ISCAS 2020
Virtual, October 10-21
IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS
CAS
IEEE CIRCUITS AND SYSTEMS SOCIETY

# MemTorch: A Simulation Framework for Deep Memristive Cross-Bar Architectures

**Corey Lammie** and Mostafa Rahimi Azghadi

College of Science and Engineering, James Cook University, Queensland 4814, Australia

2020 IEEE International Symposium on Circuits and Systems
Virtual, October 10-21, 2020

# Overview

- Background and theory
- Motivation and related works
- *MemTorch*:
  - A typical use-case workflow;
  - Implementation details;
  - Package structure of the initial release;
  - Quantization C++/CUDA extension;
  - Release management.
- Simulations
- Conclusion and outlook
- Acknowledgements

# Overview

- **Background and theory**
- Motivation and related works
- *MemTorch*:
  - A typical use-case workflow;
  - Implementation details;
  - Package structure of the initial release;
  - Quantization C++/CUDA extension;
  - Release management.
- Simulations
- Conclusion and outlook
- Acknowledgements

# Background and theory

- Memristive devices have shown great promise to facilitate the acceleration and improve the power efficiency of Deep Learning (DL) systems.
- Crossbar architectures constructed using memristive devices can be used to perform Multiply-Accumulate (MAC) operations in Θ(1) [1].

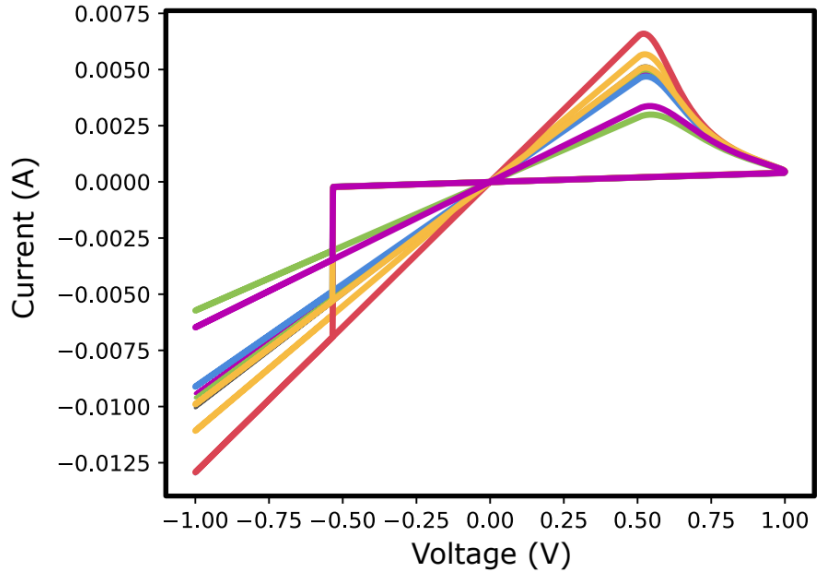


Fig. 1: I/V characteristics of a simulated VTEAM memristor model.

