned in the future owing to their power consumption and energy benefits compared to traditional computing systems, and the proposed layer ensemble averaging approach could help mitigate their defects accordingly.

Methods

Layer ensemble averaging

The core idea behind layer ensemble averaging is that by discarding outputs from severely defective rows and averaging over relatively fewer defective rows, we can reduce the effective mapping error and approximate the true output of a neural network layer implemented by non-ideal memristive devices. The overall operation can be explained by addressing two key questions: where should an ideal weight matrix $\mathbf{W_{ideal}} \in \mathbb{R}^{m \times n}$ be mapped, and how should $\mathbf{W_{ideal}}$ be encoded in device conductances $\mathbf{G_{pos}} \in \mathbb{R}^{m \times n}$ and $\mathbf{G_{neg}} \in \mathbb{R}^{m \times n}$?

To answer the first question, layer ensemble averaging finds locations that would have the least *summed conductance variation*

(SCV) from some ideal target conductance matrix $\mathbf{G_{ideal}}$ (either $\mathbf{G_{pos}}$ or $\mathbf{G_{neg}}$). We define SCV as

$$SCV_k^{(i)} = \sum_{j=1}^{n} \left| G^{(ij)} - A_k^{(ij)} \right| \quad (1)$$

where $SCV_k^{(i)}$ is the sum of absolute differences between devices on the i-th row of $\mathbf{G_{ideal}}$, and the i-th row of the k-th non-ideal conductance matrix $\mathbf{A_k} \in \mathbb{R}^{m \times n}$ determined either by writing to and reading from the crossbar, or by conductance map information measured prior to device programming. For comparing different candidate mappings, the SCV can be summed across these m contiguous rows. This approach provides a quick method for estimating the relative quality of candidate mappings, as it effectively captures the impact of device non-idealities on mapped network weights.

For the i-th row of the final output, corresponding SCV values can be used to determine which rows from the ensemble participate in the current averaging process during inference. This is a one-time process and can be done prior to the inference operation. For each row in the original weight matrix, the layer ensemble mapping algorithm finds α rows, out of which β are utilized during inference ($1 \le \beta \le \alpha$). In general, a weight matrix $\mathbf{W_{ideal}}$ is mapped to α conductance matrices $\mathbf{G_{pos}}$ and α conductance matrices $\mathbf{G_{neg}}$, and β is a hyperparameter that can be used to turn off some number of defective rows per output.

The full mapping algorithm is presented in Supplementary Algorithm 1. There are two modes of operation: *random* and *greedy*. In the random mode of operation, the algorithm finds random (uniform) non-conflicting locations on the crossbar by random sampling, while in the second mode of operation, it attempts an exhaustive greedy search. In both cases, it finds mappings with the least summed conductance variation from $\mathbf{G_{ideal}}$ as captured by Eq. (1). For a given weight matrix $\mathbf{W_{ideal}}$, the algorithm must be called for ideal $\mathbf{G_{pos}}$ as well as $\mathbf{G_{neg}}$. The random mode can be much faster than the greedy mode (when the number of sampling iterations is smaller than the number of available devices) and is particularly useful if chip defects are known to be distributed uniformly, while the greedy mode can be useful when the defect map distribution cannot be easily inferred, or when a higher run-time is acceptable. A bipartite-matching based operation could also be envisioned, and although it has been investigated before in terms of singular row-by-row mapping[45–47], we defer this as a possible future research direction due to the cubic time complexity it would incur in determining mappings for each redundant conductance matrix compared to the quadratic time complexity of our greedy algorithm. In our formulation of the mapping process, finding a crossbar mapping for a particular weight matrix is independent of layer dimensions. This is because the acceptance criteria based on the SCV metric is relative, and this not only ensures that the algorithm is scalable to larger networks, but also to different device technologies with varying dynamic ranges.

Compared to other schemes in literature[45,48], we map weight matrices as contiguous blocks instead of varying rows or lines on the crossbar. We do this to avoid additional overheads and to maintain compatibility with a future scheme for training these redundant mappings directly on the crossbar via outer product operations. However, this mapping constraint can be relaxed to allow for row-wise duplication instead of block-wise if there is a need, for example, due to a physical limitation on available devices. Additional constraints can also be added to the algorithm. For instance, mappings could be constrained to be on non-conflicting rows. Provided the mapping is successful, this would allow the produced mappings to be used in parallel (i.e., input voltages can be provided in parallel to the redundant mappings in a layer ensemble, and output currents can be measured in parallel).

