ADVANCED INTELLIGENT SYSTEMS
Open Access
www.advintellsyst.com

QS4D: Quantization-Aware Training for Efficient Hardware Deployment of Structured State-Space Sequential Models

Sebastian Siegel, Ming-Jay Yang, Younes Bouhadjar, Maxime Fabre, Emre Neftci, and John Paul Strachan*

Structured state space models (SSM) have recently emerged as a new class of deep learning models, particularly well-suited for processing long sequences. Their constant memory footprint, in contrast to the linearly scaling memory demands of transformers, makes them attractive candidates for deployment on resource-constrained edge-computing devices. While recent works have explored the effect of quantization-aware training (QAT) on SSMs, they typically do not address its implications for specialized edge hardware, for example, analog in-memory computing (AIMC) chips. In this work, it is demonstrated that QAT can significantly reduce the complexity of SSMs by up to two orders of magnitude across various performance metrics. The relation between model size and numerical precision is analyzed, and it is shown that QAT enhances robustness to analog noise and enables structural pruning. Finally, these techniques are integrated to deploy SSMs on a memristive AIMC substrate and highlight the resulting benefits in terms of computational efficiency.

1. Introduction

Structured state space models (SSMs) have recently emerged as memory-efficient and effectively trainable neural networks for processing sequential data with long contexts.[1–4] These models feature an expanded linear recurrent state space, decomposed into parallel kernels, which are projected into a lower-dimensional space before a nonlinearity is applied.[5] The linear nature of the recurrence enables fast training via convolutional operations on GPUs. The mathematically derived initialization schemes further facilitate the learning of long sequences. A layer is formed by multiple such kernels in parallel (see **Figure 1**), hence the term "structured". While sharing some conceptual similarities with recurrent neural networks (RNNs), SSMs have been shown to outperform both traditional RNNs and even recent transformer-based models on benchmarks involving long-context sequence modeling.[6,7] This makes SSMs a promising foundation for future Large Language Models (LLMs). Their recurrent state update mechanism allows them to encode historical information using a fixed amount of memory, in contrast to transformers,[8] which require explicit storage of past tokens and incur a potentially unbounded memory cost as context length grows. Beyond LLMs, efficient sequence modeling is critical in many real-world applications, such as processing physiological signals in biomedical devices,[9–11] or enabling long-term planning and perception in autonomous vehicles.[12] These applications often impose strong constraints on the power budget or allowed latency of the underlying hardware, which demands thorough hardware-software codesign of the algorithm and the underlying hardware. This can, on the one hand, guide hardware development dedicated to SSMs and specific workloads, and on the other hand, enable the execution of computationally demanding models on given resource-constrained hardware.

A widely used strategy for deploying large machine learning models on edge hardware is quantization, which reduces memory footprint and computational load by lowering the precision of weights and activation signals. Prior work has shown that quantization-aware training (QAT) can effectively reduce the bit precision of key components in classical transformers, in some cases down to ternary values.[13] Recurrent neural networks, such as SSMs, however, are known for their high sensitivity to quantization.[14] Despite this challenge, quantization techniques have been investigated for SSMs, including S5,[15] Mamba,[16–18] and the related LMU.[19] Bi-Mamba[17] only investigates the linear layers of the Mamba model, leaving the

S. Siegel, M.-J. Yang, J. P. Strachan
Peter-Grünberg-Institute (PGI-14)
Forschungszentrum Jülich GmbH
Wilhelm-Johnen-Straße, 52428 Jülich, Germany
E-mail: s.siegel@fz-juelich.de

Y. Bouhadjar, M. Fabre, E. Neftci
Peter-Grünberg-Institute (PGI-15)
Forschungszentrum Jülich GmbH
Wilhelm-Johnen-Straße, 52428 Jülich, Germany

M. Fabre
Groningen Cognitive Systems and Materials Center (CogniGron)
University of Groningen
9700 AB Groningen, The Netherlands

E. Neftci, J. P. Strachan
Faculty of Electrical Engineering and Information Technology
RWTH Aachen University
Templergraben 55, 52062 Aachen, Germany

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202501019.

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH. This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.1002/aisy.202501019

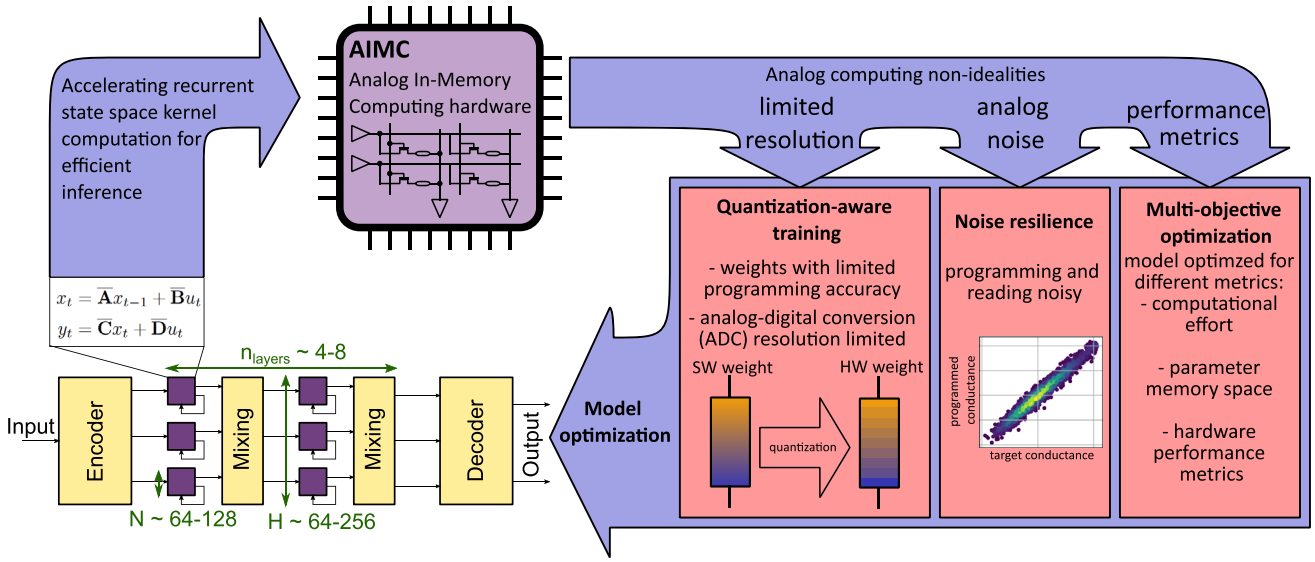

Figure 1. Hardware-software codesign for efficient deployment of state-space models. Deployment of the recurrent state update and kernel function of the SSM on AIMC hardware offers acceleration potential in terms of energy and latency. Therefore, the nonidealities of AIMC, like limited resolution and transient noise, must be incorporated in the model.

computationally heavy and algorithmically sensitive SSM block in full precision. Quamba[16] and Quamba2[18] propose advanced post-training quantization (PTQ). The benefit of PTQ over QAT is that these methods do not require training the model. However, both approaches still require one or more half-precision floating-point channels, which constitutes a hurdle for efficient integer-only hardware implementations. Except for the LMU work, these efforts primarily target implementations on general-purpose GPUs rather than specialized hardware. More recently, a number of studies have demonstrated aggressive quantization techniques for the S4D model, highlighting the importance of this class of SSM, especially for edge-hardware deployment. For instance, the work by Meyer et al.[20] demonstrates the recurrent-mode deployment of an S4D model onto the Loihi neuromorphic processor[21] and uses fine-tuning after post-training quantization.[22] investigate quantization-aware fine-tuning of small S4D models and find the recurrent kernel and the state to be the most sensitive parameters of the S4D model. Their analysis of the memory footprint already hints at the benefits of quantization on memory-constrained edge systems.[15] reach a similar conclusion for the memory footprint of the closely related S5 model. However, key aspects of SSM deployment on edge-computing hardware remain underexplored, in particular, the reduction in required hardware performance, like compute performance and memory, the system's resilience to analog noise, and the potential for hardware-software codesign, like size optimization and pruning.

