Interspeech 2025
17-21 August 2025, Rotterdam, The Netherlands

Running Conventional Automatic Speech Recognition on Memristor Hardware: A Simulated Approach

Nick Rossenbach[1,3], *Benedikt Hilmes*[1,3], *Leon Brackmann*[2], *Moritz Gunz*[3], *Ralf Schlüter*[1,3]

[1]Machine Learning and Human Language Technology, RWTH Aachen University, Germany
[2]Institute for Electronic Materials II, RWTH Aachen University, Germany
[3]AppTek GmbH, Aachen, Germany

{rossenbach,hilmes,schlueter}@ml.rwth-aachen.de, l.brackmann@iwe.rwth-aachen.de, mgunz@apptek.com

Abstract

Memristor-based hardware offers new possibilities for energy-efficient machine learning (ML) by providing analog in-memory matrix multiplication. Current hardware prototypes cannot fit large neural networks, and related literature covers only small ML models for tasks like MNIST or single word recognition. Simulation can be used to explore how hardware properties affect larger models, but existing software assumes simplified hardware. We propose a PyTorch-based library based on "Synaptogen" to simulate neural network execution with accurately captured memristor hardware properties. For the first time, we show how an ML system with millions of parameters would behave on memristor hardware, using a Conformer trained on the speech recognition task TED-LIUMv2 as example. With adjusted quantization-aware training, we limit the relative degradation in word error rate to 25% when using a 3-bit weight precision to execute linear operations via simulated analog computation.

Index Terms: speech recognition, neuromorphic hardware, memristor simulation

1. Introduction

Artifical neural networks (ANN) play an important role in natural language processing (NLP) tasks such as automatic speech recognition (ASR). The majority of current ANN architectures used in NLP such as LSTM [1] or Transformer [2] and related derivatives make use of tensor operations such as vector-matrix-multiplication (VMM). VMM-based neural networks are typically executed via graphic processing units (GPU) containing many parallel processors for efficient computation of large VMM operations. By introducing special processor units such as Tensor Cores in Nvidia GPUs or dedicated accelerators such as Google TPUs [3], the efficiency further increased for many operations [4]. Still, the underlying chip technology is based on complementary metal-oxide semiconductors (CMOS).

Memristor technologies offer the possibility to perform mathematical operations on analog level using a programmable but non-volatile resistance state [5]. This is especially suited for VMMs in ANNs, as the trainable weights of the network usually stay fixed during execution. The computation is performed in constant time by applying the input vector as voltages to a memristor crossbar that physically stores the weights as resistance, and reading the resulting currents. This requires a fraction of the energy consumption compared to CMOS-based circuits [6]. A downside is the non-deterministic physical deviation of the resistance encoding. While there is prior work discussing the scope and impact of these uncertainties, current prototype hardware often only allows storing multiple thousands of parameters [7]. Thus, implications of those uncertainties for machine learning applications were only investigated on very small tasks and with limited numbers of parameters.

In this work, we want to utilize the availability of simulation software that can mimic the stochastic behavior of an actual device. We make use of Synaptogen [8] to develop a compact framework to mimic the execution of PyTorch-based neural networks for ASR on real memristor hardware. While there are existing frameworks for memristor simulation [9, 10], these make use of simplified modeling assumptions. A detailed discussion of the modeling differences and existing frameworks will follow in Section 2.2. Current literature covering the effects of memristor properties on ML tasks [11, 12, 13] targets only small tasks such as MNIST [14] or CIFAR [15] for imaging and Google Speech Commands [16] for speech. In addition, current literature such as [11, 12, 13] often omits important information about how the models are trained and mapped, and focus more on the aspects of the hardware itself. It should be noted that for our work, we assume that the memristive hardware would exist as general purpose VMM accelerator component.

