The Terabrain Project: Simulating billions of spiking neurons on standard computer hardware

Aditya Kar, Jessim Ahdjoudj, Dominique Longin, Simon Thorpe

The Terabrain Project: Simulating billions of spiking neurons on standard computer hardware

Aditya Kar *
Centre de Recherche Cerveau et Cognition (CerCo)
Institut de Recherche en Informatique de Toulouse (IRIT)
Toulouse, France
aditya.kar@cnrs.fr

Jessim Ahdjoudj
Centre de Recherche Cerveau et Cognition (CerCo)
Toulouse, France
jessim.ahdjoudj@univ-tlse3.fr

Dominique Longin
Institut de Recherche en Informatique de Toulouse (IRIT)
Toulouse, France
dominique.longin@irit.fr

Simon Thorpe
Centre de Recherche Cerveau et Cognition (CerCo)
Toulouse, France
simon.thorpe@cnrs.fr

***Abstract*—We describe a computing architecture capable of simulating networks with billions of spiking neurons using an ordinary Apple MacBook Air equipped with a M2 processor, 24 GB of on-chip unified memory and a 4TB solid-state disk. We use an event-based propagation approach that processes packets of N spikes from the M neurons in the system on each processing cycle. Each neuron has C binary input connections, where C can be 128 or more. During the propagation phase, we increment the activation values for all targets of the N neurons that fired. In the second step, we use the histogram of activation values to determine the firing threshold on the fly and select the N neurons that will fire in the next packet. We note that this active selection process could be related to oscillatory activity in the brain, which may have the function of fixing the percentage of neurons that fire on each cycle. Critically, there are absolutely no restrictions on architecture, since each neuron can have a direct connection to any other neuron, allowing us to have both feed-forward as well as recurrent connections. With $M = 2^{32}$neurons, this allows 2^{64} possible connections, although actual connectivity is extremely sparse. Even with off-the-shelf hardware, the simulator can continuously propagate packets with thousands of spikes and millions of connections dozens of times a second. Remarkably, all this is possible using an energy budget of just 37 watts, close to the energy required by the human brain. The work demonstrates that brain-scale simulations are possible using current hardware, but this requires fundamentally rethinking how simulations are implemented.**

***Index Terms*—Spiking neural networks, Brain-scale simulations, Binary weights, Sparse network, GPU acceleration, Apple M2 chip, Bio-plausible networks**

I. Introduction

Deep Neural Networks are now state of the art in many computing areas, achieving super-human performance for many applications, including image categorization [5], [17], [23], [24], audio analysis [4], [21], [27] and language processing [1], [6], [19]. The original AlexNet architecture from 2012 required 650,000 neurons with 60 million parameters [14]. But more recent systems, particularly the latest Large Language Models such as GPT-4, have hundreds of billions and even trillions of parameters. One problem with such systems is that they require a massive energy budget. Indeed, to implement a neural network the size of the human brain using the conventional processing strategies used in Artificial Neural Networks, the amount of energy required would be over a million times more than the 20 Watts used by the brain. How can the brain function on such a tiny energy budget? Could future hardware allow more efficient implementations? Several active research areas are related to this question. For example, novel devices such as memristors could implement synapses more efficiently than conventional logic circuits [15], [16], and in-memory computing could avoid the huge energetic cost of moving data typical with Von Neumann architectures [13].

However, in this paper, we argue that brain-scale simulations with billions of neurons are already possible using off-the-shelf computer hardware without the need for additional cutting-edge technology. Specifically, we demonstrate that it is perfectly possible to simulate a fully recurrent network with billions of neurons and hundreds of billions of synaptic connections using nothing more than a standard MacBook Air equipped with an M2 processor, 24GB of on-chip unified memory and 4 Terabytes of SSD memory. Even today, it is possible to obtain a MacStudio equipped with an M1 Ultra chip that has 128 GB of on-chip unified memory. This would be sufficient to simulate the entire human neocortex, which contains roughly 16 billion neurons. Given the speed with which such technology is advancing, it seems reasonable to suppose that within a year or so, it will be possible to simulate networks with more neurons than the entire human neocortex. Indeed, with the processing architecture that we are proposing, there is effectively no upper limit on the size of the neural networks that can be simulated.