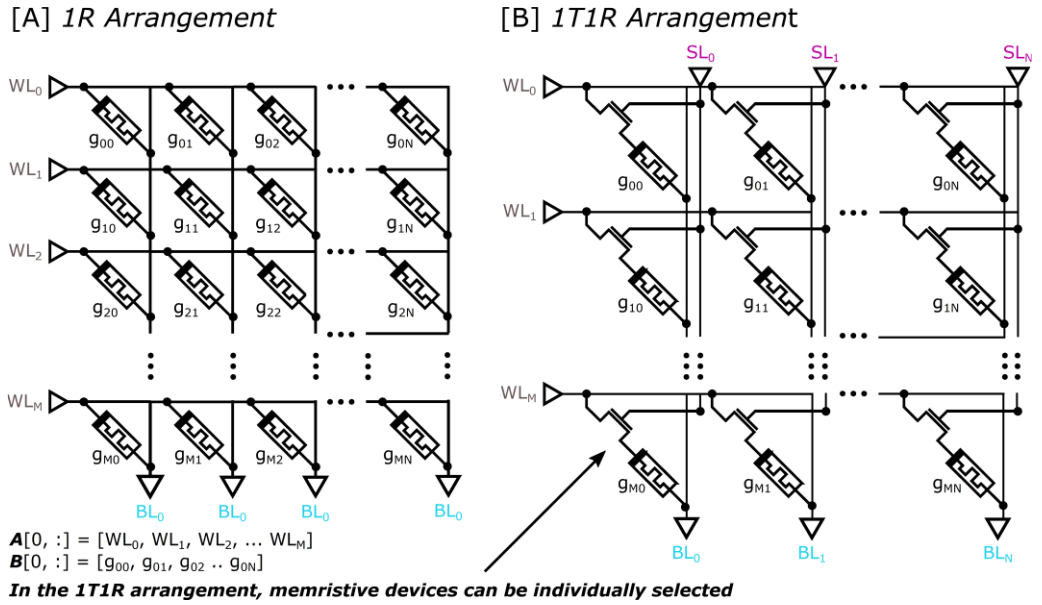


Fig. 2: Depiction of an M x N [A] 1R crossbar architecture and a [B] 1T1R crossbar architecture.

# Background and theory

- Memristors are still considered an emerging technology [2].
- Consequently, memristive DL systems are putative to be prone to severe errors due to a number of device limitations.

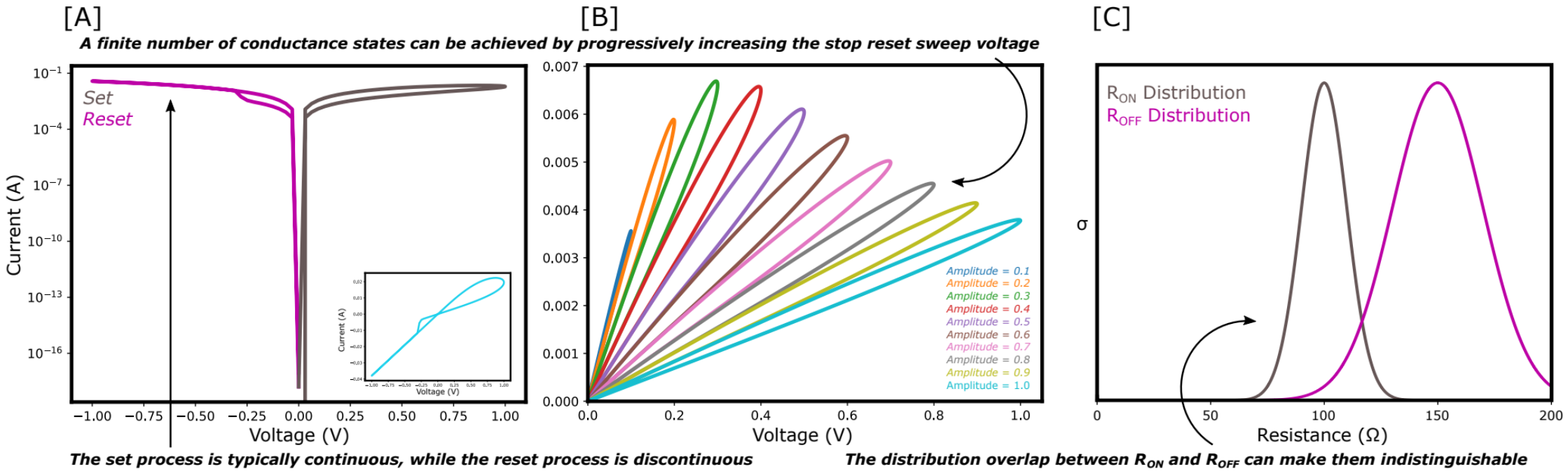


Fig. 3: Depiction of [A] device I/V characteristics, [B] reset voltage double-sweeps, and [C] distributions of $R_{ON}$ and $R_{OFF}$ for a non-ideal device.

# Overview

- Background and theory
- **Motivation and related works**
- *MemTorch*:
  - A typical use-case workflow;
  - Implementation details;
  - Package structure of the initial release;
  - Quantization C++/CUDA extension;
  - Release management.
- Simulations
- Conclusion and outlook
- Acknowledgements

# Motivation and related works

- There is a lack of a *modernized*, *open source* and *general high-level* simulation platform that can fully integrate any memristive device model and its putative non-idealities into crossbar architectures within DL systems.