In this study, we showcase the benefits of aggressive quantization-aware-training (QAT) across several metrics critical to edge-hardware devices for S4D models,[1] including memory usage and computational resources. As part of this analysis, we introduce a dedicated complexity metric tailored to analog in-memory computing (AIMC) systems, where computational resources are often dominated by the peripheral mixed-signal circuitry, particularly in architectures like memristive crossbar arrays.[23,24] We further demonstrate that model size and precision can be effectively traded off, and that, specifically in AIMC systems, QAT enhances noise resilience. Motivated by these findings, we showcase the deployment of SSM kernels on memristive crossbar arrays. Memristive crossbar arrays are well-suited for this purpose, as they enable efficient vector-matrix multiplication through analog computation. These arrays are matrix structures composed of resistive switching devices, an emerging type of nonvolatile memory.[25] Matrix elements are stored as conductances, inputs are applied as voltage vectors, and results are read out as currents. Leveraging Ohm's and Kirchhoff's laws, these arrays can compute vector-matrix multiplications in a single time step—or even continuously in time. However, due to their analog nature, they are inherently prone to noise, as illustrated in Figure 1. While we investigated the impact and mitigation of static write noise in a previous study,[26] we here focus on transient read noise. There are various memristive technologies under investigation. In our case, we are using tantalum-oxide-based resistive switching devices, cointegrated with conventional 180 nm CMOS circuits. We thereby showcase the complete hardware-software codesign process of SSMs from QAT to deployment on an AIMC substrate.

2. Results

2.1. Implementing Quantization for S4D Models

S4D models are highly heterogeneous, combining recurrent kernels, nonlinear activations, and linear feed-forward layers. In this study, we focus on the S4D model with a diagonalized **A** matrix, which has become the standard approach, with respect to the original S4 model featuring a fully populated transition matrix per kernel.[1] A study on the quantization of the original S4 model can be found in the Supporting Information, where we show that

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

the fully populated **A** matrix renders the model highly sensitive to quantization.

To investigate the quantization opportunities of the S4D model, all parameters of the model's core, i.e., the diagonal **A** matrix and **C** vectors, the vector of trainable time steps Δt, and the linear encoder and decoder layers, are quantized by projecting them into an equally spaced ladder of integer values representing the maximum range of the respective parameter, described in detail in this section. Furthermore, quantization of the activations y, which are passed between layers, is also investigated as this communication requires a significant memory bandwidth that can be optimized.

When using the convolutional approach, which allows for fast training on GPU, the state x is never explicitly computed. We thus proceed with an indirect quantization of the state. We use the convolutional kernel that is materialized for training, and the input to simulate a quantization of the state. Due to the convolution operation, the maximum resolution of the state is the sum of the precision of the kernel and the input. Therefore, if the state is quantized to k levels, the quantization of the input and the convolutional kernel are set to $k/2$. However, it should be noted that this does not yield the exact same numerical results as the recurrent computation. Zhao et al.[22] study the direct quantization in the recurrent mode.

2.2. QAT of SSMs Allows Aggressive Quantization

The plots in **Figure 2** show the impact of quantization of all key parameters of the model for the sequential gray-scale CIFAR10 benchmark, evaluating the model's capacity to remember long sequences. Starting from the right side with 16-bit parameter accuracy, a more aggressive quantization is applied, going to the left side of the plot. The results for post-training quantization (PTQ) are given in blue (dashed line) while QAT is shown in orange (solid line). The black dashed line shows the accuracy of a model with 32-bit floating-point accurate parameters and activations. Green areas indicate the range of quantization values where PTQ degrades the performance by more than 1% while QAT achieves higher accuracy.

When quantizing all parameters homogeneously (Figure 2a), it can be seen that PTQ leads to an increased error starting from 12-bit accuracy to lower resolutions. At 10 bits, the additional error reaches 1%, which is used as a level for comparison here. With QAT, in contrast, a homogeneous quantization of 7 bits can be reached before crossing this threshold.

In fact, we observe this benefit of QAT over PTQ for all parameters. It is especially striking for parameters partaking in the recurrent state update, like the transition matrix **A** (Figure 2b), the state (Figure 2d), and the time-step Δt (Figure 2h).

2.2.1. Pathfinder Dataset

The impact of quantization on the model performance depends on the task. For comparison, we investigate the Pathfinder task, another benchmark task from the Long Range Arena suite[27] like sCIFAR, which also requires memorizing a long sequence. **Table 1** lists the maximum quantizations achievable to stay below 1% of additional error for both PTQ and QAT. On the Pathfinder dataset, QAT allows for a 5-bit homogeneous quantization of the whole model, while PTQ fails below 16 bits. This is a gain of 11 bits in parameter size. The largest gains for the individual parameters can be found for the **A** matrix and the Δt parameter.