The contributions of this work are as follows: We give an overview of the available memristor technology and briefly explain the consequences of the hardware properties for input-to-weight multiplications. We present a mapping of the PyTorch nn.Linear operations to the device-based simulation of Synaptogen to simulate execution with realistic device characteristics. In order to have a model that can be mapped, we explain our quantization-aware training scheme targeting very low-precision network weights with resolutions of down to 3 bits. Finally, we present the results for a Conformer-based ASR system trained on the TED-LIUMv2 [17] task and show the degradations caused by the low-precision computation and the memristor hardware characteristics. To our knowledge, this work is the first approach to discuss the influence of memristor hardware on a large scale NLP task. The PyTorch extension for Synaptogen, the model training code and other software [18, 19] are publicly accessible.[1]

2. Simulated Memristor Hardware

A memristor is a passive device whose resistance changes in response to the application of a sufficiently large electric voltage [5]. After the voltage is removed, the resistance of the device is conserved until a sufficiently large voltage is applied again. By varying the voltage amplitude, the resistance of the device can be adjusted incrementally, allowing the remanent programming of multiple resistance levels in one device [20]. However, memristors exhibit an intrinsic stochastic behavior. Particularly the programming process remains challenging, as the resistance levels often demonstrate significant variations [21]. In their origin,

[1]https://github.com/rwth-i6/returnn-experiments/tree/master/2025-memristor-asr

10.21437/Interspeech.2025-731

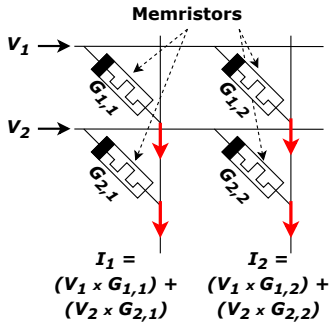


Figure 1: *Schematic representation of a Vector-matrix-multiplication in a* 2×2 *memristor crossbar. A weight matrix is programmed as conductance level of the memristors,* $G_{i,j}$ $(i, j \in [1, 2])$*. An input vector* (V_1, V_2) *is applied at the horizontal lines. The resulting current per column,* $I_{1,2}$ *yields the sum of the products between the inputs and the conductance.*

these variabilities are comparable to the phenomena of mismatch and process variations in existing CMOS technologies and call for precise characterization.

Memristor hardware has yet to see widespread commercial availability. Semiconductor manufacturers Infineon and TSMC just recently entered the market with integrated processors featuring memristors for automotive use [22].

2.1. Memristors for Vector-Matrix-Multiplication

Memristor crossbars are becoming a focus of research due to their ability to perform vector-matrix multiplication (VMM) directly within memory, eliminating the need for data transfer between storage and processing units [7]. Consequently, memristors are proposed as analog hardware for neural network computation, potentially augmenting or even replacing current digital processors [23]. A memristor crossbar consists of two perpendicular sets of voltage lines and a memristor device placed at each line intersection, as depicted in Figure 1. In a memristor-based VMM, the analog values of a weight matrix, M, are mapped to the multiple resistance states of the memristors inside a crossbar. To better relate the resistance state to the weights of the neural network, we describe a memristor state using the conductance G, which is the inverse of the resistance. The input vector V is applied as voltages along the horizontal crossbar lines. According to Ohm's Law and Kirchhoff's current law, the current, I, flowing through each vertical line is the sum of the individual products of input and conductance [6]. Physical implementations may also contain transistors to help programming [23]. Yet, the memristor variations can induce computational inaccuracies, which impact VMM performance. Consequently, these variability effects have to be carefully investigated to ensure reliable operation.

2.2. Synaptogen

Full realizations of memristor-based machine learning applications remain challenging due to strict hardware boundaries, e.g. the number of crossbars within a chip. Therefore, simulation tools tailored for memristor artificial neural networks have been developed. Prominent examples are the open-source MemTorch framework [9] and the AIHW KIT by IBM [10]. These tools utilize the PyTorch framework to simulate ANN applications and layers based on memristor crossbars on a high level of abstraction. However, these simulators lack accuracy in modeling the individual memristor behavior, especially the accurate definition of variability that significantly increases computational complexity. Although MemTorch offers several device models, including a data-driven model, the memristor parameters are idealized and the variations are based on arbitrary conductance distributions. The variability-aware memristor model Synaptogen has been proposed to address these limitations for the accurate simulation of memristor hardware [8]. The simulation model captures memristor variations based on vector autoregression, which was trained on measured electrical characteristics of over 3 million programming cycles. The captured devices in Synaptogen are state-of-the-art foundry quality oxide-based memristors and commercially available based on a 130 nm process from STMicroelectronics. Synaptogen incorporates device and programming variations as well as physical noise sources such as thermal noise (Johnson-Nyquist) and bit quantization. However, Synaptogen is not a high-level simulator, such as MemTorch and the AIHW KIT. Instead, it is a mathematical device model to provide realistic hardware behavior. In order to utilize Synaptogen for large-scale ANN designs, we implemented a PyTorch extension to execute it on GPUs.