II. Novel features of our architecture

Here, we build on ideas developed initially for Arnaud Delorme's SpikeNet simulator [7], [8] in the late 1990s, one of the first event-driven spiking neural simulators, still downloadable today on SpikeNET github. SpikeNet kept a list of neurons firing on a particular time-step and propagated spikes by updating the activation level of all neurons to which they are connected. This contrasts radically with most Artificial Neural Network implementations that recalculate the activity of every neuron in the system on every time step, using floating point values for both the activation level and the weights associated with each input, computations that are very costly.

In this section, we list the novel features that we have adopted in our network that make it possible for us to simulate activity in networks with over four billion spiking neurons on a simple MacBook Air equipped with an M2 chip, 24 GB of on-chip unified memory, an internal 2TB solid state disk, and additional external storage. The code was implemented using Swift and Metal, Apple's GPU-accelerated graphics and compute framework.

A. Zero-less Binary Weights

One of the key features that add to the memory efficiency of our simulation is using sparse binary weights instead of floating point ones and furthermore, having a zero-less compression of the weight matrix to make its memory footprint even smaller.

We achieve this compression by only storing the addresses or IDs of the pre-synaptic neurons having a non-zero connection with every post-synaptic neuron in our network. Thus, effectively discarding all the zero weights from our network, resulting in a zero-less weight matrix.

Each post-synaptic neuron has a set of C binary input connections from other neurons in the network. The current implementation uses C values of up to 256, but with more external memory, it would be trivial to increase this number. The connections are initially stored in a set of backward connection files as lists of 32-bit numbers that specify the addresses of the pre-synaptic neurons. These backward connection files are then used to generate a second set of files containing the forward connections for each neuron. On average, neurons have C forward connections, although this will vary.

The backward connections are the equivalent of the receptive field of every post-synaptic neuron, while the forward connections are the equivalent of the emitting field of the pre-synaptic neurons [20].

B. Propagation based simulation

In our GPU implementations, we employ an event-driven strategy for propagating incoming spikes akin to the SpikeNet simulator. We have employed a temporal coding based scheme with only one spike per neuron as it is a more effective spike coding scheme compared to rate coding [3] and is more hardware efficient in its implementation [25].

The simulation starts by activating a particular set of N neurons, which can be chosen randomly or used to represent sensory events. The simulator reads the list of post-synaptic targets for each neuron from the forward connection files and uses GPU threads to increment the activation levels of their targets. It also updates a histogram of the activation levels, which is used to select the N neurons with the highest activation levels. These neurons are subsequently used for the next processing cycle, and their activation levels are reset to zero accordingly.

C. Oscillations, N-of-M Coding and the Histogram Method

A major innovation is the fact that neurons do not have a predefined threshold. Instead, by generating a histogram of activation values at the end of each propagation cycle, we can pick off the N most active neurons and guarantee that activity can continue indefinitely. In conventional spiking neuron simulators, there is a high risk that activity will die out in a few cycles, or that an avalanche effect makes too many neurons spike. While this trick of having spiking thresholds determined on the fly may seem un-biological, it is possible that oscillations could be nature's way of keeping the number of active neurons strictly under control. It is a way to implement N of M coding [10] that could be extremely useful, as well as allowing the implementation of synfire chains [2] .

To achieve this, our simulator maintains a histogram of the activation levels for all M neurons in the system and updates the histogram during the propagation phase using GPU threads. Once propagation is complete, we can determine threshold values that allow a predetermined number of neurons to fire, and this is used to generate the new list during a second pass through the list of targets. Importantly, the time taken to implement this phase is independent of M - the number of neurons in the system. M can be 2^{32} (as in the current simulations) but could be even larger, allowing networks with more neurons than the entire human brain to be implemented.