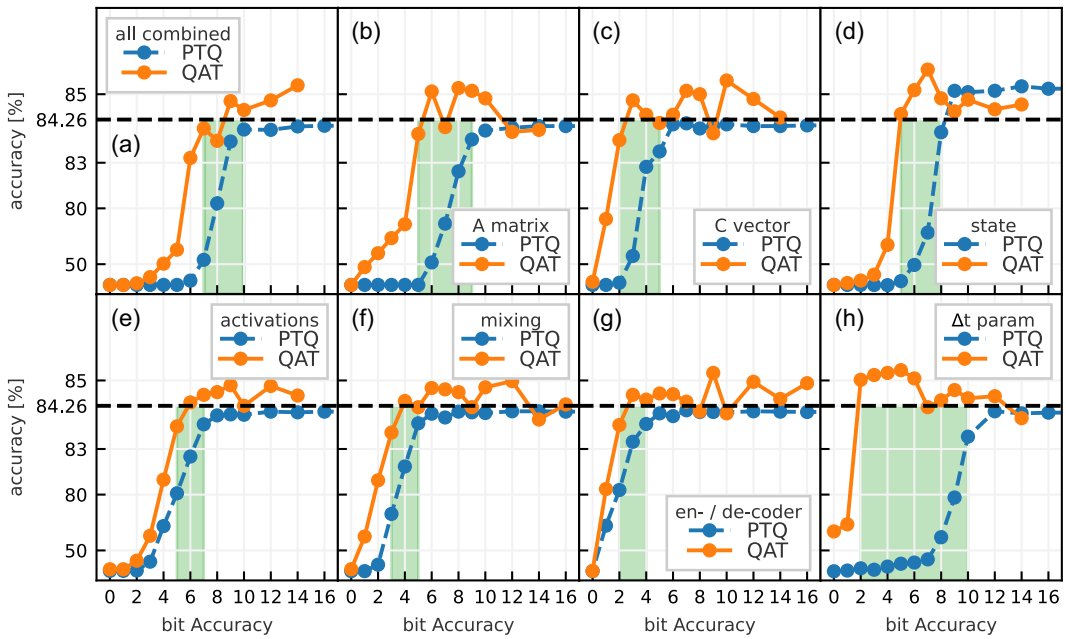

Figure 2. QAT of sequential CIFAR10 (grayscale). a) For homogeneous quantization, QAT allows a reduction to 7 bits for all model parameters before the error increases above 1% over the baseline. For PTQ, this already happens at 10-bit accuracy. b–h) For the quantization of only individual parameters, a varying benefit of QAT over PTQ can be observed. The gain in quantization is especially high for the recurrent kernel parameters (b), (d), and (h).

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Table 1. Maximum quantization to stay below 1% of additional error with PTQ and QAT for the sequential CIFAR10, pathfinder, and Heidelberg digits (HD) tasks, in bit. "all" denotes the values for homogeneous quantization of all parameters simultaneously. For the other columns, only the indicated parameter is quantized.

	Baseline		All	A	B	C	State	Activations	Mixing	En-/decoder	Δt
sCIFAR10	84.3%	PTQ	10	9	NA	5	8	7	5	4	10
		QAT	7	5	NA	2	5	5	3	2	2
		gain	3	4	NA	3	3	2	2	2	8
Pathfinder	97.6%	PTQ	16	14	2	2	7	5	2	2	16
		QAT	5	3	2	2	5	4	2	2	1
		gain	11	11	0	0	2	1	0	0	15
HD	100.0%	PTQ	14	6	NA	6	14	5	4	2	6
		QAT	5	2	NA	2	4	2	1	1	1
		gain	9	4	NA	4	10	3	3	1	5

Also, for this challenging benchmark task, QAT leads to significant gains over PTQ, especially in the recurrence matrix A, which also has a large impact on the computational complexity.

2.2.2. Keyword Spotting (Heidelberg Digits (HD))

A possible application for state space models in edge computing systems is the classification of temporal data, like audio streams, for example, in a keyword spotting application. This task is significantly different from the synthetic ones in the Long Range Arena suite that focus mostly on the system's capability for memorization. We thus investigate the quantization behavior for the classification of the Heidelberg Raw Audio Digits.[28] The dataset consists of recordings of English and German spoken digits from zero to nine of different speakers. In this work, we limit the dataset to the English subset.

Like for the benchmark tasks, for the HD dataset, a large gain of 9 bits can be achieved with QAT over PTQ. However, when discriminating by parameter, the major gains can be found for the recurrence matrix A, the state, and the activations. For CIFAR10 and Pathfinder, only minor gains can be reached for the latter parameter.

2.3. Hardware Efficiency Gains Through Aggressive Quantization

The effect of quantization on the efficiency of hardware implementations can be quantified by different metrics, depending on the type of hardware substrate. For conventional digital processing units, such as GPUs, the number of Multiply & Accumulate (MAC) operations is important. For memory or memory bandwidth-constrained systems like microcontrollers, the model size in terms of storage for the model parameters can become an important factor. To evaluate the computational complexity, we employ the ACE metric[29] as outlined in Section 5. For the memory footprint, we accumulate the bit values for all parameters (see Section 5). Finally, for AIMC solutions, the peripheral circuitry for the analog-to-digital conversion (ADC) after the analog vector-matrix-multiplication is often crucial in terms of chip area, latency, and energy consumption. We propose to estimate this complexity by accumulating the precision bits of state and activation read-outs. For details, we direct the reader to Section 5.

Using the quantization levels obtained above, significant reductions in all of these metrics can be obtained. **Figure 3** shows these results for reference floating-point (fp) models, and models with maximum quantization below 1% additional error for the PTQ and QAT methods. We cover sequential CIFAR10, Pathfinder, and the HD audio dataset. By PTQ, the computational effort decreases already by a factor of 11.3 to 23.8 (Figure 3a), which is further surpassed by the QAT models by additional factors between 2 and 11.5. Similarly, the model's size in (Figure 3b) benefits by at least a factor of 4 from PTQ and is reduced further down by a factor of up to 2.9 with QAT. In the same manner, the peripheral ADC complexity (Figure 3c) is decreased by a factor of 2.7 to 4.6 by PTQ alone and by another factor of 1.4 to 4. For all metrics, the largest benefit of QAT over PTQ can be found for the HD audio classification dataset, which is one of the most likely application areas for an SSM model deployed on edge hardware.

2.4. Trading Model Size for Quantization

Model dimensions, especially its width, are another central element in addition to quantization to benefit a model's efficiency-accuracy trade-off for edge implementations. In the following, we investigate the impact of the model size in relation to quantization, to establish an optimum of accuracy and efficiency. An educated choice of model size and resolution can significantly impact performance for the deployment of the model to a given hardware substrate with memory or computational constraints.

Three parameters determine the size of the S4D model: the state dimension **N**, the model width **H**, and the number of layers. The latter is very specific for each task and is therefore fixed in this study. **Figure 4**a,b depicts the task performance for the sequential CIFAR10 task for different model sizes and full-model quantizations. Missing data points occur if the training fails with the given parameters.

In Figure 4a, we vary the state dimension on the *x*-axis. It can be seen that for most investigated homogeneous quantization levels, the model performance increases with increasing state

 © 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

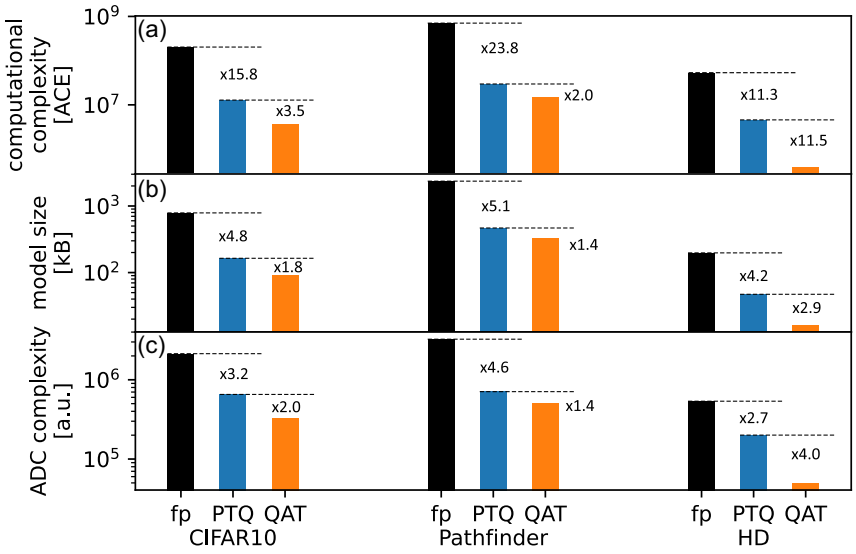

Figure 3. Quantization benefits in different hardware metrics. The graphs show the respective metrics for models with maximum quantization. a) The ACE for all investigated tasks decreases by at least a factor of 11 by quantizing the model after training. QAT allows for an additional decrease by a factor of 2 to 11.5. b) The memory required to store all model parameters decreases with PTQ by at least a factor of 4.2 and an additional 1.4 to 2.9 with QAT. c) The analog–digital conversion metric estimates the peripheral complexity in an AIMC hardware implementation. Also here benefits due to PTQ are surpassed by QAT.