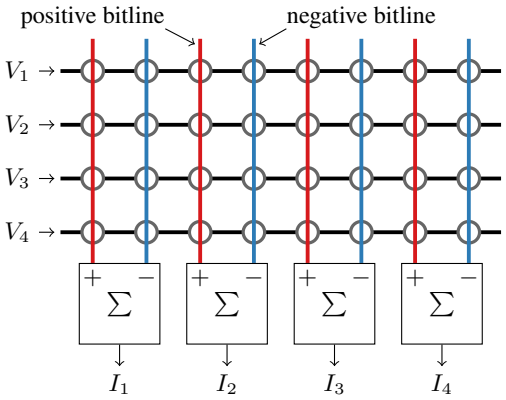


Figure 2: *Crossbar arrangement for pairwise memristors. Inspired by [6].*

2.3. Neural Network Operations with Synaptogen

Memristors can only cover the range between a specific low conductance and high conductance. To represent a weight of zero an infinite resistance would be needed so that no current flows for any applied voltage. As solution, we follow [6] and use the pairwise difference of memristors to model a single weight as depicted in Figure 2. With the subtraction of two cells in low-conductance state, a weight of zero can be achieved. Furthermore, encoding negative weights becomes possible. Thus, we model an $N \times M$ matrix with N inputs and M outputs and internal states -1, 0, and 1 using an $N \times 2M$ memristor crossbar. While it is possible to also use intermediate conductance levels, those are much more imprecise to configure, as shown later in this section. The physical input range without switching the cell is $-0.6V$ to $0.6V$. In order to be able to interpret the computation done by the memristor crossbar, we need to map the resulting output currents back to the desired value space. We introduce a correction factor c with the unit $\frac{1}{A}$ so that for the difference between the output currents of a cell on the positive bitline I^+ and negative bitline I^- the following holds for any x in [-1, 1]:

$$\left(I^+_{high}(x \cdot 0.6V) - I^-_{low}(x \cdot 0.6V)\right) \cdot c \approx x \quad (1)$$

$$\left(I^+_{low}(x \cdot 0.6V) - I^-_{low}(x \cdot 0.6V)\right) \cdot c \approx 0 \quad (2)$$

$$\left(I^+_{low}(x \cdot 0.6V) - I^-_{high}(x \cdot 0.6V)\right) \cdot c \approx -x \quad (3)$$

Table 1: *Computation results of 10000 single simulated cells set to perform the given multiplication in a [0,1] normalized space for input and weight. The first three rows show the non-linearity w.r.t input voltage on a single fixed device. Row #4 shows the noise with zero weight due to the paired subtraction of two crossbar columns necessary. Row #5 shows the increased uncertainty when trying to set "half" conductance states.*

operation	average	std-dev	min	max
1.0 x 1	1.016	0.063	0.707	1.27
0.1 x 1	$0.989e^{-1}$	$0.062e^{-1}$	$0.688e^{-1}$	$1.24e^{-1}$
0.01 x 1	$0.985e^{-2}$	$0.069e^{-2}$	$0.721e^{-2}$	$1.20e^{-2}$
1.0 x 0	$8.52e^{-4}$	$4.75e^{-2}$	-0.2751	0.2455
1.0 x 0.5	0.476	0.094	0.109	1.06