For the second question, we provide two variants of encoding $\mathbf{W_{ideal}}$ into device conductances $\mathbf{G_{pos}}$ and $\mathbf{G_{neg}}$. The first variant, *simple*, is summarized in Table 1. Here, all α copies of $\mathbf{G_{pos}}$ are kept the same and all α copies of $\mathbf{G_{neg}}$ are kept the same. The second variant, *reduced mapping error*, uses the simple variant as an initialization and sequentially updates device conductances based on information about each target parameter, as presented in Supplementary Algorithm 2. Here, the goal is to reduce the effective mapping error between the original weight matrix $\mathbf{W_{ideal}}$ and the mapped matrix $\mathbf{W_{mapped}}$ (Supplementary Algorithm 2 Line 7), defined as

Table 1 | Simple encoding scheme

W_{ideal}	G_{pos} (all redundant α mappings)	G_{neg} (all redundant α mappings)
>0	G_{ON}	G_{OFF}
0	G_{ON}	G_{ON}
<0	G_{OFF}	G_{ON}

$$Mapping\ Error = \frac{||\mathbf{W_{mapped}} - \mathbf{W_{ideal}}||}{||\mathbf{W_{ideal}}||} \quad (2)$$

We illustrate the principle of our layer ensemble averaging fault tolerance scheme in Fig. 1 for an example ternary weight matrix $\mathbf{W_{ideal}}$. A consequence of the differential encoding scheme, $\mathbf{W} \propto (\mathbf{G_{pos}} - \mathbf{G_{neg}})$, is that software layer dimensions double when mapped to hardware i.e., each weight is represented by a pair of devices (one in $\mathbf{G_{pos}}$ and the other in $\mathbf{G_{neg}}$). For neural network demonstrations in this work, the ternary quantized weights $-\eta$, 0, $+\eta$ (η differs among layers, as shown in Supplementary Fig. 6) are represented by device pairs in conductance states (G_{OFF}, G_{ON}), (G_{ON}, G_{ON}) and (G_{ON}, G_{OFF}) respectively. For consistency, inputs are always applied on columns and outputs are always measured from the rows of the crossbar for this work. However, this is not a rigid requirement and can be altered if allowed by the system.

For the example in Fig. 1, a layer ensemble mapping for $\mathbf{G_{pos}}$ is shown with $\alpha = 3$, indicating that the weight matrix (represented by conductance matrices $\mathbf{G_{pos}}$ and $\mathbf{G_{neg}}$) is mapped on to the crossbar three times with $\mathbf{G_{pos}^{(i)}}$ representing the i-th mapping of $\mathbf{G_{pos}}$ (similarly for $\mathbf{G_{neg}}$), and $\beta = 2$, indicating that from this layer ensemble of size 3, currents from exactly 2 active rows (based on the SCV metric) will be averaged for each output. For the vector-matrix multiplication process, an input vector $\mathbf{X}$ can be converted to voltages by $\mathbf{V_{in}} = V_{read} \cdot \mathbf{X}$, where V_{read} is the voltage used for reading operations on the crossbar. For each output, only currents from active rows are considered for averaging. In the presented example, the layer ensemble output is thus given by

$$\mathbf{I_{pos}} = \begin{bmatrix} I_1 & I_2 \end{bmatrix} = \begin{bmatrix} \frac{i_1^{(2)} + i_1^{(3)}}{\beta} & \frac{i_2^{(1)} + i_2^{(2)}}{\beta} \end{bmatrix} \quad (3)$$

and the final vector-matrix multiplication output is given by

$$\mathbf{XW_{ideal}} \approx \frac{\mathbf{V_{in}}}{V_{read}} \cdot \frac{\left(\mathbf{G_{pos}} - \mathbf{G_{neg}}\right)}{G_{norm}} = \frac{\mathbf{I_{pos}} - \mathbf{I_{neg}}}{G_{norm} \cdot V_{read}} \quad (4)$$

where $\mathbf{I_{pos}}$ is the final output current vector from the ensemble mappings for $\mathbf{G_{pos}}$, I_i is the averaged current for row i produced by the layer ensemble, $i_i^{(j)}$ is the output current from the j-th mapping of $\mathbf{G_{pos}}$ for I_i, and G_{norm} is a scaling constant approximated by the difference of the experimentally measured low conductance state G_{OFF} from the high conductance state G_{ON} averaged over devices present in the layer ensemble mapping $(G_{norm} = G_{ON} - G_{OFF})$. Averaging currents over β rows with lower SCV mitigates the impact of device-to-device variability and noise, while skipping over the $\alpha - \beta$ rows with higher SCV mitigates the impact of stuck and faulty devices. These non-

idealities can drastically reduce network performance if not accounted for as they cause hardware outputs to deviate from software.