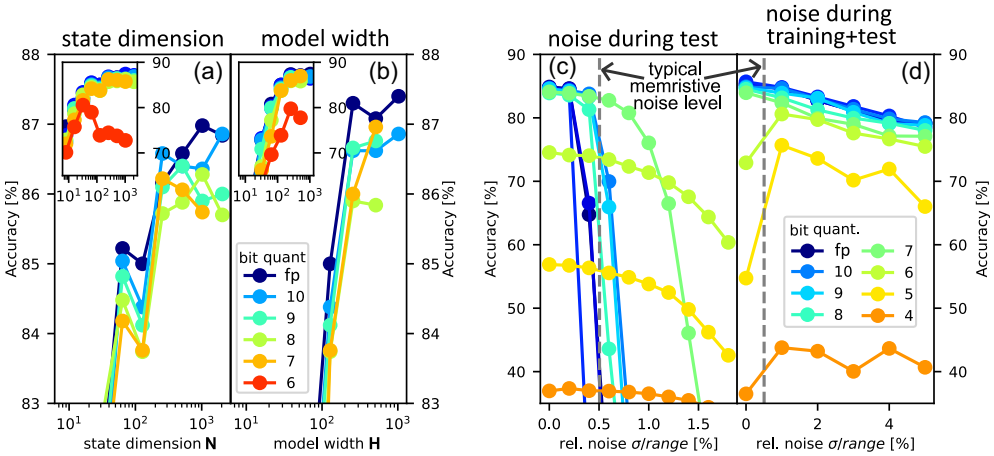

Figure 4. Relation of model size and quantization: a) below a state size of $\mathbf{N} = 256$, an increase of the state dimension increases the model accuracy and can compensate for the loss due to quantization. For a higher state dimension, the accuracy plateaus. b) A similar behavior can be observed for the model width. Noise resilience: c) increasing transient noise in the A matrix causes a degradation of the performance. Aggressive quantization can slow down this degradation, but can also reduce the performance in general. d) Introducing transient noise during training makes all models more resilient against noise and can even increase the accuracy over training without noise. All trainings are performed on the sequential gray-scale CIFAR10 benchmark task.

dimension up to a maximum, which is dependent on the quantization. For low resolution, however, the performance decreases for larger state dimensions, as can be seen for 6 bits in the inset.

For the variation of the model width in Figure 4b, we observe similar behavior. The performance increases with the width until a saturation is reached at $\mathbf{H} = 128$, at an accuracy value highly correlated with the quantization level.

Another way to reduce the model size is by pruning. Due to their highly structured shape, S4D models can significantly benefit from structural pruning if complete kernels can be eliminated. This applies especially for an in-memory hardware implementation, because kernels and states need to be materialized in hardware. When investigating the prunability of a model with homogeneous quantizations on the example of the CIFAR10 benchmark task, for each layer, a different fraction of complete kernels can be pruned (see **Figure 5**). For the model with 7-bit homogeneous quantization, in the first layer, more than half of the kernels can be eliminated without losing more than 1% task performance, while in the third layer, only about 5.49% kernels can be pruned. When comparing models with

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

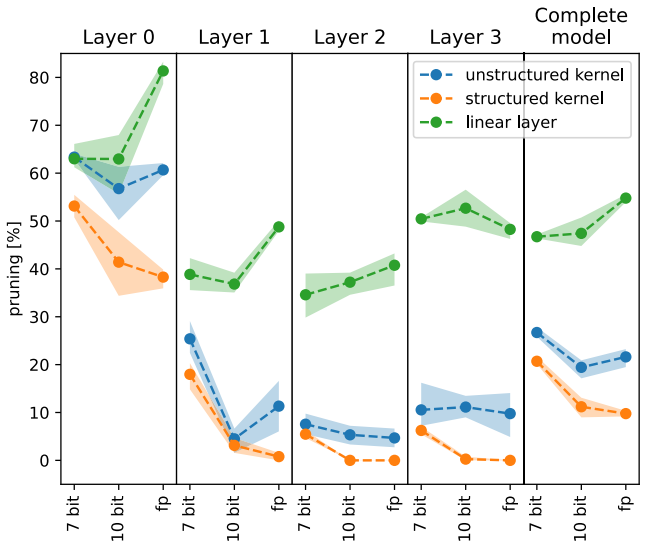

Figure 5. Pruning of recurrent kernels and linear layers for models trained for different quantization. Structured kernel pruning means the elimination of entire kernels, while unstructured pruning refers to single values in the kernel parameters. Highly quantized models allow for more structured kernel pruning, while linear layers can be more heavily pruned in highly precise networks.

different homogeneous quantization (here 7 bit, 10 bit, and floating-point), a more aggressive quantization allows for more aggressive structural pruning across all layers.

The differences in pruning potential arise especially for the structural pruning of complete kernels. While there are also small benefits of highly quantized models for unstructured kernel pruning, for the unstructured pruning of the linear layers, no significant difference can be observed. Furthermore, pruning of the linear layers only has minor impact on the analog CIM implementation because it is unstructured. A pruned weight here results in a zero weight that nevertheless must be realized in a crossbar array.

For details on the pruning process and results, we refer to the Supporting Information.

2.5. Quantization for Noise Robustness

A deployment on analog in-memory compute substrates thus necessitates a study of such noise on the task performance. Therefore, we subject the kernel parameters of models trained with QAT for different quantizations with Gaussian transient noise during inference, after training without. We report the added Gaussian noise as the standard deviation relative to the parameter range. In terms of hardware implementation, this noise is simulating the relative conductance noise of memristive devices, meaning their standard conductance deviation divided by their full conductance range. In Figure 4c, the model performance for an increasing relative parameter noise is depicted for different quantization levels for the CIFAR10 task. It can be seen that the model with full precision (fp) and models trained with high accuracy parameters (10-bit to 8-bit) lose performance quickly while increasing noise. This performance drop occurs around a relative noise of around 0.5%, which is also around the typical relative noise in memristive crossbar arrays. Such noise levels can, for example, be found in the memristive crossbar accelerator used later in this work (see Figure 5, Supporting Information). Even if significantly lower noise levels can be reached using advanced programming techniques, as in a work by Rao et al.,[30] this noise sensitivity can be an issue for efficient SSM implementation on memristive accelerator systems. For models trained with more aggressive quantization, the performance drops more slowly with increasing noise. The highest quantized models, however, exhibit a lower performance from the start but can surpass less quantized models for high noise levels.