We write I_{high} for a cell in high conductance state, and I_{low} for a low conductance state. For the paired memristor model in Synaptogen, the optimal c w.r.t. a quadratic error was $8020\frac{1}{A}$. Given that the memristors can not be precisely set to the desired conductance states, we obtain large deviations in the computations. Table 1 shows the results for the different basic operations. The last row shows the increased variation when trying to set the cell on the positive bitline to a state in between the low and high conductance, e.g. for achieving a weight of 0.5. Thus, for this work we stick to binary programming of the states. In order to model weights with higher precision, we can stack multiple crossbars, where each crossbar represents a bit level. That means for a 4-bit resolution, a first crossbar would model the weight states -4, 0, 4, the second -2, 0, 2 and the last -1, 0, 1. Thus, with a stack of 3 crossbars we can model 15 different weight levels from -7 to 7. In the physical hardware the input and output levels can not be freely chosen. In this work, we assume a resolution of 8 bit for the digital-to-analog converter (DAC) providing the input voltage and the analog-to-digital converter (ADC) reading the output currents. As memristors only support a specific physical value range, any inputs and parameters have to be scaled with their respective quantization scales. Further explanation will be given in Section 4. In order to fit larger matrices into the crossbars, we tile the weights across multiple crossbars of e.g. 128×128 memristor pairs, as this is a size that can be assumed to be the realistic maximum for current hardware [24]. All DAC, ADC and tiling parameters are freely configurable in our framework. The memristor cell conductance state functions and additional noises are not configurable, but strictly based on the Synaptogen simulation parameters of real hardware.

3. Automatic Speech Recognition

Our ASR system consists of a Conformer encoder [25] with a Connectionist-Temporal-Classification (CTC) [26] output loss layer. We chose the Conformer as encoder architecture as it is widely used in current research literature [27]. While most layers in a Conformer block make use of VMMs with static matrices where modeling with memristor hardware is possible, dynamic operations such as self-attention, layer-norm or gating do not have a static weight component, and are excluded from being executed on the memristor devices. For feature extraction we use log-mel features with a shift of 10ms, followed by a stack of convolutional layers to perform down-sampling of factor 4 on the frame level. We exclude the feature extraction and the down-sampling network from the simulated memristor execution. Our target labels are ARPA-phonemes, and we use the official lexica provided with the dataset. For decoding, we use the lexical prefix-tree search implemented in the open-source decoder Flashlight [28], which is accessible via a Python interface in TorchAudio. We include 4-gram count-based language model using the KenLM [29] interface in the recognition process.

4. Quantization-Aware Training

Previous research utilizes already trained models, usualy using post-training quantization (PTQ) to quantize the models for further usage, as also done in MemTorch. While PTQ works well for 8-bit precision formats, the performance degrades with decreasing bit depth [30]. This is because lower bit precision introduces more noise to the computations, compared to what the model sees during training. Nevertheless, these lower bit precisions are required for working with memristors, as they only allow for a low (or binary) precision, and stacking largely increases the number of necessary crossbars within a chip. Quantization-aware training (QAT) describes the process of adapting the model to this quantization noise already during training [30]. This way, the model can learn to deal with possible inaccuracies that will occur during (lower-bit) inference. For this, an observer is placed at the position of weights and activations during training , capturing the possible values during the current forward step. We fake-quantize each tensor into the desired precision, meaning while computations still happen in FP32, only a limited number of values are allowed. One important property for memristor computation is the need for a symmetric quantization range with a fixed zero-point at 0. This can be challenging for PTQ, as there is no control over the value range and the distribution the model learns during training. In contrast, QAT allows to directly induce this property during training, nudging the model to learn a symmetric distribution within the given bit precision.