Table 1: *Does not support GPU-accelerated inference and/or parameter mapping. †Models are shared using Google Drive without Application Programming Interfaces (APIs).

| Simulation framework | Open-source | GPU | Pretrained DNN conversion | Programming language(s) |
|---|---|---|---|---|
| RAPIDNN [3] | | ✓* | ✓ | C++ |
| MNSIM [4] | | | ✓ | Not Specified |
| PUMA [5] | | | ✓ | C++ |
| DL-RSIM [6] | | ✓ | ✓ | Python |
| PipeLayer [7] | | ✓* | ✓ | C++ |
| Tiny but Accurate [8] | ✓† | | ✓ | MATLAB |
| Ultra-Efficient Memristor-Based DNN Framework [9] | ✓† | | ✓ | C++, MATLAB |
| Non-ideal Resistive Synaptic Device Characteristic Simulation Framework [10] | | ✓ | ✓ | Python |
| ***MemTorch*** | ✓ | ✓ | ✓ | Python, C++, CUDA |

# Overview

- Background and theory
- Motivation and related works
- ***MemTorch***:
  - A typical use-case workflow;
  - Implementation details;
  - Package structure of the initial release;
  - Quantization C++/CUDA extension;
  - Release management.
- Simulations
- Conclusion and outlook
- Acknowledgements

# MemTorch

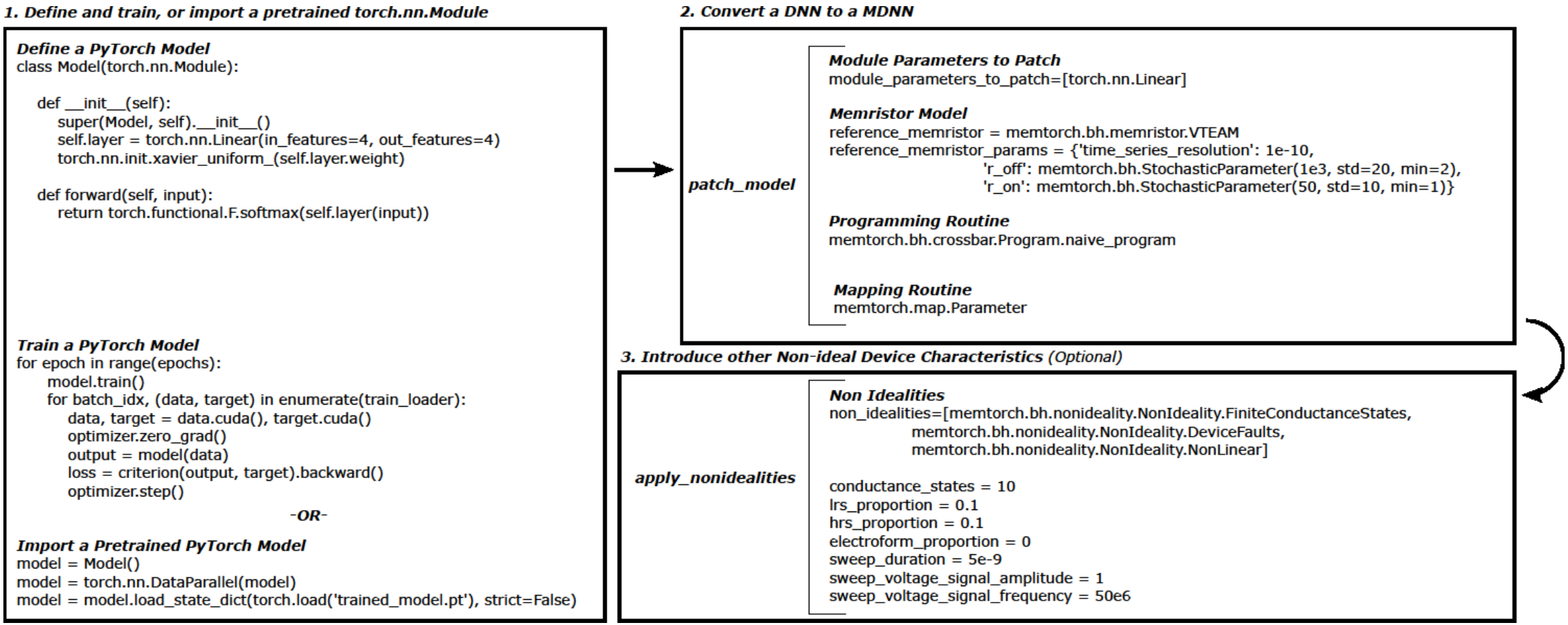


Fig. 4: A typical use-case workflow in MemTorch.