We suggest that this acquired noise robustness stems from the fact that quantization itself is a form of noise. QAT thus constitutes a form of training with noise. To prove this, we explicitly introduce Gaussian noise to the recurrent kernel parameters during training. The results can be seen in Figure 4d (the large range of the x-axis should be noted when comparing to c). All models, especially those with high precision parameters, show a slight decrease in task performance with increasing noise levels when they are trained with the respective noise. For highly quantized models (6-bit, 5-bit, and 4-bit), adding noise during training even increases their performance and restores some of the losses due to the quantization.

In conclusion, aggressive quantization through QAT can make a model more resilient to parameter noise. For a known level of noise, however, training with this noise still yields superior results.

3. Implementation of S4D Kernel on Memristive Crossbar Arrays

To enable efficient execution on GPUs, the S4D model's kernel is unrolled into a convolutional kernel, and the input is processed in batches. This allows the GPU to leverage highly parallel matrix-matrix multiplication. However, implementing this approach on dedicated hardware requires storing the unrolled convolutional matrix and input data across multiple time steps, which is impractical for resource-constrained edge systems due to memory limitations. A more viable alternative for such systems is a streaming implementation, where the input signal is processed sequentially, one time step at a time. This approach necessitates implementing the original state-space formulation, as given in Equation (3), directly in hardware. It relies on fast and efficient vector-matrix multiplication, while minimizing memory usage to the storage of latent states only.

Memristive crossbar arrays (mCBAs) have been proposed in the literature as a means to enable time- and energy-efficient vector-matrix multiplication within a single time step. In this work, due to system constraints, we implement the time-discrete version of the S4D model. However, in principle, mCBAs can also support time-continuous operation and thus could be leveraged to implement the continuous-time formulation of the model.

To minimize data movement between compute blocks, as much computation as possible should be performed within the same crossbar array. To this end, we proposed the in-memory state space accelerator (IMSSA) architecture,[26] which maps the

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

S4D kernel computation such that it performs not only the $\overline{\mathbf{A}}x$ operation, but also the $\mathbf{B}u$ and $\mathbf{C}x$. The latter two are vector-scalar operations and thus require only a single row or column within the crossbar array, as illustrated in **Figure 6**a. While this reduces the space available for the **A** matrix, it enables the entire S4D kernel to be computed within a single mCBA. The state is updated by accumulating the resulting current vector at each time step. As a consequence, the **C** vector is not multiplied directly with the current state x_t, but rather with a one-step delayed version. The IMSSA kernel computation is therefore given by

$$\begin{aligned} x_t &= \overline{\mathbf{A}}x_{t-1} + \overline{\mathbf{B}}u_t \\ y_{t-1} &= \overline{\mathbf{C}}x_{t-1} + \overline{\mathbf{D}}u_t \end{aligned} \tag{1}$$

which is a small deviation from the original kernel in Equation (3).

The proposed mCBA system comprises three memristive crossbar arrays, each configured as a 64×64 matrix of memristive devices. Further details on the system architecture are provided in the Supporting Information. A performance comparison with a commercial edge GPU (NVIDIA Jetson Nano) is presented in **Table 2**. Representing a single complex matrix element requires a 4×4 submatrix of memristive devices, as detailed in the Experimental Section, Section 5. Consequently, each mCBA can represent a 16×16 matrix of complex values. One row and one column are reserved for the **B** and **C** vectors, respectively. We allocate a 14×14 matrix for **A** and assume a device programming precision of 3 bits per cell. Using these constraints, we train an S4D model—configured according to the parameters listed in **Table 3**—on a two-class subset of the HD audio recognition dataset, achieving a test accuracy of 95.8%. The resulting kernels (Figure 6b) are programmed on the mCBA system and read out to achieve realistic write and read noise distributions. With these programmed kernels, the system achieves classification accuracies ranging from 83.69% to 86.41%. The observed drop in accuracy is attributed to programming multiple parameters with differing dynamic ranges onto the same memristive substrate, which reduces their effective quantization resolution. With the method of a common constant maximum value for all kernel parameters outlined in Ref. [26] we achieve an accuracy of 95.3%, almost achieving software task performance.

Table 2. Comparison with digital CMOS AI inference accelerators.

Benchmark metric	This work	Jetson Nano
Technology	180 nm	20 nm
Area (mm^2)	136.85	118.0
Bit Hardware precision	up to INT4	float 16
Throughput (GOPS)	122.88	472
Power (W)	16.47e-3	10
Energy efficiency (TOPS/W)	7.46	47.2e-3
Area efficiency (TOPS/mm^2)	0.9e-3	4

Table 3. Parameters used for small (sub-) dataset of Keyword spotting (HD).

Parameter	n_{layers}	N	H	r_{state}/r_{act}	$r_{kernelweights}$	SW accuracy	mcba accuracy
Value	1	14	3	8 bit	4 bit	95.8%	95.3%

4. Conclusion

In this work, we discuss the process of deploying a structured state-space sequential model (specifically, the S4D model) to edge-computing accelerators. The goal of this process is to accelerate the recurrent state update during inference, which is inefficient on classical GPU compute substrates.

In a first step, we demonstrate how QAT can successfully be applied to S4D models. For various classes of sequence learning tasks, QAT allows for a more aggressive quantization than PTQ,

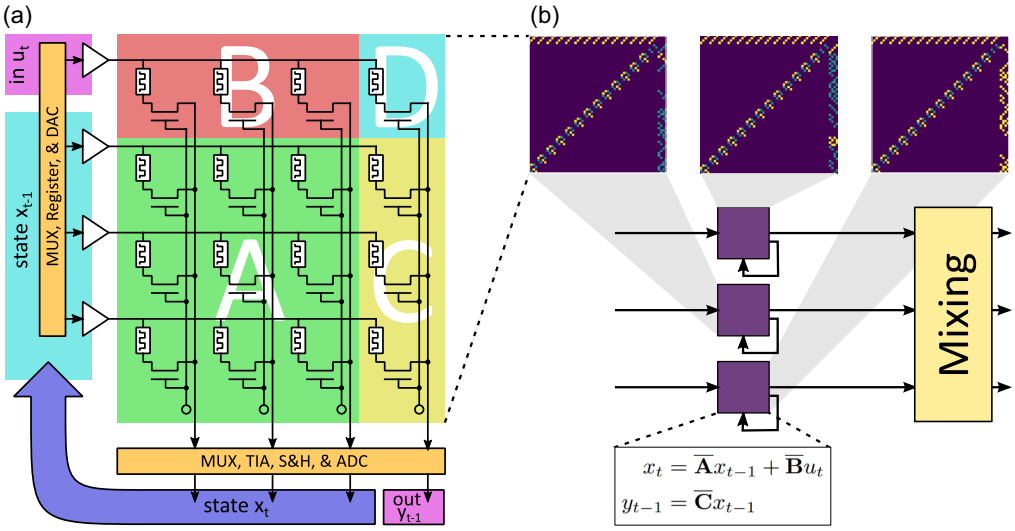

Figure 6. IMSSA kernel implementation on an mCBA.