5. Simulated Execution of Conformer Blocks

During QAT, we have observers that track the statistics of each weight and input. Assuming that for a specific linear layer we would have tracked the largest absolute input value to be 4 and the largest absolute weight value to be 0.1, we would have the input scaling factor 0.25 and a weight factor of 70 for a 3-crossbar stack with level representations from -7 to 7. For each memristor in the simulated cell pairs representing each weight, we would apply a large positive or negative voltage to set the desired high or low conductance states. During execution, we quantize the normalized input values with the DAC settings and apply the resulting input voltage via the cell simulation function. After reading the results from the simulated crossbars, we apply the previously calculated correction factor c, apply the ADC quantization, bit-shift the results with respect to their weight bit level, and then divide both the input factor and weight factor to retrieve the desired output. This process is performed for all parameterized linear transformations in the conformer encoder with the sole exception of the depth-wise convolution. This means that only less than 1% of the trainable weights of the conformer blocks remain outside of the memristor crossbars. Parameter-free operations such as activation functions, gating, the dot product of the self attention, and the softmax operations are computed as usual. The DAC precision which corresponds to the activation quantization during QAT, as well as the ADC precision, is fixed to 8 bits. As there is a high uncertainty of the resulting conductance when setting the weights in the memristor simulation, we report the recognition results over 10 different runs. Between each run, we apply randomly drawn voltages multiple times to the memristor to reset the conductance states in order to mitigate correlation effects across programming cycles.

Table 2: *Post-training quantization vs. quantization-aware training on TEDv2 dev. Activations are quantized with 8-bit precision.*

Weight Resolution (Bits)	WER [%]↓	
	PTQ	QAT
8	7.2	7.3
6	7.9	7.7
5	8.9	7.4
4	11.4	7.8
3	30.7	8.3
2	n.a.	22.1
Non-Quant Baseline	7.2	

6. Experiments

6.1. Experimental Setup

We conduct our experiments on TED-LIUMv2 (TEDv2) [17], which consists of 207 hours of TED talks. We use the dev-set of TED-LIUMv2 for evaluation. For recognition, we use a 4-gram LM trained on the text data provided with the corpus.

The model consists of 12 Conformer layers with a model dimension of 384 and a feed-forward dimension of 1536. The total number of parameters is 42M. We train the model for 50 epochs with RAdam [31], using a linearly increasing and decreasing learning rate schedule with a maximum learning rate of $5e^{-4}$. When performing QAT, we use the value range observers and the fake quantization operations right from the beginning of the training, so no non-QAT pre-training is performed. The baseline and QAT trainings are performed with identical hyperparameter settings, except for the added quantization operations.

Given that recognition with the simulation software is much slower than regular recognition, we have a real-time-factor of around 1 to 1.5 on an Nvidia L40S.

6.2. Effect of Quantization

Our first comparison is done between PTQ and QAT for low-precision weights. Table 2 shows the results for weight bit levels from 8-bit down to 2-bit. For 8-bit both approaches are on par with the non-quantized baseline. PTQ quickly degrades while QAT stays within 15% relative degradation when going to 3-bit weights. Despite 2-bit QAT training being possible, it suffers from substantial degradation. We conclude that quantizing only after the training is not a suitable approach to prepare the model for memristor deployment.

6.3. Effect of Simulation

For testing the memristor simulation, we take the models that are trained with QAT using 3-bit to 8-bit weight settings. We prepare the models by applying the steps defined in Section 5, and then run the recognition as usual. We do not alter any other settings and perform no specific adaptation after applying the programming voltages and letting the simulation determine the cell conductance. This is in contrast to other work, where the trained networks are specifically tuned towards the resulting conductance states [11]. Table 3 shows the results when taking the QAT trained models and executing recognition with the memristor model. One can see that the deviation from the baseline is substantial, but still in a reasonable range, given that no particular adjustments w.r.t. the hardware were made. There is a noticeable difference in WER for each specific programmed instance of the device, but the outliers do not exceed 5% relative degradation compared to the mean. Thus, we show that it is possible to map a Conformer model trained with QAT to a realistically modeled memristor device and achieve reasonable WER.

