A Stochastic-Computing based Deep Learning Framework using Adiabatic Quantum-Flux-Parametron Superconducting Technology

Ruizhe Cai
Ao Ren
Northeastern University
USA
{cai.ruiz,ren.ao}@husky.neu.edu

Olivia Chen
Yokohama National University
Japan
chen-olivia-pg@ynu.ac.jp

Ning Liu
Caiwen Ding
Northeastern University
USA
{liu.ning,ding.ca}@husky.neu.edu

Xuehai Qian
University of Southern California
USA
xuehai.qian@usc.edu

Jie Han
University of Alberta
Canada
jhan8@ualberta.ca

Wenhui Luo
Yokohama National University
Japan
luo-wenhui-xn@ynu.ac.jp

Nobuyuki Yoshikawa
Yokohama National University
Japan
nyoshi@ynu.ac.jp

Yanzhi Wang
Northeastern University
USA
yanz.wang@northeastern.edu

ABSTRACT

The *Adiabatic Quantum-Flux-Parametron* (AQFP) superconducting technology has been recently developed, which achieves the highest energy efficiency among superconducting logic families, potentially 10^4-10^5 gain compared with state-of-the-art CMOS. In 2016, the successful fabrication and testing of AQFP-based circuits with the scale of 83,000 JJs have demonstrated the scalability and potential of implementing large-scale systems using AQFP. As a result, it will be promising for AQFP in high-performance computing and deep space applications, with Deep Neural Network (DNN) inference acceleration as an important example.

Besides ultra-high energy efficiency, AQFP exhibits two unique characteristics: the deep pipelining nature since each AQFP logic gate is connected with an AC clock signal, which increases the difficulty to avoid RAW hazards; the second is the unique opportunity of true random number generation (RNG) using a single AQFP buffer, far more efficient than RNG in CMOS. We point out that these two characteristics make AQFP especially compatible with the *stochastic computing* (SC) technique, which uses a time-independent bit sequence for value representation, and is compatible with the deep pipelining nature. Further, the application of SC has been investigated in DNNs in prior work, and the suitability has been illustrated as SC is more compatible with approximate computations.

This work is the first to develop an SC-based DNN acceleration framework using AQFP technology. The deep-pipelining nature of AQFP circuits translates into the difficulty in designing accumulators/counters in AQFP, which makes the prior design in SC-based DNN not suitable. We overcome this limitation taking into account different properties in CONV and FC layers: (i) the inner product calculation in FC layers has more number of inputs than that in CONV layers; (ii) accurate activation function is critical in CONV rather than FC layers. Based on these observations, we propose (i) accurate integration of summation and activation function in CONV layers using bitonic sorting network and feedback loop, and (ii) low-complexity categorization block for FC layers based on chain of majority gates. For complete design we also develop (i) ultra-efficient stochastic number generator

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6669-4/19/06…$15.00
https://doi.org/10.1145/3307650.3322270

in AQFP, (ii) a high-accuracy sub-sampling (pooling) block in AQFP, and (iii) majority synthesis for further performance improvement and automatic buffer/splitter insertion for requirement of AQFP circuits. Experimental results suggest that the proposed SC-based DNN using AQFP can achieve up to 6.8×10^4 times higher energy efficiency compared to CMOS-based implementation while maintaining 96% accuracy on the MNIST dataset.

KEYWORDS

Stochastic Computing, Deep Learning, Adiabatic Quantum-Flux-Parametron, Superconducting

ACM Reference Format:

Ruizhe Cai, Ao Ren, Olivia Chen, Ning Liu, Caiwen Ding, Xuehai Qian, Jie Han, Wenhui Luo, Nobuyuki Yoshikawa, and Yanzhi Wang. 2019. A Stochastic-Computing based Deep Learning Framework using Adiabatic Quantum-Flux-Parametron Superconducting Technology. In *The 46th Annual International Symposium on Computer Architecture (ISCA '19), June 22–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307650.3322270

1 INTRODUCTION

Wide-ranging applications of deep neural networks (DNNs) in image classification, computer vision, autonomous driving, embedded and IoT systems, etc., call for high-performance and energy-efficient implementation of the inference phase of DNNs. To simultaneously achieve high performance and energy efficiency, *hardware acceleration of DNNs*, including FPGA- and ASIC-based implementations, has been extensively investigated [4, 6–9, 12, 13, 17–20, 28, 29, 33, 34, 37, 39–41, 48, 49, 51, 53–55]. However, most of these designs are CMOS based, and suffer from a performance limitation because the Moore's Law is reaching its end.

Being widely-known for low energy dissipation and ultra-fast switching speed, Josephson Junction (JJ) based superconductor logic families have been proposed and implemented to process analog and digital signals [24]. It has been perceived to be an important candidate to replace state-of-the-art CMOS due to the superior potential in operation speed and energy efficiency, as recognized by the U.S. IARPA C3 and SuperTools Programs and Japan MEXT-JSPS Project. Adiabatic quantum-flux-parametron (AQFP) logic is an energy-efficient superconductor logic family based on the quantum-flux-parametron (QFP)[26]. AQFP logic achieves high energy efficiency by adopting adiabatic switching [23], in which the potential energy profile evolves from a single well to a double well so that the logic state can change quasi-statically. The energy-delay-product (EDP) of the AQFP circuits fabricated using processes such as the AIST standard process 2 (STP2) [30] and the MIT-LL SFQ process [47], is only three orders of magnitude larger than the quantum limit [45]. It can potentially achieve 10^4-10^5 energy efficiency gain compared with

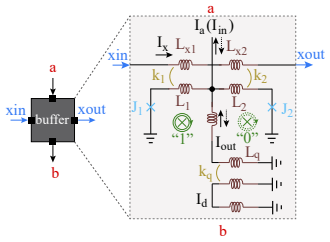

Figure 1: Junction level schematic of an AQFP buffer.

state-of-the-art CMOS (even two order of magnitude energy efficiency gain when cooling energy is accounted for), with a clock frequency of several GHz. Recently, the successful fabrication and testing of AQFP-based implementations with the scale of 83,000 JJs, have demonstrated the scalability and potential of implementing large-scale circuits/systems using AQFP [31]. As a result, it will be promising for the AQFP technology in high-performance computing, with DNN inference acceleration an important example.

The AQFP technology uses AC bias/excitation currents as both multi-phase clock signal and power supply [43] to mitigate the power consumption overhead of DC bias in other superconducting logic technologies. Besides the ultra-high energy efficiency, AQFP exhibits two unique characteristics. The first is the deep pipelining nature since each AQFP logic gate is connected with an AC clock signal and occupies one clock phase, which increases the difficulty to avoid RAW (Read after Write) hazards in conventional binary computing. The second is the unique opportunity of true random number generation (RNG) using a single AQFP buffer (double JJ) in AQFP [16, 44], which is far more efficient than RNG in CMOS.

We point out that these two characteristics make AQFP technology especially compatible with the *stochastic computing* (SC) technique [14][1], which allows the implementation of 64 basic operations using extremely small hardware footprint. SC uses a time-independent bit sequence for value representation, and lacks RAW dependency among the bits in the stream. As a result it is compatible with the deep-pipelining nature of superconducting logic. Furthermore, one important limiting factor of SC, i.e., the overhead of RNG [27][32], can be largely mitigated in AQFP.