# MemTorch

- Is implemented using C++, CUDA and Python, with a Python interface.
- Relies heavily on the open source PyTorch [11] ML framework.
- Natively supports operation on CPUs and GPUs.

# MemTorch

Table 2: Package directory structure.

| MemTorch PyPI Package Directory Structure | Description |
|---|---|
| ▾memtorch | - |
| ▾memtorch.bh | Behavioral and experimental models |
| ▾memtorch.bh.memristor | Behavioral and experimental models of memristive devices |
| ▾memtorch.bh.memristor.Memristor[1] | Abstract class for behavioral and experimental models of memristive devices |
| memtorch.bh.memristor.LinearIonDrift[1] | Ideal linear ion drift behavioral memristor model [20] |
| ▾memtorch.bh.nonideality | Behavioral non-idealities |
| memtorch.bh.nonideality.NonIdeality[2] | Abstract class for behavioral non-idealities |
| memtorch.bh.nonideality.FiniteConductanceStates[2] | Conductance state behavioral non-ideality |
| memtorch.bh.nonideality.Variability[2] | Variability behavioral non-ideality |
| ▾memtorch.cu | C++/CUDA extensions |
| memtorch.cu.quantize[1] | Quantization extension used to model a finite number of conductance states |
| memtorch.cu.quantize.gpu[3] | CUDA header |
| memtorch.cu.quantize.quant[4] | CUDA quantization kernel |
| memtorch.cu.quantize.quant_cuda[5] | Python and C++ CUDA quantization kernel wrapper |
| ▾memtorch.mn | torch.nn equivalent |
| memtorch.mn.Module[2] | Functions to patch pre-trained DNNs |
| memtorch.mn.Linear[1] | Memristive linear layer |
| memtorch.mn.Conv2d[1] | Memristive conv2d layer |
| ▾memtorch.map | Cross-bar mapping algorithms |
| memtorch.map.Module[2] | Module cross-bar mapping algorithms |
| memtorch.map.Parameters[2] | Parameter cross-bar mapping algorithms |
| ▾memtorch.examples | Usage examples |
| memtorch.examples.GeneralUsage | General usage examples |
| ▾memtorch.examples.reproduce | Examples which reproduce the experiments in Section V |
| memtorch.examples.reproduce.Model[2] | Network architecture depicted in Section V |
| memtorch.utils[2] | Utility functions |

# MemTorch

- A quantization C++/CUDA extension is used to model a finite number of conductance states.
- A binary search is performed on a sorted tensor containing defined quantization states in $\Theta(n\log(n))$.
- For execution on GPUs this is massively parallelized using 128 CUDA threads and max(min($n$ + 127)/128, 4096), 1) CUDA blocks, where $n$ is the number of elements of the tensor to quantize.

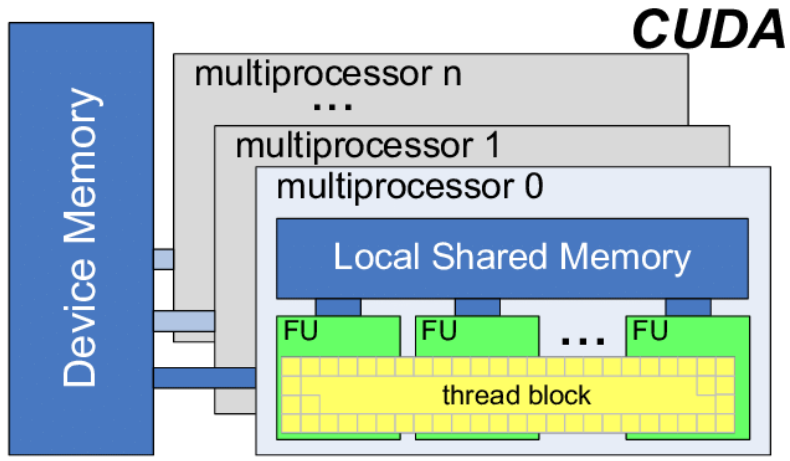


Fig. 5: NVIDIA CUDA architecture [12].