For hardware neural network results in this work, the vector-matrix multiplication operations for each network layer are implemented on the physical chip. Other operations such as non-linear activation functions and current averaging are implemented in software using the host system. For reading, a low voltage of 0.3 V is used to minimize unwanted disturbances to the memristor conductances. Values of G_{OFF} and G_{ON} are chosen as 133 µS and and 233 µS based on current *vs.* voltage sweep measurements on the physical chip (presented in Fig. 3). We note here that layer ensemble averaging is not tied to ternary weight matrices. On the contrary, we use ternary weights to demonstrate the universality of the fault tolerance scheme and its effectiveness even when devices have limited tunability (1-bit).

Neural network details

The first dataset used for this work is derived from the Yin-Yang dataset[44] containing 4-dimensional input features and 3 output classes: *Yin*, *Yang*, and *Dot*. Each sample in the dataset represents a point in a two-dimensional representation of the Yin-Yang symbol, and the task is to classify samples according to their position within the symbol. This dataset serves as a good proof-of-concept problem for early-stage hardware benchmarking efforts because it is small compared to alternative classic datasets and exhibits a clear gap between inference test accuracies attainable by shallow networks or linear solvers (63.8 %) compared to deep neural networks because of non-linear decision boundaries. The dataset is generated by randomly sampling values a_i and b_i in the feature domain [0, 1] that satisfy a set of mathematical equations defining the Yin-Yang characteristic shape. The values $(1-a_i, 1-b_i)$ are also included, leading to a representation $(a_i, b_i, 1-a_i, 1-b_i)$ in the 4-dimensional feature space. Using this dataset, a continual learning problem is obtained by converting it to a multi-task classification problem. It is derived by firstly broadening the feature domain to [0, 2]. Then, *Task 1* instances are generated using the same mathematical equations as the original dataset, and *Task 2* instances are generated by adding a constant offset (of 1) to instances produced by the same equations. In effect, this creates two individual Yin-Yang classification problems. This is motivated by choices in continual learning literature where problems of similar difficulty are studied for consistency[49–51]. For effective continual learning, a network must be trained on the tasks sequentially and must retain classification performance on *Task 1* as it learns to classify *Task 2*.

A 3-layer fully connected perceptron network with full-precision weights and learnable biases is trained in software using elastic weight consolidation[49] and then quantized to a ternary weight space using block reconstruction quantization (BRECQ)[52] for memristive hardware deployment and verification. For neural network results reported in this work, a single solution that performs better than the linear solver on both tasks is quantized and mapped to layer ensembles for simplicity. Quantization-aware training schemes were investigated as well[53,54], but they failed to yield good continual learning results with elastic weight consolidation. We hypothesize this occurs because the loss landscape changes drastically when training neural networks directly in the ternary-weight domain, and elastic weight consolidation constraints restrict such highly quantized networks from effectively learning.

The second dataset used for this work is the MNIST dataset which consists of 28 × 28 pixel images of handwritten digits[39]. The dataset is normalized by subtracting a constant (0.1307) and dividing the difference by a fixed scale factor (0.3801). This shift and scaling were determined as the mean and standard deviation of all pixels in the 60,000 images of the original MNIST training set. This normalized dataset is flattened, batched, and then utilized to train a 2-layer fully connected perceptron network for image classification. Contrary to the Yin-Yang network, here we use a quantization-aware training scheme based on the weight, activation, gradient, error quantization (WAGE) framework[55] (with the 2-8-8-8 configuration), which enforces weights to remain in the ternary domain during training.

The neural network architectures are further detailed in Table 2 and Supplementary Fig. 6. The choice of a continual learning problem is motivated by the fact that it provides a more rigorous evaluation of the robustness of our scheme compared to simpler problems such as single-task classification. This is because continual learning effectively requires squeezing in more functionality into a network with a fixed capacity[49], thereby placing greater demands on the performance of layer ensemble averaging. It can also be implemented on our hardware prototyping system. On the other hand, while the larger network for MNIST classification cannot be demonstrated on our hardware due to network parameters exceeding the count of available devices, it still allows us to evaluate the scalability of our scheme on larger, more diverse workloads (varying quantization schemes, network scales, problem type, etc.) and aids in the comparison with other hardware-correction schemes from literature.