SC is inherently an approximate computing technique [15][52], and there have been disputes about the suitability of SC for precise computing applications [3]. On the other hand, the DNN inference engine is essentially an approximate computing application. This is because the final classification result depends on the relative score/logit values of different classes, instead of absolute values. Recent work [35][3] have pointed out the suitability of SC for DNN acceleration, and [50] has further proved the equivalence between SC-based DNN and *binary neural networks*, where

the latter originate from deep learning society [11]. All the above discussions suggest the potential to build SC-based DNN acceleration using AQFP technology.

This paper is the first to develop an SC-based DNN acceleration framework using AQFP superconducting technology. We adopt bipolar format in SC because weights and inputs can be positive or negative, and build stochastic number generation block in AQFP with ultra-high efficiency. The deep-pipelining nature of AQFP circuits translates into the difficulty in designing accumulators/counters in AQFP, which makes the prior design in SC-based DNN [35] not suitable. We overcome this limitation taking into account different properties in CONV and FC layers.In summary, the contributions are listed as follows: (i) ultra-efficient stochastic number generator in AQFP; (ii) integration of summation and activation function using bitonic sorting in CONV layers; (iii) a high-accuracy sub-sampling block in AQFP; (iv) low-complexity categorization block for FC layers; (v) majority synthesis for further performance improvement and automatic buffer/splitter insertion for AQFP requirement. Experimental results suggest that the proposed SC-based DNN using AQFP can achieve up to 6.9×10^4 higher energy efficiency compared to its CMOS implemented counterpart and 96% accuracy on the MNIST dataset [21]. The proposed blocks can be up to 7.76×10^5 times more energy efficient than CMOS implementation. Simulation results suggest that the proposed sorter-based DNN blocks can achieve extremely low inaccuracy. In addition, we have successfully verified the functionality of a feature extraction chip, which is fabricated using the AIST $10kA/cm^2$ Niobium high-speed standard process (HSTP), embedded in a cryoprobe and inserted into a liquid Helium Dewar to cool down to 4.2K.

2 BACKGROUND

2.1 AQFP Superconducting Logic

As shown in Fig. 1, the basic logic structure of AQFP circuits is buffers consisting of a double-Josephson-Junction SQUID [10]. An AQFP logic gate is mainly driven by AC-power, which serves as both excitation current and power supply. By applying excitation current I_x, typically in the order of hundreds of micro-amperes, excitation fluxes are applied to the superconducting loops via inductors L_1, L_2, L_{x1} and L_{x2}. Depending on the small input current I_{in}, either the left or the right loop stores one single flux quantum. Consequently, the device is capable of acting as a *buffer cell* as the logic state can be represented by the direction of the output current I_{out}. The storage position of the quantum flux in the left or the right loop can be encoded as logic '1' or '0'.

The AQFP inverter and constant cell are designed from AQFP buffer. The AQFP inverter is implemented by negating the coupling coefficient of the output transformer in the

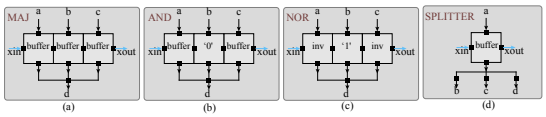

Figure 2: Example of AQFP logic gates.

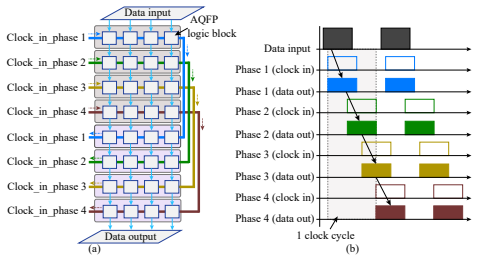

Figure 3: (a). Four phase clocking scheme for AQFP circuits; (b). Data propagation between neighbouring clock phases.

AQFP buffer, while the constant gate in AQFP is designed using asymmetry excitation flux inductance in the AQFP buffer. The AQFP splitter is also implemented based on the AQFP buffer as shown in Fig. 2. Please note that, unlike CMOS gates, which are connected to fan-out directly, all AQFP gates need to be connected to splitters for fan-out.

The AQFP standard cell library is built via the *minimalist design approach* [46], in other words, designing more complicated gates using a bottom-up manner. Logic gate shown in Fig. 2 (a) is a majority gate as the output, 'd', depending on the number of '1's at inputs, 'a' to 'c'. Replacing any buffer with inverter can produce a variation of the majority gate (such as the minority gate). As illustrated in Fig. 2 (b)-(c), an AQFP AND gate is implemented by two buffers and one constant '0' while an AQFP NOR gate is implemented by two inverters and one constant '1'. Unlike the conventional CMOS technology, both combinational and sequential AQFP logic cells are driven by AC-power. In addition, the AC power serves as clock signal to synchronize the outputs of all gates in the same clock phase. Consequently, data propagation in AQFP circuits requires overlapping of clock signals from neighbouring phases. Fig. 3 presents an example of typical clocking scheme of AQFP circuits and data flow between neighbouring clock phases. In this clocking scheme, each AQFP logic gate is connected with an AC clock signal and occupies one clock phase, which makes AQFP circuits "deep-pipelining" in nature. Such clock-based synchronization characteristic also requires that all inputs for each gate should have the same delay (clock phases) from the primary inputs.

2.2 Stochastic Computing

Stochastic computing (SC) is a paradigm that represents a number, named *stochastic number*, by counting the number of ones in a bit-stream. For example, the bit-stream 0100110100 contains four ones in a ten-bit stream, thus it represents

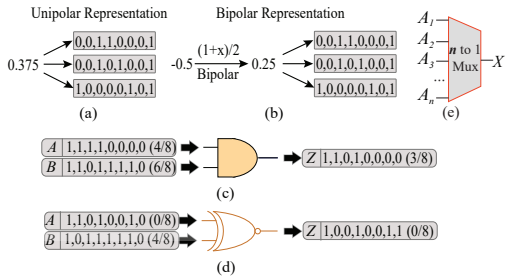

Figure 4: (a) Unipolar encoding and (b) bipolar encoding. (c) AND gate for unipolar multiplication. (d) XNOR gate for bipolar multiplication. (e) MUX gate for addition.

$x = P(X = 1) = 4/10 = 0.4$. In the bit-stream, each bit is independent and identically distributed (i.i.d.) which can be generated in hardware using stochastic number generators (SNGs). Obviously, the length of the bit-streams can significantly affect the calculation accuracy in SC [36]. In addition to this unipolar encoding format, SC can also represent numbers in the range of $[-1, 1]$ using the bipolar encoding format. In this scenario, a real number x is processed by $P(X = 1) = (x + 1)/2$. Thus 0.4 can be represented by 1011011101, as $P(X = 1) = (0.4 + 1)/2 = 7/10$. -0.5 can be represented by 10010000, as it shown in Figure 4(b), with $P(X = 1) = (-0.5 + 1)/2 = 2/8$.

