JBSA: A Bit-Serial Accelerator for Deep Neural Networks Using Superconducting SFQ Logic

Yang Su
School of Information Science and Technology, ShanghaiTech University
Shanghai, China

Sheng Li
School of Information Science and Technology, ShanghaiTech University
Shanghai, China

Huilong Jiang
State Key Lab of Processors, Institute of Computing Technology, CAS
Beijing, China

Haofei Yin
School of Information Science and Technology, ShanghaiTech University
Shanghai, China

Rongliang Fu
Department of Computer Science and Engineering, The Chinese University of Hong Kong
Hong Kong, China

Junying Huang*
State Key Lab of Processors, Institute of Computing Technology, CAS
Beijing, China

Xiaochun Ye
State Key Lab of Processors, Institute of Computing Technology, CAS
Beijing, China

Zhimin Zhang
State Key Lab of Processors, Institute of Computing Technology, CAS
Beijing, China

Jie Ren
State Key Laboratory of Functional Materials for Informatics, SIMIT, CAS
Shanghai, China

Xiaoping Gao
State Key Laboratory of Functional Materials for Informatics, SIMIT, CAS
Shanghai, China

Tsung-Yi Ho
Department of Computer Science and Engineering, The Chinese University of Hong Kong
Hong Kong, China

Dongrui Fan
State Key Lab of Processors, Institute of Computing Technology, CAS
Beijing, China

Abstract

The potential of superconducting single flux quantum (SFQ) devices in accelerating deep neural networks (DNNs) has garnered significant attention due to their ultra-fast and low-power switching capabilities. However, existing SFQ-based DNN accelerators face limitations in scaling up to larger-scale instances due to the stringent area constraints and complex architectures. Additionally, another challenge in SFQ-based DNN acceleration lies in bridging the gap between the ultra-high computing speed offered by SFQ technology and the relatively low memory bandwidth. To address these challenges, we propose JBSA, an SFQ-based bit-serial accelerator for DNN inference acceleration. JBSA leverages bit-serial computing to alleviate area constraints and reduce bandwidth requirements. A bit-serial processing element is designed to implement multiply-accumulate operations using SFQ logic cells. Furthermore, we introduce a novel work distribution model that incorporates a data partitioning scheme and a corresponding scheduling strategy to enhance the overall efficiency of the accelerator. Experimental results show that JBSA surpasses the state-of-the-art SFQ-based DNN accelerator, achieving an 11% performance improvement, a 2.3× power efficiency improvement, and a 56% Josephson junction count reduction. Compared to CMOS-based bit-serial DNN accelerators, JBSA also demonstrates outstanding performance and superior power efficiency.

CCS Concepts

• **Hardware → Emerging architectures**; **Superconducting circuits**; • **Computer systems organization → Systolic arrays**.

Keywords

Superconducting logic, single flux quantum, cryogenic computing, hardware accelerator, bit-serial architecture

ACM Reference Format:
Yang Su, Sheng Li, Huilong Jiang, Haofei Yin, Rongliang Fu, Junying Huang, Xiaochun Ye, Zhimin Zhang, Jie Ren, Xiaoping Gao, Tsung-Yi Ho, and Dongrui Fan. 2025. JBSA: A Bit-Serial Accelerator for Deep Neural Networks Using Superconducting SFQ Logic. In *2025 International Conference on Supercomputing (ICS '25), June 8–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3721145.3729517

1 Introduction

Deep neural networks (DNNs) are state-of-the-art technology in a wide range of artificial intelligence (AI) tasks, including object recognition and speech recognition. To achieve high performance and energy efficiency, significant research efforts have been devoted to exploring hardware acceleration methods for DNN [3, 22]. However, the majority of these designs rely on complementary metal-oxide-semiconductor (CMOS) technology, which poses performance limitations as Moore's Law gradually breaks down. To address this challenge, innovative technologies such as quantum computing [24, 29], neuromorphic computing [30, 47], approximate computing

*Corresponding author: huangjunying@ict.ac.cn.

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/2025/06
https://doi.org/10.1145/3721145.3729517

[3, 8, 10, 37], and stochastic computing [7, 31, 42, 43] have emerged, offering potential solutions.