Effectively, this histogram method is just a computational trick. But, interestingly, it is functionally equivalent to an oscillation-based system that progressively depolarizes the entire population of neurons, leading them to fire in turn according to their underlying level of activation. The system would need an additional inhibitory circuit that effectively counts the number of neurons that have fired and blocks the excitatory ramp signal as soon as a given number of neurons has fired. This temporal coding trick for performing a k-Winner Take All (k-WTA) operation was originally proposed in [26] but also discussed more recently [9].

While it would be possible to implement this sort of oscillatory mechanism in a spiking neural network simulator, it would be computationally extremely expensive, especially when M, the number of neurons in the population, is large. It would involve modifying the activation level of every neuron over and over again. However, the histogram-based method described here works well irrespective of the number of neurons in the simulation.

III. Implementation Overview

In this section, we outline the implementation details that determine the life cycle of neurons in a typical Terabrain architecture. On the arrival of incoming spikes, neurons in our architecture go through three main phases, namely propagation, reset and leak phase, resembling biological neurons. Furthermore, we used a p-bit quantised neuronal activation for reduced memory footprint, which means the highest activity in the system can be $2^p - 1$, where p can be 4, 8 or higher depending on the requirements. For the sake of clarity, we provide the pseudo-code for our algorithm in the sub-section below.

A. Phases in Terabrain architecture

The propagation phase involves an event-based increment of neuronal activation for every neuron present in the forward connection list of the N excited neurons, while the rest remains untouched. In this phase, we also update our activation histogram to keep track of the suitable threshold (τ_s) for selecting N out of M neurons to spike, in each cycle.

Algorithm 1 Propagation cycle

```
Require: Contiguous list of target neuron IDs, denoted by actForwrdConnx, in the emitting field of the N excited neurons out of M
Require: p-bit Counter array for neuronal activation, denoted by actArr and the initialised histogram of the p-bit activation, histoList for all M neurons
  function PROPAGATE SPIKES
      for targNeur in actForwrdConnx do          ▷ GPU threaded
          prevCount = actArr[targNeur]
          actArr[targNeur] += 1                  ▷ Incrementing activation
          histoList[prevCount] += 1              ▷ Update histogram
          histoList[prevCount-1] -= 1
  function CHOOSE SPIKE THRESHOLD
      pop = 0                                    ▷ Population counter for N of M
      index = len(histoList)
      while $pop < N$ do                         ▷ Highest activity population of N
          pop += histoList[index-1]
          index -= 1
      $thresh \leftarrow index$                  ▷ Spiking Threshold on the fly
      return thresh
```

The reset phase involves selecting the addresses or IDs of neurons that will be part of the N most highly activated neurons to be propagated along-with the external spikes for the next cycle and resetting their activation back to zero subsequently.

To optimise the reset phase in a way that it remains unaffected by the actual size of the network, we dispatch GPU threads to only go through the list of target neurons in the current cycle, select the N-of-M most active neurons, and reset their activation. In this way, we save the computational cost of going through all the neurons in our network, which can be extremely large.

Algorithm 2 Spike List generation and reset cycle

```
Require: Empty list of outgoing spikes, spikeList which stores the IDs of outgoing spikes
Require: Spiking threshold returned from the previous function, thresh
  function GENERATE SPIKE LIST
      for targNeur in actForwrdConnx do          ▷ GPU threaded
          count = actArr[targNeur]
          if $count \geq thresh$ and $\mathrm{len}(spikeList) < N$ then
              spikeList.append(targNeur)
              actArr[targNeur] = 0               ▷ Reset activation
              histoList[count] -= 1              ▷ Update histogram
              histoList[0] += 1
```

The leak phase requires us to go through the activity of all the neurons in our simulation and perform different leak modes on them akin to the functioning of biological LIF neurons.