Compared to conventional computing, the major advantage of stochastic computing is the significantly lower hardware cost for a large category of arithmetic calculations. A summary of the basic computing components in SC, such as multiplication and addition, is shown in Figure 4. As an illustrative example, a unipolar multiplication can be performed by a single AND gate since $P(A \cdot B = 1) = P(A = 1)P(B = 1)$, and a bipolar multiplication is performed by a single XNOR gate since $c = 2P(C = 1) - 1 = 2(P(A = 1)P(B = 1) + P(A = 0)P(B = 0)) - 1 = (2P(A = 1) - 1)(2P(B = 1) - 1) = ab$.

Besides multiplications and additions, SC-based activation functions are also developed [35][5]. As a result, SC has become an interesting and promising approach to implement DNNs [38][3][22] with high performance/energy efficiency and minor accuracy degradation.

3 MOTIVATION AND CHALLENGES OF THE PROPOSED WORK

One observation we make is that SC technique, together with extremely small hardware footprint compared with binary computing, is especially compatible with the deep-pipelining nature in AQFP circuits. This is because SC uses a time-independent bit sequence for value representation, and there are no data dependency in a number of consecutive clock cycles (equal to the bit-stream length, ranging from 128 to 2,048). In this way the RAW hazards can be avoided without

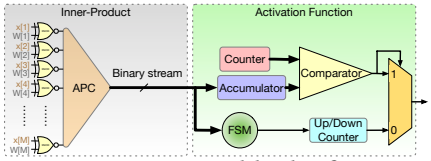

Figure 5: Feature extraction block of CMOS-based SC-based DNNs implementation in prior work.

bubble insertions, which is not avoidable in conventional binary computing implemented using AQFP. Additionally, it is important to note that the overhead of RNG, which is a significant overhead in SC using CMOS logic (even accounts for 40%-60% hardware footprint), can be largely mitigated in AQFP. This is because of extremely high efficiency in true RNG implementation (instead of pseudo-RNG) in AQFP, thanks to its unique operating nature.

As illustrated in Fig. 7 (a), a 1-bit true RNG can be implemented using the equivalent circuit of an AQFP buffer. The direction of the output current I_{out} represents the logic state 1 or 0. I_{out} will be determined by the input current I_{in}. However, when $I_{in} = 0$, the output current I_{out} will be randomly 0 or 1 depending on the thermal noise, as shown in Fig. 7 (b). As a result, an on-chip RNG is achieved with two JJs in AQFP, and an independent random bit (0 or 1) will be generated in each clock cycle.

In a nutshell, we forecast SC to become a competitive technique for AQFP logic on a wide set of applications. For the acceleration of DNN inference phase in particular, SC has been demonstrated as a competitive technique to reduce hardware footprint while limiting accuracy loss [2][35]. Compared with precise computing applications [3], SC is more suitable for approximate computing where DNN is an example. Moreover, it has been proved recently on the equivalence between SC-based DNN and binary neural networks (BNN) [50]. As the latter originates from the deep learning society [11] and many accuracy enhancement techniques have been developed, these advances can be migrated to enhance the accuracy in SC-based DNNs to be close to the software-based, floating point accuracy levels.

Despite the above advantages in SC-based DNNs using AQFP, certain challenges need to be overcome to realize an efficient implementation. One major challenge is the difficulty to implement accumulators/counters due to the super-deep pipelining nature. This is because one single addition takes multiple clock phases, say n, in the deep-pipelining structure. Then the accumulation operation can only be executed once in every n clock phases to avoid RAW hazards. Throughput degradation will be resulted in especially when a large number is allowed in the accumulator (in the case of baseline structure of SC-based DNNs[35] as we shall see later). Similarly, it is also challenging for efficient implementation of finite state machines (FSM) in AQFP.

4 PROPOSED SC-BASED DNN FRAMEWORK USING AQFP

4.1 System Architecture: Difference from the Prior Arts

Fig. 5 demonstrates an M-input *feature extraction block* as defined in the prior work [35] of SC-based DNNs (built in CMOS), as a basic building block in CONV layers for feature extraction and FC layers for categorization. This structure performs inner product and activation calculation in the SC domain. The whole SC-based DNN chip comprises parallel and/or cascaded connection of such structures: the former for computing multiple outputs within a DNN layer, while the latter for cascaded DNN layers.

The functionality in the feature extraction block is to implement $y_i = \psi(\mathbf{w}_i^T \mathbf{x} + b_i)$, where $\mathbf{x}$ and $\mathbf{w}_i$ are input vector and one weight vector, respectively; b_i is a bias value; and ψ is the activation function. In CMOS-based implementation of feature extraction block [35], multiplication in the inner product calculation is implemented using XNOR gates (both weights and inputs are bipolar stochastic numbers); summation is implemented using *approximate parallel counter* (APC) for higher accuracy or using mux-tree (adder-tree) for low hardware footprint; activation is implemented using binary counter (for APC output) or FSM (for mux-tree output).

The aforesaid challenge in accumulator/counter and FSM implementations in AQFP translates into the difficulty in the direct, AQFP-based implementation of the above feature extraction block. More specifically, the binary counter or FSM that are necessary for activation function implementation can only be executed once in multiple clock cycles to avoid RAW hazards, resulting in throughput degradation. This limitation needs to be overcome in our proposed SC-based DNN architecture using AQFP.

Fig. 6 demonstrates our proposed SC-based DNN architecture using AQFP, which uses different implementation structures of inner product/activation for CONV layers and

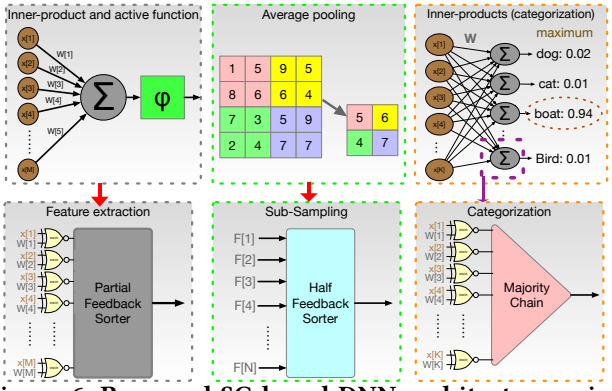

Figure 6: Proposed SC-based DNN architecture using AQFP.

FC layers. For CONV layers, the number of inputs for inner product computation is less compared with that for FC layers. Meanwhile, the requirement of accumulation operation is more significant (in terms of effect on overall accuracy) in the CONV layers compared with FC layers. Based on these observations, we propose (i) accurate integration of summation and activation function in CONV layers using bitonic sorting network and feedback loop, and (ii) low-complexity categorization block for FC layers based on chain of majority gates. The former is more accurate and more complicated, which is compatible with CONV layers which have higher impact on overall accuracy but with smaller number of inputs. The latter has low complexity, and is compatible with FC layers which has lower impact on overall accuracy but with large number of inputs. The overall accuracy loss will be minor and under control as shall be shown in the experiments.