# MemTorch

- MemTorch is released using the *PyPi* repository and can easily be installed using *pip install memtorch* or *pip install memtorch-cpu*.
- MemTorch is completely open-source: *https://github.com/coreylammie/MemTorch*.
- ReadTheDocs documentation is publicly-accessible.
- The Travis Continuous Integration (CI) service is used for unit testing.
- MemTorch is licensed under the GNU General Public License v3.0.

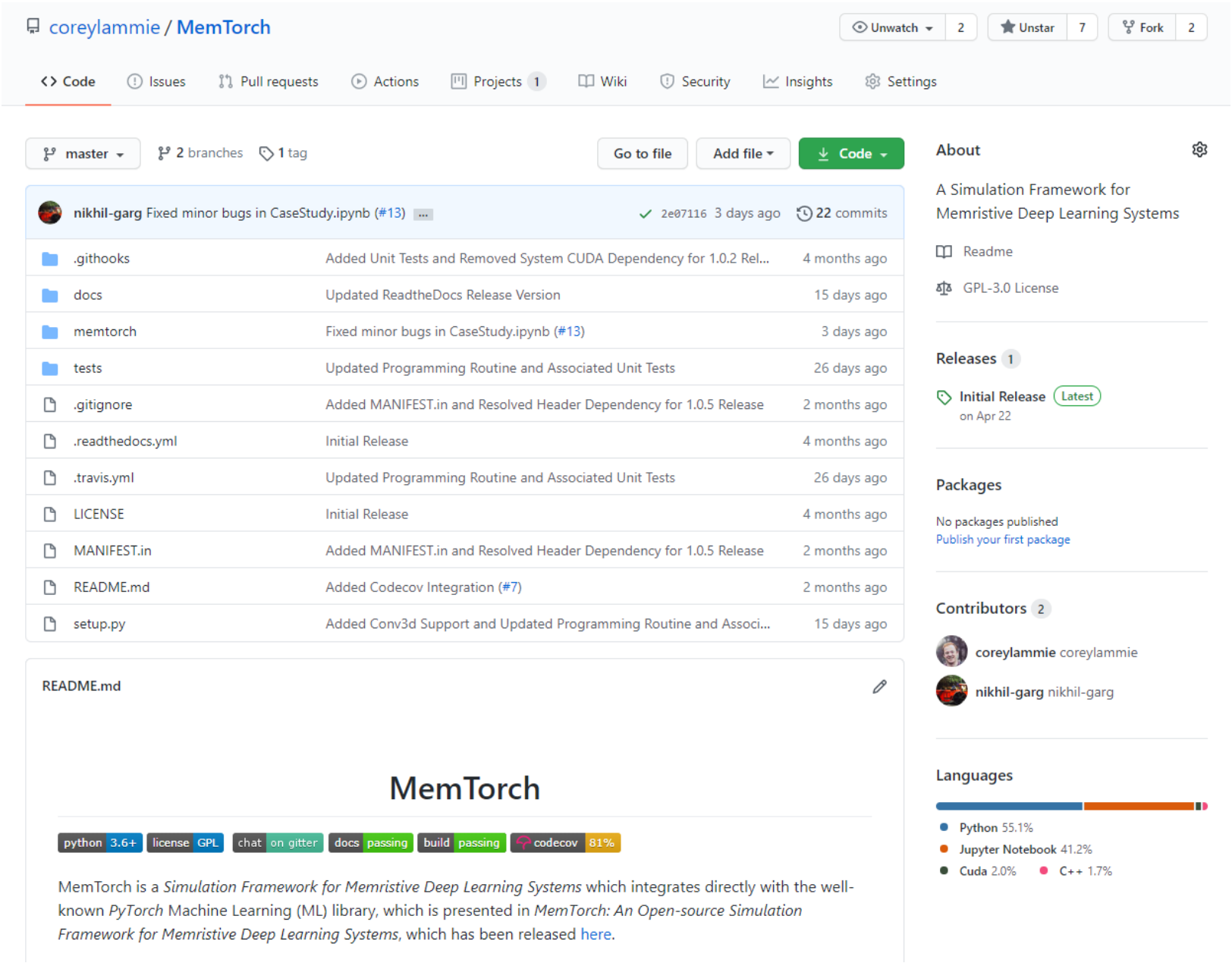


Fig. 6: A screenshot of the MemTorch GitHub repository.