Among these innovative candidates, the Josephson junction (JJ)-based superconducting single flux quantum (SFQ) logic family is a highly promising solution, which offers ultra-fast switching speeds (in the tens of GHz range) and significantly lower energy consumption ($\sim 10^{-19} J$ per switch) [23, 26]. These high potentials have sparked growing interest in the research of superconducting NN accelerators in recent years, covering attempts to port existing CMOS-based accelerators [11, 18, 49], superconducting technology with less traditional computing paradigms [7, 14, 39, 40], and superconducting neuromorphic computing [5, 9, 35, 46].

While the JJ-based superconducting circuits can provide significant power and performance benefits, the development of SFQ DNN accelerators faces three fundamental constraints that motivate our architectural innovation. (1) Manufacturing-driven area constraints. Current SFQ fabrication technology remains at 250nm CMOS equivalent nodes [45], imposing severe integration density limitations. (2) Clocking-induced feedback loop challenges. The SFQ flow clocking mechanism creates critical path dependencies in feedback loops (Section 2.1.3), significantly reducing circuit operating frequencies. (3) Computation and memory bandwidth gap. The disparity between high SFQ logic computation speeds and memory bandwidth limits the effective performance achievable by SFQ circuits. Therefore, to address these design challenges, it is necessary to propose innovative architectures that minimize area overhead while enhancing performance and power efficiency in SFQ-based DNN accelerators.

Bit-serial computing architecture seems to be a promising solution due to its low area overhead, enhanced computational efficiency, and ability to reduce on-chip storage capacity (Section 4.1) compared to bit-parallel SFQ designs. However, the bit-serial scheme for SFQ logic still needs to address two problems: (1) eliminating feedback loops within processing elements (PEs) and (2) developing an efficient work distribution model for bit-serial computation. Bit-serial schemes often employ the output stationary (OS) dataflow, which typically involves feedback loops consisting of registers and bit-parallel adders. These loops significantly degrade the frequency of PEs within the flow clocking scheme. Furthermore, directly applying the bit-parallel work distribution model to bit-serial computation would result in extremely low computational and storage efficiency (Section 3.4). Therefore, to improve PE and on-chip buffer utilization, an efficient work distribution model for SFQ-based bit-serial computation is necessary.

In this paper, we propose JBSA, a bit-serial SFQ-based DNN accelerator. To create an efficient DNN accelerator using SFQ logic, we first implemented area-efficient Multiply-Accumulate (MAC) operations using bitwise AND operations, counting, and shifting, resulting in a reduced JJ overhead for the bit-serial PE. Our 16-bit configuration achieved a 59% reduction in JJ count compared to the bit-parallel design. Secondly, the bit-serial architecture eliminates feedback loops within the OS dataflow, significantly enhancing performance. Moreover, the OS dataflow reduces on-chip buffer usage by achieving high data reuse efficiency with minimal data volume, decreasing JJ count of buffers to 1/6 of that in the bit-parallel weight stationary (WS) dataflow (Section 5.4). Furthermore, leveraging the lower effective precision of data in bit-serial MACs reduces off-chip memory bandwidth requirements, bridging the gap between computing speed and memory bandwidth. Experimental results indicate that the off-chip memory bandwidth requirement of the bit-serial design is only 35% of that in SuperNPU [18] (Section 5.2).

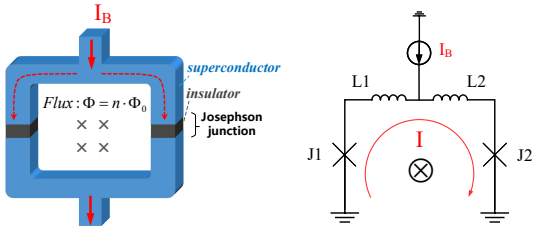

Figure 1. Superconducting ring with Josephson junctions and its equivalent circuit.

To evaluate JBSA's performance, power efficiency, and JJ overhead, we compare it with three DNN accelerators: (1) SuperNPU [18], (2) Loom [36], a state-of-the-art CMOS-based bit-serial DNN accelerator operating at room temperature (Loom@300K), and (3) its cryogenic version operating at 77 K (CryoLoom@77K). Our workload comprises six DNN models: AlexNet, GoogLeNet, NiN, VGG19, VGGM, and VGGS. The results show that JBSA outperforms SuperNPU with an 11% improvement in performance and 2.3× enhancement in power efficiency, while reducing the JJ count by 56%. Compared with Loom@300K (CryoLoom@77K), JBSA achieves a speedup of 316× (144×). Considering the 400× cooling cost, the energy-efficient rapid single flux quantum (ERSFQ)-based JBSA exhibits 1.2× (2.7×) higher power efficiency than Loom@300K (CryoLoom@77K). However, assuming no cooling cost, the power efficiency of ERSFQ-based JBSA significantly surpasses that of CMOS-based designs by 491× (101×) for Loom@300K (CryoLoom@77K), respectively.