Algorithm 3 Neuron leak cycle

```
Require: p-bit Counter array for neuronal activation, denoted by actArr
Require: Wipe mode, w for different leaky decays of the neuronal activation with time and wipe threshold wipeTh for special mode $w = 3$
  function NEURON LEAK FUNCTION
      for neuron in M do                         ▷ GPU threaded
          if (actArr[neuron] > 0) and (w == 1) then
              actArr[neuron] = 0                 ▷ Hard reset
          if (actArr[neuron] > 0) and (w == 2) then
              actArr[neuron] -= 1                ▷ Gradual decrement
          if (0 < actArr[neuron] < wipeTh) and (w == 3) then
              actArr[neuron] = 0                 ▷ Hard reset for low activity
```

B. In-Core vs. Out-of-Core

In principle, our architecture has no real limitation in terms of scaling. However, simulating networks of considerable size presents a challenge in storing the binary connection matrix, even after applying the zero-less compression technique. Despite this, we identify two feasible strategies for storing synaptic connections in our network, each with distinct advantages and limitations.

The first approach involves utilizing the 24 GB unified shared RAM of the M2 chip for storage, resulting in an *In-core* version of our architecture. While this method offers higher I/O bandwidth, facilitating rapid read/write operations, it is constrained by the RAM's limited storage capacity, which may become saturated during extremely large simulations.

Alternatively, the second strategy involves storing synaptic connections on external Solid State Drives (SSDs). This Out-of-Core approach allows for the simulation of substantially larger architectures without memory constraints. The primary challenge here is the lower I/O bandwidth of external SSDs, which slows down read/write speeds. To mitigate this, we employed multi-threaded reading using Swift's Grand Central Dispatch. This enables concurrent read/write operations, leveraging the 10-core CPU of the M2 chip to counteract the reduced I/O speed.

IV. Results

A. Scalability of simulations

The first important result of the present study is the demonstration that when the simulation is propagating packets with N spikes that each have C binary connections, the time taken to run each step increases roughly linearly [1] with the number of spikes in each packet.

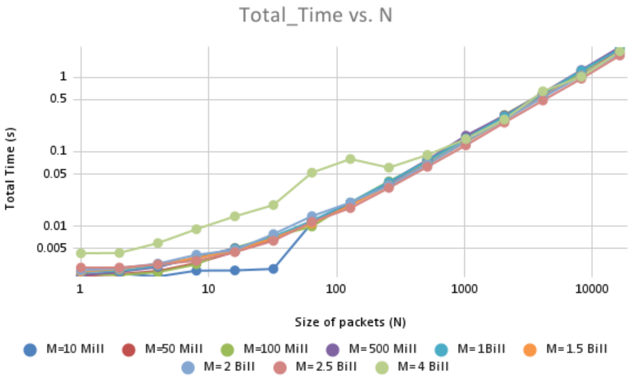

Fig. 1: Log-Log scaled graph of total time versus the packet of spike size (N), with C = 128 binary connections on Apple M1 chip with 16 GBs of shared RAM

The second important result is the demonstration that the time taken to propagate each packet is effectively invariant with the size of the network, making it feasible to simulate networks with billions of neurons. The only real constraint is the amount of external storage space available. For the current simulations, we equipped the basic Mac with two additional 4TB SSDs (Solid-State-Disks) that could each be used to store up to 1 trillion connections (because each connection requires 4 bytes (32 bits) of storage).