Table 3: *Results of Conformer CTC recognition on TEDv2 dev when using the simulated memristor hardware. Results are averages over 10 differently drawn devices, including the standard-deviation as well as minimum and maximum over the runs.*

Weight Resolution (Bits)	WER [%]↓		
	avg ± std	min	max
8	8.3 ± 0.13	8.1	8.5
6	8.3 ± 0.09	8.2	8.5
5	8.3 ± 0.16	8.1	8.6
4	8.9 ± 0.12	8.7	9.2
3	9.2 ± 0.12	9.0	9.4
Non-Quant Baseline	7.2		

7. Limitations and Future Work

This work only investigates the effect of a particular kind of memristor hardware on linear transformation layers for a single speech recognition system. As chip co-integration has been disregarded in this work, we can not make any assumptions yet about the speed and energy benefit achieved by using memristor hardware. In the future, we would like to focus on aspects that are relevant for the direction of research regarding the hardware capabilities. This would include research about programming the memristors with more than 2 states as well as a more detailed investigation about the needed ADC and DAC resolutions and ranges. Additionally, we would like to extend the current implementation to further network components such as convolutions or recurrent units such as LSTMs [1]. Through this, we can investigate the behavior of further ASR architectures such as Transducers [32]. Outside a simulation, the maximum of allowed model parameters is given by the crossbar count of a specific memristor chip and the used mapping method. Future work should investigate the optimal trade-off between layer count, model size and necessary bit resolution either via multi-conductance states or crossbar stacking.

8. Conclusion

In this work, we presented how a Conformer-based ASR model performs on simulated memristor hardware that introduces a strong uncertainty in tensor operations. We present a PyTorch extension to the Synaptogen toolkit to easily map nn.Linear layers in order to allow neural networks to be executed via a realistic memristor device simulation. We described a detailed approach on how to simulate the execution of a Conformer ASR system on a memristor based device. We could show that even under strong weight deviations and non-linear behavior, the system can perform reasonably well using solely QAT and no further hardware-specific tuning. Given the ability to place a sufficient number of memristor crossbars on a single chip, the current precision of memristor crossbars is already sufficient when being used in a binary-paired setting. While there are many open questions on how actual hardware would perform in terms of energy and speed, we can show that the non-deterministic behavior of memristors poses no restrictions on replacing fixed-weight linear operations with memristive counterparts. Nevertheless, many issues regarding scaling the number of crossbars in a chip and the co-integration with regular computation devices have to be solved in the future.

9. Acknowledgments

This work was partially supported by NeuroSys, which as part of the initiative "Clusters4Future" is funded by the Federal Ministry of Education and Research BMBF (funding IDs 03ZU2106DA and 03ZU2106DD).

10. References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, 1997.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, Dec. 2017, pp. 6000–6010.

[3] N. P. Jouppi, G. Kurian, S. Li, P. C. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. A. Patterson, "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA 2023, Orlando, FL, USA, June 17-21, 2023*, Y. Solihin and M. A. Heinrich, Eds. ACM, 2023, pp. 82:1–82:14.

[4] B. Hanindhito and L. K. John, "Accelerating ml workloads using gpu tensor cores: The good, the bad, and the ugly," in *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 178–189.

[5] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008.

[6] F. Aguirre, A. Sebastian, M. Le Gallo, W. Song, T. Wang, J. J. Yang, W. Lu, M.-F. Chang, D. Ielmini, Y. Yang, A. Mehonic, A. Kenyon, M. A. Villena, J. B. Roldán, Y. Wu, H.-H. Hsu, N. Raghavan, J. Suñé, E. Miranda, A. Eltawil, G. Setti, K. Smagulova, K. N. Salama, O. Krestinskaya, X. Yan, K.-W. Ang, S. Jain, S. Li, O. Alharbi, S. Pazos, and M. Lanza, "Hardware implementation of memristor-based artificial neural networks," *Nature Communications*, vol. 15, no. 1, p. 1974, Mar 2024.

[7] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H.-S. P. Wong, and G. Cauwenberghs, "A compute-in-memory chip based on resistive random-access memory," *Nature*, vol. 608, no. 7923, pp. 504–512, 08 2022.

[8] T. Hennen, L. Brackmann, T. Ziegler, S. Siegel, S. Menzel, R. Waser, D. J. Wouters, and D. Bedau, "Synaptogen: A cross-domain generative device model for large-scale neuromorphic circuit design," *IEEE Transactions on Electron Devices*, vol. 71, no. 9, pp. 5345–5353, 2024.

[9] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, "MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems," *Neurocomputing*, 2022.