# Overview

- Background and theory
- Motivation and related works
- *MemTorch*:
  - A typical use-case workflow;
  - Implementation details;
  - Package structure of the initial release;
  - Quantization C++/CUDA extension;
  - Release management.
- **Simulations**
- Conclusion and outlook
- Acknowledgements

# Simulations

- We investigate the performance degradation that three nonideal characteristics of memristive devices introduce to an ideal linear ion drift model for memristive devices: *device-to-device variation, the number of finite conductance states* and the $R_{ON}/R_{OFF}$ *ratio*.
- We converted a pretrained VGG-16 DNN, which achieves 91.41% on the CIFAR-10 test set.
- Each memristive layer's weights were mapped to a double column line cross-bar architecture using (1), where for the positive crossbar $\sigma(\boldsymbol{w}) = \boldsymbol{w}[\boldsymbol{w} \geq 0]$, and for the negative crossbar $\sigma(\boldsymbol{w}) = \boldsymbol{w}[\boldsymbol{w} \leq 0]$.

$$g[i,j] = \frac{(R_{ON} - R_{OFF})(\sigma(\boldsymbol{w})[i,j] - \boldsymbol{w}_{min})}{|\boldsymbol{w}|_{max} - \boldsymbol{w}_{min}} + R_{OFF} \qquad (1)$$

# Simulations

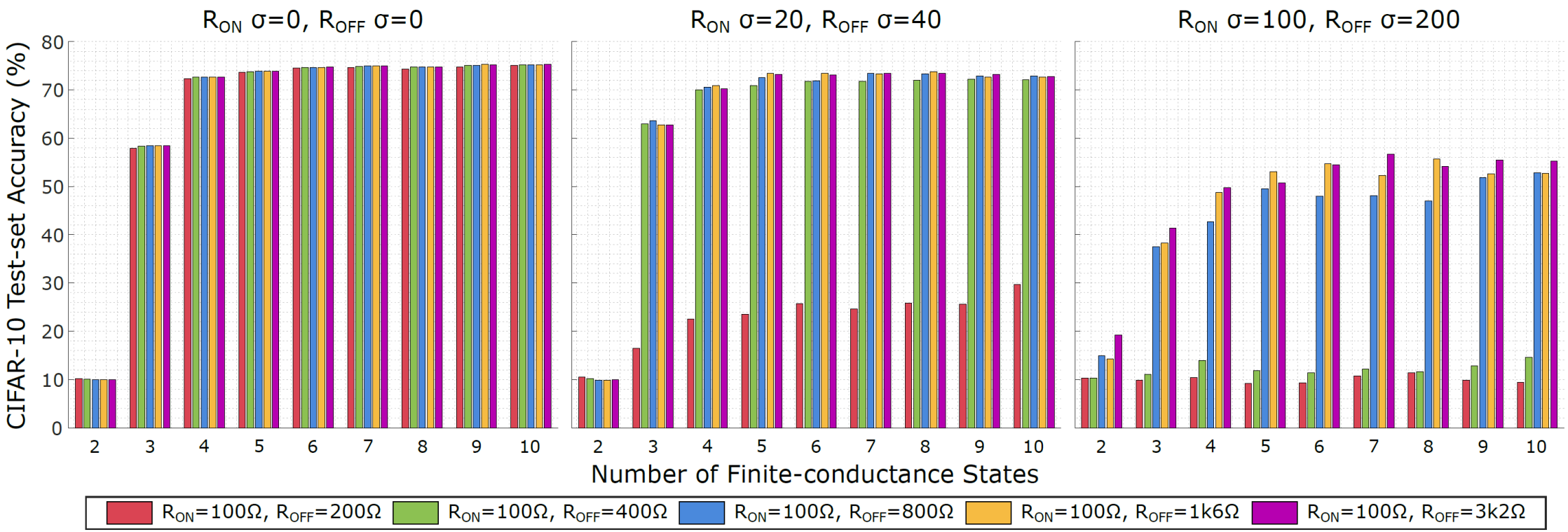


Fig. 7: Simulation results investigating the performance degradation of non-ideal MDNNs.