In the following subsections, we will describe in details in the proposed feature extraction block for CONV layers and categorization block for FC layers. Despite the distinct structure, both blocks essentially implement the inner product and activation $y_i = \psi(\mathbf{w}_i^T \mathbf{x} + b_i)$ as discussed before, and multiplication is implemented in XNOR in SC domain. The inputs of both blocks are from previous DNN layer or from primary inputs, and are stochastic numbers in principle. The weights are stored (hardwired) in the AQFP chip in binary format for reducing hardware footprint, and will be transformed to stochastic format through RNG and comparators. In the following, we also describe (i) the stochastic number generator that can be implemented with ultra-high efficiency in AQFP, and (ii) accurate sub-sampling block in AQFP for pooling operation, which has higher accuracy than the version in [35].

A *stochastic number generator* (SNG) is a key part in the SC scheme, which converts a binary number to its corresponding stochastic format. This is achieved via comparing between the binary number (to convert) with a stream of random numbers [15][1], and a uniformly distributed RNG is needed for this conversion process. In AQFP, an n-bit true RNG can be implemented using n 1-bit true RNGs, whose layout is shown in Fig. 9, each using two JJs for efficient implementation as discussed in the previous section. An SNG can be implemented using the n-bit true RNG and n-bit comparator, where n is the number of bits of the source binary number.

For more efficient hardware utilization, we propose a true random number generator matrix, in which each true RNG unit can be exploited in more than one SNGs. As shown in Fig. 8, an $N \times N$ RNG matrix is capable of generating $4N$ N-bit random numbers in parallel. Each true RNG unit is shared among four random numbers. This design can maintain limited correlation because all unit RNGs are independent and

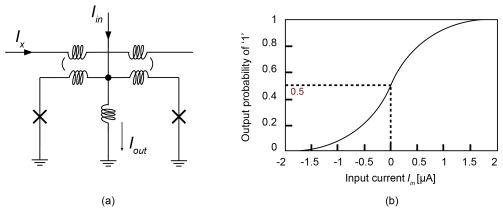

Figure 7: (a). 1-bit true RNG in AQFP; (b). Output distribution (which will converge to 0 and 1).

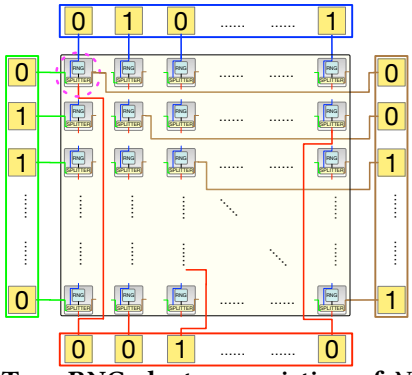

Figure 8: True RNG cluster consisting of $N \times N$ unit true RNGs, where each unit is shared by four N-bit random numbers.

each two output random numbers only share a single bit in common.

4.2 Integration of Summation and Activation Function in CONV Layers

To overcome the difficulty in accumulator implementation in AQFP, we re-formulate the operation of SC-based feature extraction block in a different aspect. The stochastic number output of inner-product (and activation), **SO**, should reflect the summation of input-weight products, whose stochastic format can be viewed as a matrix **SP**. Each row of **SP** is the stochastic number/stream of a input-weight product. Therefore, the binary value represented by **SO** is:

$$\frac{2\times\sum_{i=1}^{N}\mathbf{SO}_i - N}{N} = clip\Big(\frac{2\times\sum_{i=1}^{N}\sum_{j=1}^{M}\mathbf{SP}_{i,j} - N\times M}{N}, -1, 1\Big) \quad (1)$$

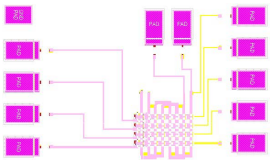

Figure 9: Layout of 1-bit true RNG using AQFP.

where N is the length of the stochastic stream and M is the number of inputs. The *clip* operation restricts the value between the given bounds. This formulation accounts for inner product (summation) and activation function. Consequently,

$$\sum_{i=1}^{N}\mathbf{SO}_i = clip\Big(\sum_{i=1}^{N}\sum_{j=1}^{M}\mathbf{SP}_{i,j} - \frac{M-1}{2}\times N, 0, N\Big) \quad (2)$$

That is, the total number of 1's in the output stochastic stream, **SO**, should equal to the total number of 1's in the input matrix, **SP**, minus $\frac{M-1}{2}\times N$. The n-th bit of **SO** can be determined by the following value:

$$\sum_{i=1}^{n}\sum_{j=1}^{M}\mathbf{SP}_{i,j} - i\times\frac{M-1}{2} - \sum_{i=1}^{n-1}\mathbf{SO}_i = \sum_{i=1}^{n-1}\Big(\sum_{j=1}^{M}\mathbf{SP}_{i,j} - \frac{M-1}{2} - \mathbf{SO}_i\Big) + \sum_{j=1}^{M}\mathbf{SP}_{n,j} - \frac{M-1}{2} \quad (3)$$

More precisely, the n-th bit of **SO** is 1 if the above value is greater than 0, and is 0 otherwise. The above calculation needs to accumulate $\sum_{j=1}^{M}\mathbf{SP}_{i,j} - (M-1)/2 - \mathbf{SO}_i$, further denoted as $\mathbf{D}_i$, in each clock cycle. To simplify the computation given hardware limitations, we propose an effective, accurate approximation technique as shown in Algorithm 1.

As described in Algorithm 1, each column of the input bit-stream matrix **SP** (corresponding to the current clock cycle), combined with the remaining M bits from the previous iteration, are fed to the sorting block. By performing binary sorting, the sorting block will decide whether the number of 1's in all inputs is greater than $(M-1)/2$ or not. More specifically, the $(M-1)/2$-th bit of the sorted results can be used to determine the next stochastic bit of **SO** (to be 1 or 0). The M bits following the $(M-1)/2+1$-th bit at output, whose number of 1's represents $\sum_{i=1}^{n} clip(\mathbf{D}_i, 0, 1)$, are fed back to the sorting block. The above step acts as subtraction as the top $(M-1)/2$ bits are ignored, and the surplus 1's can still be used in the following iteration.

Algorithm 1: Proposed sorter-based feature extraction block (implemented by SC in hardware)

input : **SP** is the matrix containing all input-weight products and bias
N is the bit-stream length
M is the input size
output: **SO** is the activated inner-product.

$\mathbf{D}_{prev} = \mathbf{0}$; //initialize feed back vector to all 0
for $i++ < N$ **do**
  $\mathbf{D}_i = \mathbf{SP}[:i]$; //current column
  $\mathbf{D}_s = sort((D_i, D_{prev}), descending)$; //sort the vector consist of the current column and previous feedback in descending order
  $\mathbf{SO}[i] = \mathbf{D}_s[(M-1)/2]$; // check the $(M-1)/2$-th bit for output as the top $(M-1)/2$ bits are subtracted each iteration
  $\mathbf{D}_{prev} = \mathbf{D}_i[(M+1)/2 : (M+1)/2+M]$; // feedback the M bits following the $(M-1)/2$-th bit for next iteration

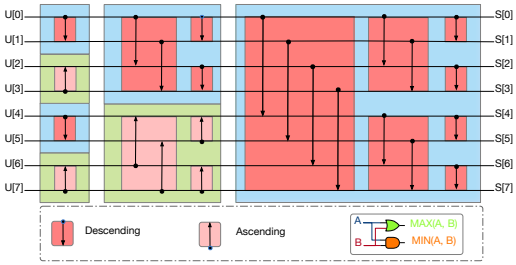

Figure 10: Example of 8-input binary bitonic sorter.