[10] M. J. Rasch, D. Moreda, T. Gokmen, M. Le Gallo, F. Carta, C. Goldberg, K. El Maghraoui, A. Sebastian, and V. Narayanan, "A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[11] T.-H. Wen, J.-M. Hung, W.-H. Huang, C.-J. Jhang, Y.-C. Lo, H.-H. Hsu, Z.-E. Ke, Y.-C. Chen, Y.-H. Chin, C.-I. Su, W.-S. Khwa, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, M.-S. Ho, C.-C. Chou, Y.-D. Chih, T.-Y. J. Chang, and M.-F. Chang, "Fusion of memristor and digital compute-in-memory processing for energy-efficient edge computing," *Science*, vol. 384, no. 6693, pp. 325–332, 2024.

[12] J. Souto, G. Botella, D. García, R. Murillo, and A. del Barrio, "Neuromorphic circuit simulation with memristors: Design and evaluation using memtorch for mnist and cifar," 2024.

[13] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "Rxnn: A framework for evaluating deep neural networks on resistive crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 326–338, 2021.

[14] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[16] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, Apr. 2018.

[17] A. Rousseau, P. Deléglise, and Y. Estève, "Enhancing the ted-lium corpus with selected data for language modeling and more ted talks," in *International Conference on Language Resources and Evaluation*, 2014.

[18] J. Peter, E. Beck, and H. Ney, "Sisyphus, a workflow manager designed for machine translation and automatic speech recognition," in *EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pp. 84–89.

[19] P. Doetsch, A. Zeyer, P. Voigtlaender, I. Kulikov, R. Schlüter, and H. Ney, "Returnn: The RWTH extensible training framework for universal recurrent neural networks," in *ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pp. 5345–5349.

[20] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, no. 1, pp. 13–24, Jan 2013.

[21] D. J. Wouters, Y.-Y. Chen, A. Fantini, and N. Raghavan, *Reliability Aspects*. John Wiley & Sons, Ltd, 2016, ch. 21, pp. 597–622.

[22] Infineon Technologies AG. (2022). [Online]. Available: https://www.infineon.com/cms/de/about-infineon/press/market-news/2022/INFATV202211-031.html

[23] Y. Huang, T. Ando, A. Sebastian, M.-F. Chang, J. J. Yang, and Q. Xia, "Memristor-based hardware accelerators for artificial intelligence," *Nature Reviews Electrical Engineering*, vol. 1, no. 5, pp. 286–299, May 2024.

[24] J. Chen, S. Yang, H. Wu, G. Indiveri, and M. Payvand, "Scaling limits of memristor-based routers for asynchronous neuromorphic systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 3, pp. 1576–1580, 2024.

[25] A. Gulati, C.-C. Chiu, J. Qin, J. Yu, N. Parmar, R. Pang, S. Wang, W. Han, Y. Wu, Y. Zhang, and Z. Zhang, Eds., *Conformer: Convolution-augmented Transformer for Speech Recognition*, 2020.

[26] A. Graves, S. Fernández, and F. Gomez, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *In Proceedings of the International Conference on Machine Learning, ICML*, 2006, pp. 369–376.

[27] R. Prabhavalkar, T. Hori, T. N. Sainath, R. Schlüter, and S. Watanabe, "End-to-end speech recognition: A survey," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 325–351, 2024.

[28] J. D. Kahn, V. Pratap, T. Likhomanenko, Q. Xu, A. Hannun, J. Cai, P. Tomasello, A. Lee, E. Grave, G. Avidov, B. Steiner, V. Liptchinsky, G. Synnaeve, and R. Collobert, "Flashlight: Enabling innovation in tools for machine learning," in *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, 17–23 Jul 2022, pp. 10 557–10 574.

[29] K. Heafield, "KenLM: Faster and smaller language model queries," in *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland: Association for Computational Linguistics, Jul. 2011, pp. 187–197.

[30] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021.

[31] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[32] A. Graves, "Sequence Transduction with Recurrent Neural Networks," in *Proc. ICML Workshop on Representation Learning*, Edinburgh, Scotland, Jun. 2012.