# Overview

- Background and theory
- Motivation and related works
- *MemTorch*:
  - A typical use-case workflow;
  - Implementation details;
  - Package structure of the initial release;
  - Quantization C++/CUDA extension;
  - Release management.
- Simulations
- **Conclusion and outlook**
- Acknowledgements

# Conclusion and outlook

- We presented an open source simulation framework for deep memristive cross-bar architectures entitled MemTorch.
- We performed experiments using it to demonstrate its functionality in large-scale simulations.
- Despite its currently limited scope, MemTorch has been designed with expandability and ease-of use in mind.
- We hope that MemTorch will be continuously used, expanded, and improved by the larger Circuits and Systems (CAS) community and designers alike.

# Overview

- Background and theory
- Motivation and related works
- *MemTorch*:
  - A typical use-case workflow;
  - Implementation details;
  - Package structure of the initial release;
  - Quantization C++/CUDA extension;
  - Release management.
- Simulations
- Conclusion and outlook
- **Acknowledgements**

# Acknowledgements


- The James Cook University Domestic Prestige Research Training Program Scholarship;
- My supervisor- Dr. Mostafa Rahimi Azghadi;
- My co-supervisor- Prof. Wei Xiang;
- Prof. Bernabé Linares-Barranco.

# References


[1] C. Lammie, O. Krestinskaya, A. James, and M. R. Azghadi, “Variation aware Binarized Memristive Networks,” in Proc. 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, Nov. 2019, pp. 490–493.

[2] G. C. Adam, A. Khiat, and T. Prodromakis, “Challenges Hindering Memristive Neuromorphic Hardware from Going Mainstream,” Nature Communications, vol. 9, no. 1, p. 5267, 2018.

[3] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, “RAPIDNN: In-Memory Deep Neural Network Acceleration Framework,” CoRR, vol. abs/1806.05794, 2018. [Online]. Available: http://arxiv.org/abs/1806.05794

[4] L. Xia, B. Li, T. Tang, P. Gu, P. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, “MNSIM: Simulation Platform for Memristor-Based Neuromorphic Computing System,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 5, pp. 1009–1022, May. 2018.

[5] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, “PUMA: A Programmable Ultra-efficient Memristor based Accelerator for Machine Learning Inference,” CoRR, vol. abs/1901.10351, 2019. [Online]. Available: http://arxiv.org/abs/1901.10351

[6] M. Lin, H. Cheng, W. Lin, T. Yang, I. Tseng, C. Yang, H. Hu, H. Chang, H. Li, and M. Chang, “DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning,” in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, Nov. 2018, pp. 1–8.

[7] L. Song, X. Qian, H. Li, and Y. Chen, “PipeLayer: A Pipelined ReRAM Based Accelerator for Deep Learning,” in Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, Feb. 2017, pp. 541–552.

[8] X. Ma, G. Yuan, S. Lin, C. Ding, F. Yu, T. Liu, W. Wen, X. Chen, and Y. Wang, “Tiny but Accurate: A Pruned, Quantized and Optimized Memristor Crossbar Framework for Ultra Efficient DNN Implementation,” arXiv e-prints, p. arXiv:1908.10017, Aug. 2019.

[9] G. Yuan, X. Ma, C. Ding, S. Lin, T. Zhang, Z. S. Jalali, Y. Zhao, L. Jiang, S. Soundarajan, and Y. Wang, “An Ultra-Efficient Memristor-Based DNN Framework with Structured Weight Pruning and Quantization Using ADMM,” arXiv e-prints, p. arXiv:1908.11691, Aug. 2019.

[10] X. Sun and S. Yu, “Impact of Non-Ideal Characteristics of Resistive Synaptic Devices on Implementing Convolutional Neural Networks,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 3, pp. 570–579, 2019.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in PyTorch,” in NIPS Autodiff Workshop, 2017.

[12] D. Chatterjee, A. DeOrio, and V. Bertacco, “Event-driven Gate-level Simulation with GPU-GPUs,” in Proc. 46th ACM/IEEE Design Automation Conference, 2009, pp. 557–562.

# Q&A