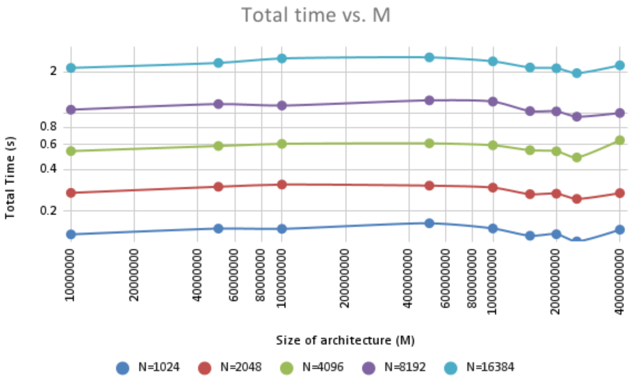

Fig. 2: Log-Log scaled graph of total time versus the architecture size (M), with C = 128 binary connections on Apple M1 chip with 16 GBs of shared RAM

B. Optimum thread size

We systematically evaluated various thread grid configurations to determine the optimal thread count for parallel spike propagation. Our findings indicate that a configuration of 4 or 8 thread blocks, each consisting of 1024 concurrent threads, is adept at efficiently propagating large batches of spikes. This efficiency correlates with the maximal concurrent thread processing capacity of the M2 chip.

The following section includes a comparative performance analysis graph, illustrating the efficacy of our architecture across different thread grid dimensions:

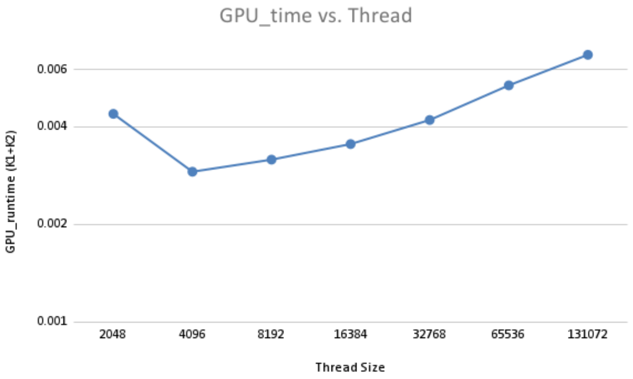

Fig. 3: Variation of GPU compute time for an architecture with 4 Billion neurons, according to the number of threads dispatched

V. Discussions

A. Comparing the In-Core and Out-of-Core versions

As we discussed in section III-B , the main limitation with the *out-of-core* version will be the speed with which data can be read from the external file. Read speeds on currently available SSD technologies can exceed 6 GB/second, but this is for sequential reads. In our case, the main limit corresponds to the maximum number of random reads and writes (IOPS), currently around 1 million. Note that each operation reads in 4KB of data, meaning that each neuron that spikes could effectively have up to 1K targets.

Such simulations will work much faster when the connection lists are loaded directly into RAM as in the *in-core* version. Indeed, we have noticed that throughput can be considerably increased using this strategy. The In-Core version is roughly 30 times faster than the out-of-core version. However, while systems such as the Mac Studio Ultra can already have up to 128 GB of on-chip unified memory, and there are rumours of future systems with up to 384 GB, such systems will remain very expensive. For this reason, we are concentrating on what can be achieved with more affordable systems.

[1] We can infer from figure 1, that the trend is perfectly linear in Log-Log scaled graph for N >= 100, i.e. for bigger packet sizes

B. *Visualisation*

The simulator also displays the list of spiking neurons in a rotating cube, for example with dimensions of 1000 x 1000 x 1000 for visualising a billion neurons in real time with the simulation. A video of the simulator can be found on the github link provided here.

C. *Propagating spikes in the Terabrain*

As we discussed already in section III about the implementational details, here we provide a schematic of the propagation of spikes in a typical Terabrain architecture.