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

especially for the recurrent kernel parameters. This is an important step, because edge-computing substrates often offer only compute elements with limited precision. We find the largest benefits of QAT over PTQ for the recurrent update matrix A and the state, which is confirmed by the parallel study of Zhao et al.[22] for small models and tasks. We here show the same trend for larger models and a wide range of applications, from perfect memory tasks like the Pathfinder challenge to audio classification of spoken digits. In future works, it could be interesting to investigate the applicability of advanced PTQ techniques, like in works by Chiang et al.[16,18]

Depending on the task, aggressive quantization with QAT reduces the computational complexity by multiple orders of magnitude, the memory required for the model parameters by approximately a factor of 10, and also the complexity for analog-digital conversion in an AIMC substrate significantly. Furthermore, QAT allows for aggressive structural pruning after training, further enhancing these gains. These findings are promising for the deployment of S4D models on edge-computing hardware, which is often constrained by computational power, memory, or peripheral complexity in the case of AIMC.

Edge-computing substrates often come with restrictions on the size of matrix operations they can support efficiently. We demonstrate how the size of the recurrent kernels and the number of kernels per layer can be traded against the level of quantization that can be applied. A similar scaling law was observed for large language models (LLMs).[31] This opens up the design space for hardware-software co-design for tailored accelerator systems and makes it possible to fit models to the specific properties of a given edge-computing system.

Especially, AIMC systems suffer from inherent noise when reading and writing parameters. After investigating the read noise in a previous study, we showcase here how QAT makes models more resilient against transient read noise. More aggressive quantization leads to more noise resilience, but there is a trade-off with the generally lower task performance at very high quantizations. For known noise levels, we demonstrate that training with noise can lead to even more resilient models and even increase the task performance of highly quantized models.

Finally, we demonstrate the successful deployment of an S4D model for audio classification on a memristive AIMC substrate using the techniques showcased in this paper and the deployment methods described in Ref. [26]. For that, we use a mapping of the recurrent kernel to a memristive crossbar array that combines all kernel functions in one array and thus allows to perform state update and output to be computed in a single time step.

In this study, we showcase the benefits of QAT of SSMs for deployment on edge-computing substrates by reducing computational complexity and memory footprint. Specifically for AIMC, we demonstrate how the two key factors, peripheral circuit complexity and noise resilience, benefit from QAT and conclude the study by showcasing the deployment of S4D kernels on memristive crossbar arrays. Thereby, we pave the way for efficient sequence processing with ultra-long contexts at the edge.

5. Experimental Section

Structured State Space Models: As proposed in the work by Ref. [1], each layer of the S4 state space model consists of a transition matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ describing the evolution of the state $x(t) \in \mathbb{R}^N$ over time. The input influences the state $x(t)$ via the matrix $\mathbf{B} \in \mathbb{R}^{N\times 1}$, which additionally projects the input signal into the dimension of $x(t)$. The output $y(t)$ of a layer is a combination of a projection of the state $x(t)$ via the matrix $\mathbf{C} \in \mathbf{R}^N$ and the input signal $u(t)$ via the matrix $\mathbf{D}$. In this work, as is usually the case, $\mathbf{D}$ is set to zero.

$$\begin{aligned} \frac{dx(t)}{dt} &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t) \end{aligned} \tag{2}$$

$\mathbf{H}$ blocks of this structure are stacked to form an S4 layer, which is repeated n_{layer} times to form the model. Initially, the input signal to the network is projected to the dimension $\mathbf{H}$ of the model by a linear encoder. The output of the model is again projected via a linear transformation to the output dimension. In this work, classification tasks are investigated. Therefore, the output dimension is the number of classes.

For the execution on time-discrete compute systems like CPUs and GPUs, the time-continuous kernel of the S4 model needs to be discretized in time. In this work, a Zero-Order-Hold (ZOH) discretization is chosen as proposed in Ref. [1]. Given a time step Δt, this yields the kernel formulation

$$\begin{aligned} x_t &= \overline{\mathbf{A}}x_{t-1} + \overline{\mathbf{B}}u_t \\ y_t &= \overline{\mathbf{C}}x_t + \overline{\mathbf{D}}u_t \end{aligned} \tag{3}$$

The time step Δt is individual for each S4 block and becomes a trainable parameter. This yields an additional parameter vector $\Delta t \in \mathbb{R}^N$ per layer.

To reduce the computational complexity of the state update, the $\mathbf{A}$ matrix is diagonalized. Since this is by default not possible, Gu et al.[6] propose splitting $\mathbf{A}$ into a normal (and thus diagonalizable) part and a low-rank residual. Dropping this residual has only a minor impact on the task performance, but reduces the computational burden to a linearly scaling element-wise multiplication. These models are called S4D models and are used for the remainder of this analysis.

The model given in 3 is a recurrent model. This would mean that the state x is read and recomputed for each time step of a sequence and stored in memory after each step. This would lead to a high IO bandwidth and render the model inefficient on modern GPU systems. Therefore, Gu et al.[1] propose to take advantage of the linearity of 3 and unroll the recurrence into a convolutional kernel to execute multiple steps of 3 in a single matrix–matrix multiplication, like

$$\begin{aligned} \mathbf{K} &= (\overline{\mathbf{C}}\,\overline{\mathbf{B}}, \overline{\mathbf{C}}\,\overline{\mathbf{A}}\,\overline{\mathbf{B}}, \overline{\mathbf{C}}\overline{\mathbf{A}}^k\overline{\mathbf{B}}, \ldots, \overline{\mathbf{C}}\overline{\mathbf{A}}^L\overline{\mathbf{B}}) \\ y &= \mathbf{K} * u \end{aligned} \tag{4}$$

for an L time-step long sample. For practical implementation, this convolution is additionally transformed to the frequency space by fast-Fourier-transformation, and the convolution becomes a multiplication.

Gu et al. found that replacing $\mathbf{B}$ with an identity and not training it in many cases only leads to a small drop in performance.[6] Since this would also increase the computational efficiency of the model, we apply this approach whenever possible. For this study, this means that the analysis on the CIFAR10 and the Heidelberg Digits data set employs a $\mathbf{B} = 1$. For the training of the pathfinder data set, however, a trainable $\mathbf{B}$ matrix is necessary, as shown by previous studies.

Quantization: Quantization Methods: Quantization describes the resolution of weights and activations in the model, which often involves changing from floating-point precision to fixed-point precision, also called integer precision. However, the operations reported in this work are executed on GPUs, and for the sake of comparability, all are performed with 32-bit floating-point values for all parameters. Therefore, to emulate a lower resolution, parameters are subjected to artificial quantization by rounding each value within a weight layer or activation vector to the next value in an equally spaced grid, with the target quantization reflected in the number of grid cells. The range of the grid can either be determined by the maximum and minimum values or the average value in the respective layer. In this work, the grid spans around zero with a range of the maximum absolute

 © 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

value. Other works, such as Ref. [13], also use the average of absolutes as a range. The formula for quantization can be expressed as

$$f_{\text{scale}} = \max(\text{abs}(x))$$
$$x_{i,\text{quant}} = \text{round}\left(x_i * \frac{n_{\text{levels}}}{f_{\text{scale}}}\right) * \frac{f_{\text{scale}}}{n_{\text{levels}}} \tag{5}$$

Quantization: Post-Training Quantization (PTQ): For the post-training quantization, the model is trained with all parameters and activations being 32-bit floating-point values. After the training, parameters are subjected to the quantization formula 5. The same method can also be used to quantize the activations. In this specific case, this means that the signals propagated between layers (y_t in 3) are quantized.