Experimental setup

To experimentally evaluate the effectiveness of layer ensemble averaging, we utilize our custom mixed-signal prototyping system, which we named Daffodil[38]. This experimental setup is shown in Fig. 2. It consists of a custom complementary metal-oxide-semiconductor (CMOS)/memristor integrated circuit or chip, a custom mixed-signal PCB named the Daffodil board, and a Zynq-based FPGA development board that acts as the host. The chip with foundry 180 nm CMOS and in-house integrated ReRAM devices is packaged and connects to the PCB via a 21 × 21 pin Ceramic Pin Grid Array (CPGA) package, and the PCB connects to the FPGA via a fully populated FPGA mezzanine card (FMC) connector. The ReRAM devices were fabricated in house on top of commercial 180 nm node CMOS wafers. The wafers, fabricated up through metal 5 of a 6-metal layer process, were pulled after the via 5-to-6 tungsten damascene step. The resulting open vias surrounded by a planarized dielectric surface tended to have an average roughness of <2 nm, ideal for in-house device integration. The devices were fabricated via three lithography steps and appropriate material deposition and etching for the bottom electrode, active material and top electrode and the contact pads. More details are included in the Supplementary Material.

At its core, the Daffodil system is an integrated mixed-signal vector-matrix multiplication processor acting as a neural network accelerator based on emerging two-terminal memristor devices. The crossbar array consists of a total of 20,000 2T-1R devices which are organized into 32 25 × 25 subarrays called *kernels*. Daffodil provides access to the kernels via five external CMOS logic signals. Each kernel is controlled by 75 distinct DAC channels, allowing parallel biasing for

Table 2 | Summary of the investigated neural network architectures

Dataset	Network Architecture	Biases	Activations	Quantization	Optimizer	Test Accuracy
Yin-Yang (2-task variant)	4 × 12 × 6 × 3	Y	Tanh	BRECQ[52]	Adam with elastic weight consolidation[49]	72.85 ± 1.25
MNIST	784 × 150 × 10	N	ReLU	WAGE[55]	Stochastic gradient descent	94.49 ± 0.12

The test accuracy is reported as mean ± standard deviation across 4 and 8 independent runs for the Yin-Yang and MNIST networks respectively.

each kernel row, column, and gate. The PCB also has 25 reconfigurable ADC channels for measuring currents in parallel on kernels via dedicated transimpedance amplifiers. The reference voltages to these amplifiers can be tuned via a separate DAC channel, and the feedback resistance for each amplifier can be tuned via dedicated digital potentiometers. There are three possible configurations based on ADC and DAC connections suited for different applications. These are summarized in Supplementary Fig. 7. A hard processor on the FPGA development board runs a custom Linux operating system built using Xilinx's PYNQ framework[56]. The programmable logic contains custom register-transfer level code for important use cases such as timed pulse generation, read and write operations, and kernel selection.

The primary library, *daffodil-lib*, acts as the system's control software and design verification framework and handles hardware communication as well as hardware-accurate simulation and modeling. The secondary library, *daffodil-app*, utilizes primitives from *daffodil-lib* and provides complex applications to the user. All experimental results reported in this work were extracted using these libraries. Hardware vector-matrix multiplication operations were implemented on physical crossbars on the 20,000-device chip, and other operations such as network activation functions and layer ensemble averaging were implemented in software directly on the host FPGA development board. The system simplifies studying hardware-aware algorithms for device resistive state tuning and/or neural network mapping, and hardware-software results can be easily verified jointly under the same platform by toggling between the hardware and simulation classes given by *daffodil-lib*. Further details of the prototyping system are presented in Supplementary Material. An overview of the hardware-software architecture is also presented in Supplementary Fig. 8.

Data availability

The data that supports plots within this paper and other findings of this study is available in the provided code & data repository (ref. 57 and https://github.com/ADAM-Lab-GW/layer-ensemble-averaging). Additional data related to this study can be made available from the corresponding authors upon request. The MNIST handwritten digit dataset used for simulations is available at: https://yann.lecun.com/exdb/mnist/. Code for the original Yin-Yang dataset is available at: https://github.com/lkriener/yin_yang_data_set.