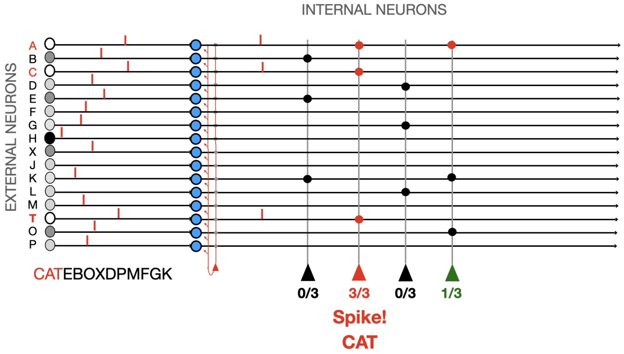

Fig. 4: Simple terabrain architecture with M = 16 neurons and N = 3, for detecting CAT pattern in text input sequence

The input stimulus in this case is a text sequence, and the neuron which is selective to respond to the "CAT" stimulus has connections with the corresponding input pre-synaptic neurons. There is a winner-take-all inhibition circuit that only lets the $N = 3$ shortest latency spikes to be propagated in a sparse and zero-less fashion to make the "CAT" neuron reach the spiking threshold, which is $\tau_s = 3$ in this case.

VI. Perspectives

The results presented here are primarily aimed at demonstrating the feasibility of simulating very large networks of spiking neurons without the need for any specialized hardware. The basic simulations use patterns of connections that are initially completely random, so the system does not perform any useful function. To perform a specific task using a Terabrain architecture, the connections would have to be specified using some form of learning procedure. We already have some well-defined ideas for how this sort of learning could be done. Essentially, we have developed unsupervised learning schemes that effectively swap the inputs to a given neuron to match patterns of firing that tend to repeat, similar to the JAST Learning algorithm that was patented by Simon Thorpe et. al, which can be found on this link.

One might wonder why one would ever require so many neurons. Suppose that we used text to develop the equivalent of a large language model, with neurons selective to all the words in the language and connections that would allow semantic knowledge to be stored in a semantic network. In such a case, you might think it would be enough to have one neuron for each word and use floating point connections to link them together, with strengths that depend on the degree of association. However, it is important to note that we do not anticipate implementing a mechanism that could prevent multiple copies of the same neuron from developing. In this case, the very large number of neurons available could allow the frequency of a given word to be represented by the number of selective neurons. High-frequency words would naturally have more copies than rare ones, making them more likely to be included in the N neurons active on the next processing cycle. Effectively, this means that the Bayesian prior for a particular stimulus could be encoded by the number of selective neurons.

At this stage, it is totally unclear what would be a suitable value for N. The most useful value might well depend on the type of data being processed and the task at hand. One additional advantage of the current simulation system is that N can be chosen at will and continuously varied if needed. The simple fact is that keeping N low provides massive advantages in terms of computational load. Keeping N very low could be the secret of the human brain's remarkable power efficiency.

Moreover, having lower values of N allows us to have a more controllable system which could be essential for having an explainable system. In other words, since N can be as low as we like, we can isolate the minimal subset of features that cause the inference, thus resulting in a fully explainable white box in contrast to generic neural networks in literature.

VII. Conclusions

In conclusion, we have shown that today's computer hardware can already simulate extremely large spiking neural networks with billions of neurons and potentially trillions of binary connections.

Related Neuromorphic approaches include the SpiNNaker project developed by Steve Furber's group [11], [12] and the recently announced DeepSouth project [18], [22]. But these systems typically assume that neurons are firing at 1 or 2 spikes a second. Under such conditions, simulating systems the size of the human neocortex would require propagating billions of spikes every second, and this requires very specialised hardware for real time processing. One of the key innovations of the present work is the decision to only allow N spikes on each processing cycle. With modest values for N, this allows extremely large networks to be simulated without the need for specialized hardware.

Remarkably, our simulations were possible using an energy budget of around 37W, less than twice of the human brain's consumption. However, this requires radical changes to the conventional processing strategies for simulating neural networks. Some of those ideas could be useful for other less conventional approaches to implementing neural networks. A critical challenge will be to see whether such ultra-sparse systems are capable of performing advanced tasks such as language processing and reasoning. Our ongoing work is looking at how to implement learning algorithms that allow neurons to become selective to repeat activity patterns.