Quantization: QAT: In contrast to PTQ, QAT includes quantization already in the training cycle. As described in Ref. [13], this is done in a way that the quantization method is applied during the forward pass, but not during the backward pass, also called straight-through estimator. This is achieved by detaching the quantization from the computational graph for the autograd function.

Hardware Metrics: Computational Complexity: To quantify the computational complexity, we employ the arithmetic computational effort (ACE) metric,[29] which computes the computational effort in relation to single-bit MAC operations. For the kernel and linear layer operations of the S4D model, the ACE complexity can be computed as follows

$$\begin{aligned}
\text{ACE}_{Ax} &= c * \mathbf{N} * r_A * r_{\text{act}} * \mathbf{H} * n_{\text{layer}} \\
\text{ACE}_{Bu} &= c * \mathbf{N} * r_B * r_{\text{act}} * \mathbf{H} * n_{\text{layer}} \\
\text{ACE}_{Cx} &= c * \mathbf{N} * r_C * r_{\text{act}} * \mathbf{H} * n_{\text{layer}} \\
\text{ACE}_{\text{linear}} &= \mathbf{H} * \mathbf{H} * r_{\text{act}} * r_{\text{linear}} * n_{\text{layer}} \\
\text{ACE}_{\text{en-,decoder}} &= \mathbf{H} * (n_{\text{in}} + n_{\text{out}}) * r_{\text{act}} * r_{\text{coder}}
\end{aligned} \tag{6}$$

where r_A is the bit resolution of the A matrix, r_B and r_C are the bit resolution of the B and C vectors, r_{act} is the bit resolution of the activation and the state, and n_{layer} is the number of layers. c is a constant factor that is 1 for real values and 4 for complex values. r_{linear} is the bit resolution of the linear mixing layers, while the resolution of the encoder and decoder layers is given by r_{coder}, n_{in} and n_{out} are the number of input and output channels of the model, respectively. The sum of all contributions is the total ACE.

Hardware Metrics: Model Size: The model size is the sum of bits that its parameters occupy in memory. This comprises the kernel parameters, the linear mixing, encoder, and decoder layers. It can be calculated as

$$\begin{aligned}
\text{Mem} = {} & c * \mathbf{N} * r_A * \mathbf{H} * n_{\text{layer}} \\
& + c * \mathbf{N} * r_B * \mathbf{H} * n_{\text{layer}} \\
& + \mathbf{N} * r_C * \mathbf{H} * n_{\text{layer}} \\
& + \mathbf{H} * \mathbf{H} * r_{\text{linear}} * n_{\text{layer}} \\
& + \mathbf{H} * (n_{\text{in}} + n_{\text{out}}) * r_{\text{coder}}
\end{aligned} \tag{7}$$

where here c is 1 for real-valued parameters and 2 for complex.

Hardware Metrics: ADC Complexity: As a measure for the complexity of the analog-to-digital conversion circuitry necessary for an AIMC implementation, we propose the number of bits that need to be converted. This can be calculated as

$$\begin{aligned}
\text{ADC}_{\text{kernel}} &= (c * \mathbf{N} * r_{\text{state}} + r_{\text{act}}) * \mathbf{H} * n_{\text{layers}} \\
\text{ADC}_{\text{mixing}} &= r_{\text{act}} * \mathbf{H} * n_{\text{layer}} \\
\text{ADC}_{\text{coder}} &= r_{\text{act}} * (\mathbf{H} + n_{\text{out}})
\end{aligned} \tag{8}$$

Memristive Crossbar Hardware Implementation: In-Memory S4D Kernels: Memristive devices are an emerging class of nonvolatile memory technology. Their electrical conductance can be modulated by applying electrical stimuli, often in a gradual and controllable manner. Various physical mechanisms underlie this behavior. In this work, we employ transition metal oxide-based memristive devices[32] that utilize the valence change mechanism (VCM) to alter their conductance. Specifically, the devices consist of a tantalum oxide (TaO_x)/tantalum switching layer, sandwiched between two noble metal electrodes. Each memristive device is integrated in series with a conventional CMOS transistor, forming a 1-transistor-1-resistor (1T1R) configuration. These 1T1R cells are arranged in a matrix-like structure known as a memristive crossbar array (mCBA). Mathematical matrices can be encoded in the mCBA as conductance values g. When an input vector is applied as a voltage vector ν, the resulting output current vector i reflects the outcome of a vector-matrix multiplication. This operation is inherently parallel and can be executed in a single computational step, and even supports time-continuous processing under appropriate conditions.

The conductances of memristive devices are inherently positive and real-valued. To represent complex-valued matrices—such as the **A** matrix of an S4D model, which may also contain negative elements—each matrix element must be decomposed and encoded using multiple memristive conductances. A signed real value is typically implemented using two conductances, enabling differential encoding. Furthermore, due to technical constraints, memristive arrays often support only positive voltage inputs. As a result, the computation of a single matrix element—whether real or complex—must be expanded accordingly to account for these hardware limitations

$$\begin{pmatrix} i^+ \\ i^- \end{pmatrix} = \begin{pmatrix} g^+ & g^- \\ g^- & g^+ \end{pmatrix} * \begin{pmatrix} \nu^+ \\ \nu^- \end{pmatrix} \tag{9}$$

where all components i, g, and ν are positive. $g^+ - g^- = m$ represents each matrix element m if the input voltage $\nu = \nu^+ - \nu^-$. The result can be computed from the output currents like $i = i^+ - i^-$.

Diagonalization of the **A** matrix in the S4D kernel may result in complex-valued entries. To represent these using the real-valued conductances of the mCBA, each complex number is mapped to a 2D real-valued vector. When incorporating the representation of signed values as described in Equation (9), the multiplication of complex-valued inputs and matrix elements expands accordingly to

$$\underline{i} * \underline{m} = \begin{pmatrix} m_r & -m_i \\ m_i & m_r \end{pmatrix} * \begin{pmatrix} i_r \\ i_i \end{pmatrix} = \begin{pmatrix} g_r^+ & g_r^- & g_i^- & g_i^+ \\ g_r^- & g_r^+ & g_i^+ & g_i^- \\ g_i^+ & g_i^- & g_r^+ & g_r^- \\ g_i^- & g_i^+ & g_r^- & g_r^+ \end{pmatrix} * \begin{pmatrix} \nu_r^+ \\ \nu_r^- \\ \nu_i^+ \\ \nu_i^- \end{pmatrix} = \begin{pmatrix} i_r^+ \\ i_r^- \\ i_i^+ \\ i_i^- \end{pmatrix} \tag{10}$$

Each diagonal matrix element is thus represented by a 4×4 block, resulting in a block-diagonal matrix structure. Consequently, the dimensions of the mCBA must be at least four times the state dimension **N**.