Code availability

The code that supports plots within this paper is available in the provided repository (available on Zenodo, see ref. 57 and https://github.com/ADAM-Lab-GW/layer-ensemble-averaging). A reference implementation of layer ensemble averaging is also provided.

References

1. Wang, H. et al. BitNet: Scaling 1-bit Transformers for Large Language Models. Preprint at https://doi.org/10.48550/arXiv.2310.11453 (2023).
2. Ma, S. et al. The Era of 1-bit LLMs: all large language models are in 1.58 Bits. Preprint at http://arxiv.org/abs/2402.17764 (2024).
3. Zhang, Y., Cui, M., Shen, L. & Zeng, Z. Memristive quantized neural networks: a novel approach to accelerate deep learning on-chip. *IEEE Transac. Cybernetics* **51**, 1875–1887 (2021).
4. Valentian, A. et al. Fully integrated spiking neural network with analog neurons and RRAM Synapses. In *2019 IEEE International Electron Devices Meeting (IEDM)* 14.3.1-14.3.4. https://doi.org/10.1109/IEDM19573.2019.8993431 (2019).
5. Prezioso, M. et al. Spiking neuromorphic networks with metal-oxide memristors. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)* 177–180. https://doi.org/10.1109/ISCAS.2016.7527199 (2016).
6. Li, C. et al. Analogue signal and image processing with large memristor crossbars. *Nat. Electron* **1**, 52–59 (2018).
7. Adam, G. C. et al. 3-D memristor crossbars for analog and neuromorphic computing applications. *IEEE Trans. Electron Devices* **64**, 312–318 (2017).
8. Hu, M. et al. Memristor crossbar-based neuromorphic computing system: a case study. *IEEE Transac. Neural Netw. Learn. Syst.* **25**, 1864–1878 (2014).
9. Chen, Y. ReRAM: history, status, and future. *IEEE Transac. Electron Devices* **67**, 1420–1433 (2020).
10. Hu, M. et al. Memristor-based analog computation and neural network classification with a dot product engine. *Adv. Mater.* **30**, 1705914 (2018).
11. Burr, G. W. et al. Experimental demonstration and tolerancing of a large-scale neural network (165 000 Synapses) using phase-change memory as the synaptic weight element. *IEEE Transac. Electron Devices* **62**, 3498–3507 (2015).
12. Fick, L. et al. Analog in-memory subthreshold deep neural network accelerator. In *2017 IEEE Custom Integrated Circuits Conference (CICC)* 1–4. https://doi.org/10.1109/CICC.2017.7993629 (2017).
13. Harris, N. C. et al. Linear programmable nanophotonic processors. *Opti. Opt.* **5**, 1623–1631 (2018).
14. Kingra, S. K. et al. Methodology for realizing VMM with binary RRAM Arrays: experimental demonstration of Binarized-ADALINE using OxRAM Crossbar. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5. https://doi.org/10.1109/ISCAS45731.2020.9180915 (2020).
15. Long, Y. et al. A Ferroelectric FET-Based processing-in-memory architecture for DNN Acceleration. *IEEE J. Explor. Solid-State Comput. Devices Circ.* **5**, 113–122 (2019).
16. Migliato Marega, G. et al. A large-scale integrated vector–matrix multiplication processor based on monolayer molybdenum disulfide memories. *Nat. Electron* 1–8 https://doi.org/10.1038/s41928-023-01064-1 (2023).
17. Bonnet, D. et al. Bringing uncertainty quantification to the extreme-edge with memristor-based Bayesian neural networks. *Nat. Commun.* **14**, 7530 (2023).
18. Liu, S. et al. An ultrasmall organic synapse for neuromorphic computing. *Nat. Commun.* **14**, 7655 (2023).
19. Wan, W. et al. A compute-in-memory chip based on resistive random-access memory. *Nature* **608**, 504–512 (2022).
20. Xue, C.-X. et al. 15.4 A 22nm 2Mb ReRAM Compute-in-Memory Macro with 121-28TOPS/W for Multibit MAC Computing for Tiny AI Edge Devices. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)* 244–246. https://doi.org/10.1109/ISSCC19947.2020.9063078 (2020).
21. Cai, F. et al. A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. *Nat. Electron* **2**, 290–299 (2019).
22. Kaiser, J. et al. Hardware-aware in situ learning based on stochastic magnetic tunnel junctions. *Phys. Rev. Appl.* **17**, 014016 (2022).
23. Zhu, Z. et al. MNSIM 2.0: a Behavior-Level modeling tool for memristor-based neuromorphic computing systems. In *Proc. 2020 on Great Lakes Symposium on VLSI* 83–88 (Association for Computing Machinery, New York, NY, USA). https://doi.org/10.1145/3386263.3407647 (2020).
24. Yi, S., Kendall, J. D., Williams, R. S. & Kumar, S. Activity-difference training of deep neural networks using memristor crossbars. *Nat. Electron* **6**, 45–51 (2023).
25. Fang, Y. et al. Improvement of HfOx-Based RRAM device variation by Inserting ALD TiN Buffer Layer. *IEEE Electron Device Lett.* **39**, 819–822 (2018).
26. Joksas, D. et al. Committee machines—a universal method to deal with non-idealities in memristor-based neural networks. *Nat. Commun.* **11**, 4273 (2020).