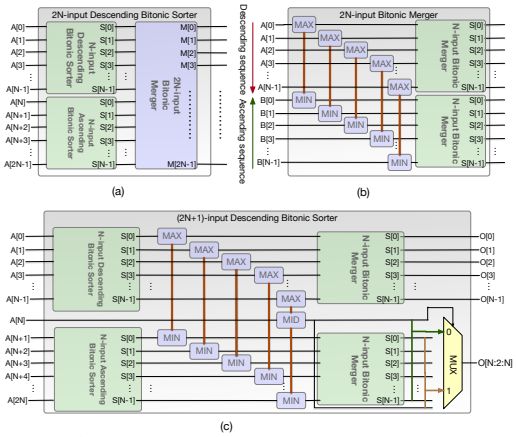

Figure 11: (a) Bitonic sorter for even-numbered inputs. (b) Bitonic merger for even-numbered inputs. (c) Bitonic sorter for odd-numbered inputs.

The binary sorting operation can be realized using the bitonic sorting network [25], which applies a divide-and-merge technique. Fig. 10 shows an 8-input binary bitonic sorter, where each sorting unit can be implemented using an AND gate for the maximum and an OR gate for the minimum of the two input bits. A bitonic sorter has two stages: sorting the top and bottom halves of its inputs where the bottom half is sorted in the opposite order of the top half; then merge the two sorted parts. As shown in Fig. 11 (a), a binary bitonic sorter with even inputs can be built up with bitonic sorters with 1/2 size and a bitonic merger, which is shown in Fig. 11 (b). For a binary bitonic sorter with odd inputs, we propose to add a three-input sorter in the first merging stage to sort the maximum of the bottom half, the middle input, and the minimum of the top half. The maximum of the three is fed to the top merger; the minimum is feed to the bottom merger; and the median, which is also the median of the entire inputs or the minimum, is used as a select signal to decide the bottom half output. This design allows any binary bitonic sorter to be implemented in a modular manner. The three-input sorter can be implemented using an AND gate, an OR gate and an majority gate for the median.

As shown in Fig. 12, the operation in (3) can be realized using a bitonic sorting circuit. As the feedback vector is already sorted, only the input column needs to be sorted. The

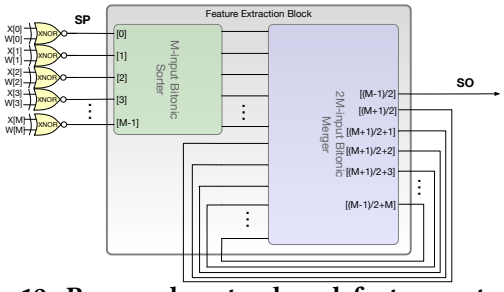

Figure 12: Proposed sorter based feature-extraction block.

two sorted sequences are then merged using a bitonic merger to form the final sorted vector. When M is an even number, a neutral noise of repeated 0 and 1 stochastic sequence (whose value is 0) is added to the input, in order to mitigate the limitation that $(M-1)/2$ cannot be fully represented in an integer format in this case.

As shown in Fig. 13, the output of the proposed feature extraction block resembles a shifted rectified linear unit (ReLU). Table 1 shows the absolute error of the proposed bitonic sorter based feature extraction block. The performance of the proposed sorter-based feature extraction block is very consistent, and does not degrade as the input size increases. The inaccuracy is no more than 0.06 for bit-streams longer than 1024.

4.3 Sub-Sampling Block in AQFP

The sub-sampling operation is realized using the pooling block. The max-pooling operation is not compatible with AQFP as it requires FSM for implementation, while the multiplexer based average pooling implementation [35] suffers from high inaccuracy as input size grows.

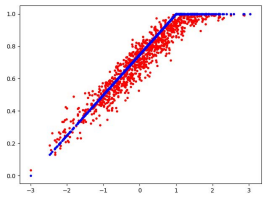

Figure 13: Activated output of the proposed feature extraction block.

Table 1: Absolute inaccuracy of the bitonic sorter-based feature extraction block.

Input size	Bit-stream length				
	128	256	512	1024	2048
9	0.1131	0.0847	0.0676	0.0573	0.0511
25	0.1278	0.0896	0.0674	0.0536	0.0434
49	0.1267	0.0954	0.0705	0.0528	0.0468
81	0.129	0.0937	0.0685	0.0531	0.0396
121	0.1359	0.0942	0.0654	0.0513	0.0374

Algorithm 2: Proposed sorter-based average-pooling block (implemented by SC in hardware)

input : **SP** is the matrix containing all inputs
N is the bit-stream length
M is the input size
output: **SO** is the average of its inputs.

$\mathbf{D_{prev}} = \mathbf{0}$; //initialize feed back vector to all 0
for $i++ < N$ **do**
  $\mathbf{D_i} = \mathbf{SP}[:i]$; //current column
  $\mathbf{D_s} = sort((D_i, D_{prev}), descending)$; //sort the vector consist of the current column and previous feedback in descending order
  if $\mathbf{D_s}[M] == 1$ **then**
    $\mathbf{D_{prev}} = \mathbf{D_s}[1:M]$; // less than M 1s have been encountered since last 1 has been output, feedback current saved 1s for next iteration
  else
    $\mathbf{D_{prev}} = \mathbf{D_s}[M+1:2M]$; // more than M 1s encountered, output should be 1 and feedback the following surplus M bits
  $\mathbf{SO}[i] = \mathbf{D_s}[M]$; //

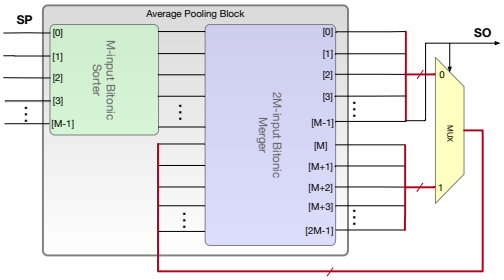

Figure 14: Proposed bitonic sorter based sub-sampling block.

We address these limitations and propose an accurate, AQFP-based average pooling block. The output stochastic number/stream of the average-pooling block, **SO**, should satisfy:

$$\frac{2\times\sum_{i=1}^{N}\mathbf{SO}_i - N}{N} = clip\left(\frac{2\times\sum_{i=1}^{N}\sum_{j=1}^{M}\mathbf{SP}_{i,j} - N\times M}{N\times M}, -1, 1\right) \quad (4)$$

Therefore:

$$\sum_{i=1}^{N}\mathbf{SO}_i = clip\left(\frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\mathbf{SP}_{i,j}}{M}, 0, N\right) \quad (5)$$

In other words, the number of 1's in **SO** should be equal to the number of 1's in **SP** divided by M (the number of inputs). Using the similar idea as the proposed feature extraction block, the sub-sampling block can also be realized using a sorting circuit.