Memristive Crossbar Hardware Implementation: Programming Quantized Values on Memristive Crossbar: To accurately map RNN kernel matrices onto memristive conductance levels and efficiently adapt to new weight values, we employ an advanced tuning technique[33] for memristive devices. This approach leverages a compact 1T1R device model specifically tailored for memristive crossbar arrays.[34] The model predicts optimal SET/RESET pulse parameters by considering both the deviation from the target conductance and device variability. Compared to the widely used write-and-verify approach, such as the incremental step pulse method, this technique achieves a 40%–100% reduction in programming time.[34,35]

Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

Acknowledgements

This work has been supported by NeuroSys as part of the initiative "Clusters4Future" by the Federal Ministry of Research, Technology and

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Space BMFTR (03ZU2106CB) and as part of the NEUROTEC-II project (grant no. 16ME0398K and 16ME0399).

Open Access funding enabled and organized by Projekt DEAL.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Keywords

edge AI, hardware-software co-design, memristive devices, quantization, quantization-aware training, state-space models

Received: August 29, 2025
Revised: December 18, 2025
Published online:

[1] A. Gu, K. Goel, C. Ré, arXiv preprint arXiv:2111.00396 **2021**.
[2] J. T. Smith, A. Warrington, S. W. Linderman, arXiv preprint arXiv:2208.04933 **2022**.
[3] A. Voelker, I. Kajić, C. Eliasmith, *Adv. Neural Inf. Process. Syst.* **2019**, *32*.
[4] A. Gu, T. Dao, arXiv preprint arXiv:2312.00752 **2023**.
[5] A. Gu, T. Dao, S. Ermon, A. Rudra, C. Ré, *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1474.
[6] A. Gu, K. Goel, A. Gupta, C. Ré, *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 35971.
[7] T. Dao, A. Gu, arXiv preprint arXiv:2405.21060 **2024**.
[8] A. Vaswani, *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
[9] B. Ballinger, J. Hsieh, A. Singh, N. Sohoni, J. Wang, G. H. Tison, G. M. Marcus, J. M. Sanchez, C. Maguire, J. E. Olgin, M. J. Pletcher, In *Thirty-Second AAAI Conf. on Artificial Intelligence*, AAAI Press, New Orleans, Louisiana, USA **2018**.
[10] B. Ay, O. Yildirim, M. Talo, U. B. Baloglu, G. Aydin, S. D. Puthankattil, U. R. Acharya, *J. Med. Syst.* **2019**, *43*, 1.
[11] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, J. Dean, *Nat. Med.* **2019**, *25*, 24.
[12] A. Pandharipande, C.-H. Cheng, J. Dauwels, S. Z. Gurbuz, J. Ibanez-Guzman, G. Li, A. Piazzoni, P. Wang, A. Santra, *IEEE Sens. J.* **2023**, *23*, 11097.
[13] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, F. Wei, arXiv preprint arXiv:2402.17764 **2024**.
[14] J. Ott, Z. Lin, Y. Zhang, S.-C. Liu, Y. Bengio, arXiv preprint arXiv:1608.06902 **2016**.
[15] S. Abreu, J. E. Pedersen, K. M. Heckel, A. Pierro, arXiv preprint arXiv:2406.09477 **2024**.
[16] H.-Y. Chiang, C.-C. Chang, N. Frumkin, K.-C. Wu, D. Marculescu, In *The Thirteenth Inter. Conf. on Learning Representations (ICLR)*, PMLR **2025**.
[17] S. Tang, L. Ma, H. Li, M. Sun, Z. Shen, arXiv preprint arXiv:2411.11843 **2024**.
[18] H.-Y. Chiang, C.-C. Chang, N. Frumkin, K.-C. Wu, M. S. Abdelfattah, D. Marculescu, In *Forty-Second Inter. Conf. on Machine Learning (ICML)*, OpenReview.net **2025**.
[19] P. Blouw, G. Malik, B. Morcos, A. R. Voelker, C. Eliasmith, arXiv preprint arXiv:2009.04465 **2020**.
[20] S. M. Meyer, P. Weidel, P. Plank, L. Campos-Macias, S. B. Shrestha, P. Stratmann, M. Richter, arXiv preprint arXiv:2409.15022 **2024**.
[21] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al., *IEEE Micro.* **2018**, *38*, 82.
[22] L. Zhao, T. Torchet, M. Payvand, L. Kriener, F. Moro, arXiv preprint arXiv:2506.12480 **2025**.
[23] Q. Xia, J. J. Yang, *Nat.Mater.* **2019**, *18*, 309.
[24] F. Cai, J. M. Correll, S. H. Lee, Y. Lim, V. Bothra, Z. Zhang, M. P. Flynn, W. D. Lu, *Nat. Electron.* **2019**, *2*, 290.
[25] R. Waser, M. Aono, *Nat. Mater.* **2007**, *6*, 833.
[26] S. Siegel, M.-J. Yang, J.-P. Strachan, In *2025 IEEE Inter. Symp. on Circuits and Systems (ISCAS)*, IEEE, New York City, New York, USA **2025**.
[27] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, D. Metzler, arXiv preprint arXiv:2011.04006 **2020**.
[28] B. Cramer, Y. Stradmann, J. Schemmel, F. Zenke, *IEEE Trans. Neural Networks Learn. Syst.* **2020**, *33*, 2744.
[29] Y. Zhang, Z. Zhang, L. Lew, In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition* IEEE, New York City, New York, USA **2022**, pp. 12475–12485.
[30] M. Rao, H. Tang, J. Wu, W. Song, M. Zhang, W. Yin, Y. Zhuo, F. Kiani, B. Chen, X. Jiang, et al., *Nature* **2023**, *615*, 823.
[31] T. Kumar, Z. Ankner, B. F. Spector, B. Bordelon, N. Muennighoff, M. Paul, C. Pehlevan, C. Ré, A. Raghunathan, arXiv preprint arXiv:2411.04330 **2024**.
[32] C. Li, J. Ignowski, X. Sheng, R. Wessel, B. Jaffe, J. Ingemi, C. Graves, J. P. Strachan, *IEEE Inter. Memory Workshop*, IEEE, New York City, New York, USA **2020**.
[33] M.-J. Yang, Z. Yu, G. Pedretti, E. Neftci, J. P. Strachan, In *2025 IEEE Inter. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, New York City, New York, USA **2025**.
[34] M.-J. Yang, J. P. Strachan, *ICONS '23: Proceedings of the 2023 Inter. Conf. on Neuromorphic Systems* **2023**.
[35] Z. Yu, M.-J. Yang, J. Finkbeiner, S. Siegel, J. P. Strachan, E. Neftci, In *2024 IEEE Inter. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*. **2024**.

© 2026 The Author(s). Advanced Intelligent Systems published by Wiley-VCH GmbH

26404567, 0, Downloaded from https://advanced.onlinelibrary.wiley.com/doi/10.1002/aisy.202501019 by Universitatsbibliothek RWTH Aachen University, Wiley Online Library on [11/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License