References

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

[2] Moshe Abeles. Synfire chains. *Scholarpedia*, 4(7):1441, 2009.

[3] Lina Bonilla, Jacques Gautrais, Simon Thorpe, and Timothée Masquelier. Analyzing time-to-first-spike coding schemes: A theoretical approach. *Frontiers in Neuroscience*, 16, 2022.

[4] Keunwoo Choi, György Fazekas, and Mark B. Sandler. Automatic tagging using deep convolutional neural networks. *CoRR*, abs/1606.00298, 2016.

[5] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2012.

[6] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery.

[7] A. Delorme, J. Gautrais, R. Van Rullen, and S. Thorpe. Spikenet: A simulator for modeling large networks of integrate and fire neurons. *Neurocomputing*, 26-27:989–996, 1999.

[8] Arnaud Delorme and Simon J Thorpe. Spikenet: an event-driven simulation package for modelling large networks of spiking neurons. *Network: Computation in Neural Systems*, 14(4):613–627, 2003. PMID: 14653495.

[9] Pascal Fries, Danko Nikolić, and Wolf Singer. The gamma cycle. *Trends in neurosciences*, 30(7):309–316, 2007.

[10] Steve Furber, W Bainbridge, J Cumpstey, and Steve Temple. Sparse distributed memory using n-of-m codes. *Neural networks : the official journal of the International Neural Network Society*, 17:1437–51, 12 2004.

[11] Michael Hopkins, Jakub Fil, Edward George Jones, and Steve Furber. Bitbrain and sparse binary coincidence (sbc) memories: Fast, robust learning and inference for neuromorphic architectures. *Frontiers in Neuroinformatics*, 17, 2023.

[12] Michael Hopkins, Garibaldi García, Petruț Bogdan, and Steve Furber. Spiking neural networks for computer vision. *Interface Focus*, 8:20180007, 06 2018.

[13] Doo Seok Jeong, Kyung Min Kim, Sungho Kim, Byung Joon Choi, and Cheol Seong Hwang. Memristors for energy-efficient new computing paradigms. *Advanced Electronic Materials*, 2(9):1600090, 2016.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[15] Shahar Kvatinsky, Misbah Ramadan, Eby G. Friedman, and Avinoam Kolodny. Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, 2015.

[16] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, Wenhao Song, John Paul Strachan, Mark Barnell, Qing Wu, R. Stanley Williams, J. Joshua Yang, and Qiangfei Xia. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications*, 9(1):2385, 2018.

[17] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Convolutional neural networks for large-scale remote-sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):645–657, 2017.

[18] Ali Mehrabi, Yeshwanth Bethi, André van Schaik, and Saeed Afshar. An optimized multi-layer spiking neural network implementation in fpga without multipliers. *Procedia Comput. Sci.*, 222(C):407–414, jan 2023.

[19] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016.

[20] Laurent U Perrinet. Accurate detection of spiking motifs by learning heterogeneous delays of a spiking neural network. In *32nd International Conference on Artificial Neural Networks (ICANN 2023)- Special Session on Recent Advances in Spiking Neural Networks*, 2023.

[21] Karol J. Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2015.

[22] Western Sydney Press. World first supercomputer capable of brain-scale simulation being built at Western Sydney University. dec 2023.

[23] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449, 2017.

[24] Neha Sharma, Vibhor Jain, and Anju Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018. International Conference on Computational Intelligence and Data Science.

[25] Simon Thorpe. *Timing, Spikes and the Brain*. World Scientific, 2023.

[26] S.J. Thorpe. *Spike arrival times : A highly efficient coding scheme for neural networks*, pages 91–94. North-Holland Elsevier, 1990.

[27] Lonce Wyse. Audio spectrogram representations for processing with convolutional neural networks. *CoRR*, abs/1706.09559, 2017.