As described in Algorithm 2, the current column of inputs and previous feedback are sorted to group all 1's since the last 1 produced in **SO**. The M-th bit is used to determine the next output bit and feedback bits. If a 1 is produced, the top M 1's are discarded and the remaining are feedback to the input. Otherwise, the top M bits are feedback for further

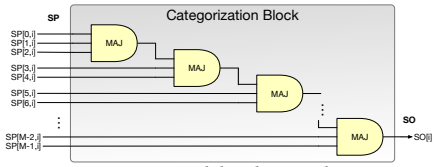

Figure 15: Categorization block implementation using majority chain structure.

accumulation. In summary, this would allow one 1 to be produced in **SO** for every M 1's in **SP**.

The average pooling block is also implemented using a bitonic sorter, as shown in Fig. 14. Similar to the design of the feature extraction block, only one M-input sorter is needed to sort the input column as the feedback vector is already sorted. The final sorted vector can be generated using a bitonic merger to merge the sorted input column and previous feedback. The output bit also acts as the select signal for feedback vector.

The proposed sub-sampling block produces very low inaccuracy as presented in Table 2, which is usually far less than 0.01 for bit-streams longer than 1024. Even when the input size is very small, the proposed bitonic sorter-based average pooling block can still operate with negligible error, which is no more than 0.025 regardless of the bit-stream length.

4.4 Categorization Block for FC Layers in AQFP

Categorization blocks are for (final) FC layers of DNNs, in which each output is the inner-product of corresponding input and weight vectors. We find it not ideal to use the same implementation for the categorization block as feature-extraction block for CONV layers, as the categorization block usually involves more inputs, leading to higher hardware footprint. In addition, the categorization result is reflected via the ranking of the outputs rather than relying on the calculation accuracy of inner product. As a result of the above two reasons, we propose to implement the categorization block in a less complicated manner, which is capable of maintaining the relative ranking of outputs without offering precise inner-product.

The proposed categorization block is implemented using the majority logic, whose output is the majority of all its inputs, i.e. the output is 1 if the input vector has more 1's than

Table 2: Absolute inaccuracy of the bitonic sorter-based average-pooling block.

Input size	Bit-stream length				
	128	256	512	1024	2048
4	0.0249	0.0163	0.0115	0.0085	0.0058
9	0.0173	0.0112	0.0079	0.0055	0.0039
16	0.0141	0.0089	0.0061	0.0042	0.0030
25	0.0122	0.0078	0.0049	0.0033	0.0024
36	0.0105	0.0065	0.0043	0.0029	0.0019

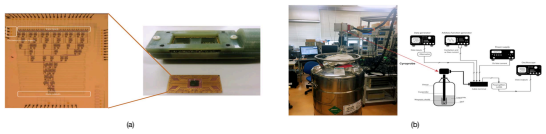

Figure 16: AQFP chip testing.

0's; the output is 0 otherwise. The relative value/importance of each output can be reflected in this way, thereby fulfilling the requirement of categorization block in FC layers.The proposed categorization logic can be realized using a simple majority chain structure. Thanks to the nature of AQFP technology, a three-input majority gate costs the same hardware resource as a two-input AND/OR gate. As shown in Fig. 15, a multi-input majority function can be factorized into multi-level of three-input majority gate chain based on the following equation: $Maj(x_0, x_1, x_2, x_4, x_5) = Maj(Maj(x_0, x_1, x_2), x_4, x_5)$.

Table. 3 shows the top 1 inaccuracy of the proposed categorization block. We simulate 10 categorization outputs with 100, 200, 500 and 800 different inputs under different lengths of bit-streams. The inaccuracy is evaluated using the relative difference between the highest output value in software and in SC domain. The relative inaccuracy can be limited to 0.4%, i.e., if the largest output outscores the second largest by more than 0.4%, the majority chain based categorization block can give the correct classification. As in most categorization cases, the highest output is usually far greater than the rest, the proposed majority chain categorization block should be able to operate with high accuracy.

5 SOFTWARE RESULTS AND HARDWARE PROTOTYPE TESTING

The proposed SC-based DNN using AQFP is evaluated in various aspects: 1) component-wise hardware performance; 2) application-level performance; 3) system-level hardware utilization efficiency. For hardware performance of the four basic blocks, we compare their energy efficiency and latency with CMOS-based counterparts. The application-level performance is evaluated by testing the accuracy of the proposed framework on the MNIST [21] dataset. Finally, the overall hardware resource utilization for two different DNNs (one for the MNIST dataset and one with deeper architecture for more complex tasks) are compared with SC-based DNN implementations using CMOS.

Table 3: Relative inaccuracy of the majority chain-based categorization block

Input size	Bit-stream length				
	128	256	512	1024	2048
100	0.3718%	0.2198%	0.1235%	0.0620%	0.0376%
200	0.2708%	0.2106%	0.1671%	0.0743%	0.0301%
500	0.2769%	0.2374%	0.1201%	0.0687%	0.0393%
800	0.2780%	0.1641%	0.1269%	0.0585%	0.0339%

Table 4: Hardware utilization of stochastic number generator

Output size	Energy(pJ)		Delay(ns)	
	AQFP	CMOS	AQFP	CMOS
100	9.700E-5	14.42	0.2	0.6
500	4.850E-4	72.11	0.2	0.6
800	7.760E-4	115.4	0.2	0.6

In addition we have verified the functionality of a feature extraction chip, as shown in Fig. 16 (a), which is fabricated with AIST $10kA/cm^2$ Niobium high-speed standard process (HSTP) [42]. The functionality of the chip is verified through 4.2K low-temperature measurements. As shown in Fig. 16 (b), the under-testing chip is embedded in a cryoprobe and inserted into a liquid Helium Dewar to cool down to 4.2K, with the protection of a double-layer magnetic shield. On-chip I/Os are connected to a cable terminal through the cryoprobe. A data pattern generator, an arbitrary function generator and a power supply are employed to generate the date inputs and excitation current. Oscilloscopes with pre-amplifiers are used to read the outputs.

5.1 Hardware Utilization

To validate the efficiency of the proposed SC-based DNN using AQFP, we firstly compare components-wise and overall hardware utilization with CMOS-based implementation. The CMOS implementation is synthesized with 40nm SMIC CMOS process using Synopsys Design Compiler. We test and compare SNGs, feature extraction blocks, sub-sampling blocks and categorization blocks with physical different configuration.