27. Xia, L. et al. Stuck-at fault tolerance in RRAM computing systems. *IEEE J. Emerg. Selected Topics Circ. Syst.* **8**, 102–115 (2018).
28. Zhao, J. et al. Gradient decomposition methods for training neural networks with non-ideal synaptic devices. *Front. Neurosci.* **15**, 749811 (2021).
29. Borders, W. A. et al. Measurement-driven neural-network training for integrated magnetic tunnel junction arrays. *Phys. Rev. Appl.* **21**, 054028 (2024).
30. Kaleem, M. A., Cai, J., Amirsoleimani, A. & Genov, R. A Survey of ensemble methods for mitigating memristive neural network non-idealities. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5. https://doi.org/10.1109/ISCAS46773.2023.10181553 (2023).
31. Li, Y. et al. Mixed-precision continual learning based on computational resistance random access memory. *Adv. Intell.Syst.* **4**, 2200026 (2022).
32. Karunaratne, G. et al. In-memory realization of in-situ few-shot continual learning with a dynamically evolving explicit memory. In *ESSCIRC 2022- IEEE 48th European Solid State Circuits Conference (ESSCIRC)* 105–108. https://doi.org/10.1109/ESSCIRC55480.2022.9911329 (2022).
33. Huangfu, W. et al. Computation-oriented fault-tolerance schemes for RRAM computin\g systems. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)* 794–799. https://doi.org/10.1109/ASPDAC.2017.7858421 (2017).
34. Kopperberg, N. et al. Endurance of 2 Mbit Based BEOL Integrated ReRAM. *IEEE Access* **10**, 122696–122705 (2022).
35. Grossi, A. et al. Resistive RAM endurance: array-level characterization and correction techniques targeting deep learning applications. *IEEE Transac. Electron Devices* **66**, 1281–1288 (2019).
36. Wang, Y. et al. Compact model of dielectric breakdown in spin-transfer torque magnetic tunnel junction. *IEEE Transac. Electron Devices* **63**, 1762–1767 (2016).
37. Ikeda, S. et al. Magnetic tunnel junctions for spintronic memories and beyond. *IEEE Transac. Electron Devices* **54**, 991–1002 (2007).
38. Hoskins, B. et al. A system for validating resistive neural network prototypes. In *International Conference on Neuromorphic Systems 2021* 1–5 (ACM, Knoxville TN USA). https://doi.org/10.1145/3477145.3477260 (2021).
39. Deng, L. The MNIST database of handwritten digit images for machine learning research [Best of the Web]. *IEEE Signal Process. Magazine* **29**, 141–142 (2012).
40. Roth, R. M. Fault-Tolerant Dot-Product Engines. *IEEE Transac. Inform. Theory* **65**, 2046–2057 (2019).
41. Liu, T. et al. A fault-tolerant neural network architecture. In *Proc. 56th Annual Design Automation Conference 2019* 1–6 (Association for Computing Machinery, New York, NY, USA). https://doi.org/10.1145/3316781.3317742 (2019).
42. He, Z., Lin, J., Ewetz, R., Yuan, J.-S. & Fan, D. Noise injection adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping. In *Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19)* **57**, 1–6 (Association for Computing Machinery, New York, NY, USA). https://doi.org/10.1145/3316781.3317870 (2019).
43. Quan, C. et al. Training-Free Stuck-At Fault Mitigation for ReRAM-Based Deep Learning Accelerators. *IEEE Transac. Computer-Aided Des. Integrated Circ. Syst.* **42**, 2174–2186 (2023).
44. Kriener, L., Göltz, J. & Petrovici, M. A. The Yin-Yang dataset. In *Proc. 2022 Annual Neuro-Inspired Computational Elements Conference* 107–111 (Association for Computing Machinery, New York, NY, USA). https://doi.org/10.1145/3517343.3517380 (2022).
45. Chen, L. et al. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017* 19–24. https://doi.org/10.23919/DATE.2017.7926952 (2017).
46. Xu, Q. et al. Fault tolerance in memristive crossbar-based neuromorphic computing systems. *Integration* **70**, 70–79 (2020).
47. Xu, Y., Jin, S., Wang, Y. & Qi, Y. Aggressive fault tolerance for memristor crossbar-based neural network accelerators by operational unit level weight mapping. *IEEE Access* **9**, 102828–102834 (2021).
48. Liu, B. et al. Vortex: Variation-aware training for memristor X-bar. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)* 1–6. https://doi.org/10.1145/2744769.2744930 (2015).
49. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl Acad. Sci.* **114**, 3521–3526 (2017).
50. Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv.org* https://arxiv.org/abs/1706.05098v1 (2017).
51. Zhang, Y. & Yang, Q. An overview of multi-task learning. *Natl. Sci. Rev.* **5**, 30–43 (2018).
52. Li, Y. et al. BRECQ: pushing the Limit of Post-Training Quantization by Block Reconstruction. In *International Conference on Learning Representations*. https://openreview.net/forum?id=POWv6hDd9XH (2020).
53. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R. & Bengio, Y. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. Preprint at https://doi.org/10.48550/arXiv.1602.02830 (2016).
54. Li, F., Liu, B., Wang, X., Zhang, B. & Yan, J. Ternary Weight Networks. Preprint at https://doi.org/10.48550/arXiv.1605.04711 (2022).
55. Wu, S., Li, G., Chen, F. & Shi, L. Training and Inference with Integers in Deep Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJGXzmspb (2018).
56. PYNQ - Python productivity for Zynq. *PYNQ - Python productivity for Zynq* http://www.pynq.io/.
57. Yousuf, O. et al. Layer Ensemble Averaging for Fault Tolerance in Memristive Neural Networks (ADAM-Lab-GW/layer-ensemble-averaging: v0.1.1). Zenodo https://doi.org/10.5281/zenodo.14550770 (2024).

Acknowledgements

The authors acknowledge Pi-Feng Chiu, formerly at Western Digital, for assistance with the initial FPGA design, and Patrick Braganca and Tom Boone, both currently at Western Digital, for valuable advice on nanofabrication and relevant applications. This work was supported in part by NIST under grant **70NANB22H018**, by Western Digital under grant **ECNS21932N**, by **NSF CAREER** under grant **2239951**, by **AFOSR YIP** under grant **FA9550-23-1-0173**, and by the GW Cross-Disciplinary Research Fund. The authors acknowledge the use of nanofabrication facilities, high-performance computing clusters and advanced IT support from the research technology services at GW and NIST. The NIST authors acknowledge support from the NIST Hardware for AI Program.

Author contributions

O.Y. and G.C.A. conceived the scientific concept. O.Y. developed the core layer ensemble averaging algorithms, and G.C.A., M.W.D., and A.D. helped refine it. O.Y. performed array characterization, neural network training and hardware inference experiments. A.M. designed the integrated circuit. B.D.H. integrated the ReRAM devices. B.D.H. and M.L.B. designed the initial Daffodil board and its API, and K.R. and O.Y. assisted with a subsequent re-design used for this work. K.R. and B.D.H. developed the device tuning algorithm, with help from W.A.B. K.R. and M.F. developed interfacing drivers for the FPGA and daughterboard. M.F., O.Y., and K.R. designed hardware description language code for the FPGA and worked on experimental testing and debugging. O.Y. developed the custom Linux-based operating system for the FPGA development board. All authors assisted in data analysis. O.Y. wrote the initial manuscript and all authors participated in co-editing. G.C.A. and J.J.M. supervised the project.

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/s41467-025-56319-6.

Correspondence and requests for materials should be addressed to Gina C. Adam.

Peer review information *Nature Communications* thanks the anonymous reviewers for their contribution to the peer review of this work. A peer review file is available.