Table 4 shows hardware utilization comparison for SNGs to generate 1024 bit-stream stochastic numbers based on 10-bit random numbers generated by RNGs. The AQFP based design is much efficient given its advantage in true random number generation in addition to the clusterized RNG design. The proposed AQPF SNG operates at 1.48×10^5 times higher energy efficiency compared to its CMOS counterpart. The

Table 5: Hardware utilization of feature extraction block

Input size	Energy(pJ)		Delay(ns)	
	AQFP	CMOS	AQFP	CMOS
9	2.972E-4	320.819	2.2	1024.0
25	1.350E-3	520.704	3.4	1228.8
49	3.978E-3	843.469	4.8	1535.0
81	9.168E-3	1099.776	6.6	1741.8
121	1.333E-2	2948.496	6.8	1946.6
500	9.147E-2	6807.552	10.8	2455.6
800	0.186	9804.800	12.4	2868.2

Table 6: Hardware utilization of sub sampling block

Input size	Energy(pJ)		Delay(ns)	
	AQFP	CMOS	AQFP	CMOS
4	5.898E-5	18.432	1.2	614.3
9	3.007E-4	21.504	2.4	716.8
16	9.063E-4	23.552	3.4	819.2
25	1.359E-3	24.576	3.6	819.2
36	2.946E-3	32.768	5	921.6

bitonic sorter based feature extraction block using AQFP is very energy efficient as its energy efficiency is 1.08×10^6 times higher than CMOS based implementation while being much faster thanks to its effective bitonic sorter-based architecture. For very large and dense layers, we still consider them as feature extraction layers as they derives global features. The sorter-based implementation, regardless of the big input size, can still maintain reasonable hardware performance.

The bitonic sorter-based average pooling block using AQPF out-performs its CMOS counterpart by 3.12×10^5 times in energy efficiency and up to 465.46 times in computational speedup. This relative margin is lower because that the CMOS average pooling block implementation simply uses a multiplexer. Therefore the complexity is lower than the proposed bitonic sorter based implementation. However, as aforementioned, the proposed sorter-based average-pooling block can achieve extremely low inaccuracy. Finally, the categorization block can achieve up to 7.76×10^5 times better energy efficiency compared to its CMOS counterpart. Its consumption grows linearly given the majority chain design structure, whose size also grows linearly as input size increases. Overall the hardware consumption of the four basic proposed blocks are far more efficient in energy consumption compared to CMOS-based implementation.

5.2 Application Performance

With all components being well developed and configured, we build a convolutional neural network that has the architecture of Conv3_x – AvgPool – Conv3_x – AvgPool – FC500 – FC800 – OutLayer, as indicated in Table 8 The validity of the network is proven by performing the MNIST [21] classification. As the first step to implement a SC-based DNN using AQFP with high inference accuracy, the network is trained with taking all limitations of AQFP and SC into

Table 7: Hardware utilization of categorization block

Input size	Energy(pJ)		Delay(ns)	
	AQFP	CMOS	AQFP	CMOS
100	1.008E-2	7825.408	10	1945.6
200	3.957E-2	17131.220	20	2252.8
500	0.244	37396.480	50	2867.2
800	0.624	58880.409	80	4300.8

Table 8: DNN Layer Configuration

Layer Name	Kernel Shape	Stride
Conv3_x	$[3 \times 3, 32]$	1
Conv5_x	$[5 \times 5, 32]$	1
Conv7_x	$[7 \times 7, 64]$	1
Conv9_x	$[9 \times 9, 128]$	1
AvgPool	$[2 \times 2]$	2
FC500	500	–
FC800	800	–

considerations, thereby to prevent the accuracy degradation as much as possible. As shown in Table 9, the inference accuracy of this shallow neural network (SNN) is 97.91% when it is implemented in AQFP, and that of its CMOS counterpart is 97.35%. Although their accuracy are comparable, the hardware performance of AQFP-based implementation is much more remarkable than the CMOS implementation. The energy improvement of AQFP-based implementation is up to 5.4×10^4 times and its throughput improvement is 35.9 times, comparing to its CMOS counterpart.

As discussed above, the AQFP-based neural network can achieve acceptable inference accuracy with remarkable hardware performance when a shallow neural network is implemented. To further explore the feasibility of building a deep neural network in AQFP, we build a deep neural network with the architecture of Conv3_x – Conv3_x – AvgPool – Conv5_x – Conv5_x – AvgPool – Conv7_x – FC500 – FC800 – OutLayer, and the configuration of each layer is indicated in Table 8. As shown in Table 9, the AQFP-based DNN can achieve 96.95% accuracy in MNIST classification, and the energy efficiency of AQFP is even more remarkable when the network is deeper, the energy improvement is up to 6.9×10^4 times and the throughput improvement is 29 times, comparing to the CMOS-based implementation.

6 CONCLUSION

In this work, we propose a stochastic-computing deep learning framework using Adiabatic Quantum-Flux-Parametron superconducting technology, which can achieve the highest energy efficiency among superconducting logic families. However, the deep-pipelining nature of AQFP circuits makes the prior design in SC-based DNN not suitable. By taking account of this limitation and other characteristics of AQFP, we redesign the neural network components in SC-based DNN. We propose (i) an accurate integration of

Table 9: Network Performance Comparison

Network	Platform	Accuracy	Energy(μJ)	Throughput(images/ms)
SNN	Software	99.04%	–	–
	CMOS	97.35%	39.46	231
	AQFP	97.91%	5.606E-4	8305
DNN	Software	99.17%	–	–
	CMOS	96.62%	219.37	229
	AQFP	96.95%	2.482E-3	6667

summation and activation function using bitonic sorting network and feedback loop, (ii) a low-complexity categorization block based on chain of majority gates, (iii) an ultra-efficient stochastic number generator in AQFP, (iv) a high-accuracy sub-sampling (pooling) block in AQFP, and (v) majority synthesis for further performance improvement and automatic buffer/splitter insertion for requirement of AQFP circuits. Experimental results show that our proposed AQFP-based DNN can achieve up to 6.9×10^4 times higher energy efficiency compared to CMOS-based implementation while maintaining 96% accuracy on the MNIST dataset.

ACKNOWLEDGMENTS

This paper is in part supported by National Science Foundation CNS-1704662, CCF-1717754, CCF-1750656, and Defense Advanced Research Projects Agency (DARPA) MTO seedling project.

REFERENCES

[1] Armin Alaghi and John P Hayes. 2013. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)* 12, 2s (2013), 92.

[2] Armin Alaghi, Weikang Qian, and John P Hayes. 2018. The promise and challenge of stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 8 (2018), 1515–1531.

[3] Arash Ardakani, François Leduc-Primeau, Naoya Onizawa, Takahiro Hanyu, and Warren J Gross. 2017. VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2688–2699.

[4] Suyoung Bang, Jingcheng Wang, Ziyun Li, Cao Gao, Yejoong Kim, Qing Dong, Yen-Po Chen, Laura Fick, Xun Sun, Ron Dreslinski, Trevor Mudge, Hun Seok Kim, David Blaauw, and Dennis Sylvester. 2017. 14.7 A 288μW programmable deep-learning processor with 270KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 250–251.

[5] Bradley D Brown and Howard C Card. 2001. Stochastic neural computation. I. Computational elements. *IEEE Transactions on computers* 50, 9 (2001), 891–905.

[6] Ruizhe Cai, Ao Ren, Ning Liu, Caiwen Ding, Luhao Wang, Xuehai Qian, Massoud Pedram, and Yanzhi Wang. 2018. VIBNN: Hardware Acceleration of Bayesian Neural Networks. *SIGPLAN Not.* 53, 2 (March 2018), 476–488. https://doi.org/10.1145/3296957.3173212

[7] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices* 49, 4 (2014), 269–284.

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[9] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[10] John Clarke and Alex I Braginski. 2006. *The SQUID handbook: Applications of SQUIDs and SQUID systems*. John Wiley & Sons.

[11] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[12] Giuseppe Desoli, Nitin Chawla, Thomas Boesch, Surinder-pal Singh, Elio Guidetti, Fabio De Ambroggi, Tommaso Majo, Paolo Zambotti, Manuj Ayodhyawasi, Harvinder Singh, and Nalin Aggarwal. 2017. 14.1 A 2.9 TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 238–239.

[13] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 92–104.

[14] Brian R Gaines. 1967. Stochastic computing. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 149–156.

[15] Brian R Gaines. 1969. Stochastic computing systems. In *Advances in information systems science*. Springer, 37–172.

[16] James E Gentle. 2006. *Random number generation and Monte Carlo methods*. Springer Science & Business Media.

[17] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William J. Dally. 2017. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA.. In *FPGA*. 75–84.

[18] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 243–254.

[19] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[20] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 461–475.

[21] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[22] Vincent T Lee, Armin Alaghi, John P Hayes, Visvesh Sathe, and Luis Ceze. 2017. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 13–18.

[23] K. Likharev. 1977. Dynamics of some single flux quantum devices: I. Parametric quantron. *IEEE Transactions on Magnetics* 13, 1 (January 1977), 242–244. https://doi.org/10.1109/TMAG.1977.1059351

[24] K. K. Likharev and V. K. Semenov. 1991. RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Transactions on Applied Superconductivity* 1, 1 (March 1991), 3–28. https://doi.org/10.1109/77.80745

[25] Kathy J Liszka and Kenneth E Batcher. 1993. A generalized bitonic sorting network. In *Parallel Processing, 1993. ICPP 1993. International Conference on*, Vol. 1. IEEE, 105–108.

[26] K. Loe and E. Goto. 1985. Analysis of flux input and output Josephson pair device. *IEEE Transactions on Magnetics* 21, 2 (March 1985), 884–887. https://doi.org/10.1109/TMAG.1985.1063734

[27] Pierre LâĂŹEcuyer. 2012. Random number generation. In *Handbook of Computational Statistics*. Springer, 35–71.

[28] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. 2016. Tabla: A

unified template-based framework for accelerating statistical machine learning. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 14–26.

[29] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 2017. 14.5 Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 246–247.

[30] Shuichi Nagasawa, Yoshihito Hashimoto, Hideaki Numata, and Shuichi Tahara. 1995. A 380 ps, 9.5 mW Josephson 4-Kbit RAM operated at a high bit yield. *IEEE Transactions on Applied Superconductivity* 5, 2 (1995), 2447–2452.

[31] T. Narama, F. China, N. Takeuchi, T. Ortlepp, Y. Yamanashi, and N. Yoshikawa. 2016. Yield evaluation of 83k-junction adiabatic-quantum-flux-parametron circuit. In *2016 Appl. Superconductivity Conference (ASC2016)*.

[32] Harald Niederreiter. 1992. *Random number generation and quasi-Monte Carlo methods*. Vol. 63. Siam.

[33] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 26–35.

[34] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 267–278.

[35] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. 2017. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. *ACM SIGOPS Operating Systems Review* 51, 2 (2017), 405–418.

[36] Ao Ren, Zhe Li, Yanzhi Wang, Qinru Qiu, and Bo Yuan. 2016. Designing reconfigurable large-scale deep learning systems using stochastic computing. In *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 1–7.

[37] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[38] Hyeonuk Sim and Jongeun Lee. 2017. A new stochastic computing multiplier with application to deep convolutional neural networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 29.

[39] Jaehyeong Sim, Jun-Seok Park, Minhye Kim, Dongmyung Bae, Yeongjae Choi, and Lee-Sup Kim. 2016. 14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*. IEEE, 264–265.

[40] Mingcong Song, Kan Zhong, Jiaqi Zhang, Yang Hu, Duo Liu, Weigong Zhang, Jing Wang, and Tao Li. 2018. In-Situ AI: Towards Autonomous and Incremental Deep Learning for IoT Systems. In *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 92–103.

[41] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 16–25.

[42] Naoki Takeuchi, Shuichi Nagasawa, Fumihiro China, Takumi Ando, Mutsuo Hidaka, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2017. Adiabatic quantum-flux-parametron cell library designed using a 10 kA cm-2 niobium fabrication process. *Superconductor Science and Technology* 30, 3 (2017), 035002.

[43] Naoki Takeuchi, Dan Ozawa, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2013. An adiabatic quantum flux parametron as an ultra-low-power logic device. *Superconductor Science and Technology* 26, 3 (2013), 035010.

[44] Naoki Takeuchi, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2013. Measurement of 10 zJ energy dissipation of adiabatic quantum-flux-parametron logic using a superconducting resonator. *Applied Physics Letters* 102, 5 (2013), 052602.

[45] Naoki Takeuchi, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2014. Energy efficiency of adiabatic superconductor logic. *Superconductor Science and Technology* 28, 1 (nov 2014), 015003. https://doi.org/10.1088/0953-2048/28/1/015003

[46] Naoki Takeuchi, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2015. Adiabatic quantum-flux-parametron cell library adopting minimalist design. *Journal of Applied Physics* 117, 17 (2015), 173912.

[47] Sergey K Tolpygo, Vladimir Bolkhovsky, Terence J Weir, Alex Wynn, Daniel E Oates, Leonard M Johnson, and Mark A Gouker. 2016. Advanced fabrication processes for superconducting very large-scale integrated circuits. *IEEE Transactions on Applied Superconductivity* 26, 3 (2016), 1–10.

[48] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 65–74.

[49] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, and Anand Raghunathan. 2017. ScaleDeep: A Scalable Compute Architecture for Learning and Evaluating Deep Networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 13–26.

[50] Yanzhi Wang, Zheng Zhan, Jiayu Li, Jian Tang, Bo Yuan, Liang Zhao, Wujie Wen, Siyue Wang, and Xue Lin. 2018. On the Universal Approximation Property and Equivalence of Stochastic Computing-based Neural Networks and Binary Neural Networks. *arXiv preprint arXiv:1803.05391* (2018).

[51] Paul N Whatmough, Sae Kyu Lee, Hyunkwang Lee, Saketh Rama, David Brooks, and Gu-Yeon Wei. 2017. 14.3 A 28nm SoC with a 1.2 GHz 568nJ/prediction sparse deep-neural-network engine with> 0.1 timing error rate tolerance for IoT applications. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 242–243.

[52] Dongbin Xiu. 2010. *Numerical methods for stochastic computations: a spectral method approach*. Princeton university press.

[53] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2016. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 1–8.

[54] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. 2016. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 326–331.

[55] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani B Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs.. In *FPGA*. 15–24.