In summary, our work makes the following contributions:

- To the best of our knowledge, this is the first design of an SFQ-based bit-serial DNN accelerator architecture that achieves exceptional performance and power efficiency.
- An SFQ-based bit-serial PE is proposed, which eliminates feedback loops within the OS dataflow and enables high-speed processing of MAC operations. The Bit-serial architecture assisted by OS dataflow remarkably reduces the area of the on-chip buffer.

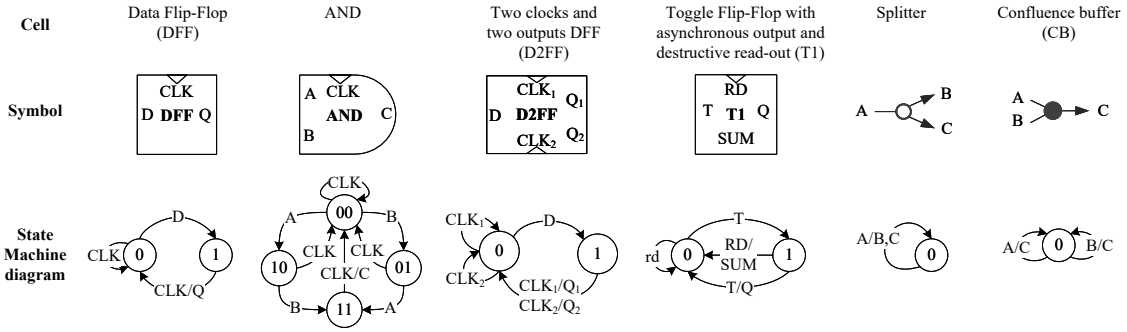

Figure 2. Symbols and state machine diagrams of fundamental SFQ gates: DFF, AND, D2FF, T1, splitter, and confluence buffer.

- A novel work distribution model, including a data partitioning scheme, a data scheduling strategy, and data dispatchers, has been designed.
- We thoroughly evaluated the performance, energy efficiency, and resource consumption of JBSA, comparing it with advanced SFQ-based and CMOS-based accelerators across multiple DNN architectures, revealing the significant advantages of JBSA.

2 Background

2.1 SFQ logic and its characteristics

The elementary circuit element of SFQ logic is a superconductor ring, as shown in Fig. 1. It can store and transfer the SFQ voltage pulse by using the superconducting device, JJ, which has a superconductor-insulator-superconductor structure. The magnetic flux in the superconductor loop is quantized as $\Phi = n \cdot \Phi_0$ where $\Phi_0 = h/2e \approx 2.07 \times 10^{-15}$ Wb and n is an integer. As shown on the right side of Fig. 1, when the current of J2 exceeds the critical value in a short time due to bias current and external source, J2 will produce a voltage pulse (SFQ pulse). In the context of SFQ circuits, the presence or absence of an SFQ pulse between two adjacent clock pulses represents a logic '1' or '0', respectively. It is the voltage pulse driving characteristics of SFQ circuits that enable the realization of JJ switching with extremely low latency ($\sim 10^{-12}$ s) and low energy consumption ($\sim 10^{-19}$ J) [23, 26].

2.1.1 Gate-level pipeline. In contrast to CMOS circuits, most SFQ logic gates need to be synchronized with the clock, requiring a clock pulse to transfer the stored SFQ to neighboring gates. In other words, almost every SFQ logic gate possesses latch functionality, enabling gate-level pipelining. Fig. 2 describes the symbols and state machine diagrams of SFQ gates used in this work.

2.1.2 Shift-register-based on-chip memory. While SFQ logic gates can be used to implement random access memory, the complex control logic required results in significant hardware overhead and consumes a large number of JJs, which limits the scalability of storage capacity [49]. Additionally, alternative SFQ storage technologies, such as vortex transition memory, Josephson-CMOS SRAM, magnetic memory, and superconducting nanowire memory, are either not mature enough or have strict manufacturing technology prerequisites. Thus, employing shift-register-based memory for SFQ circuits emerges as a more convenient approach.

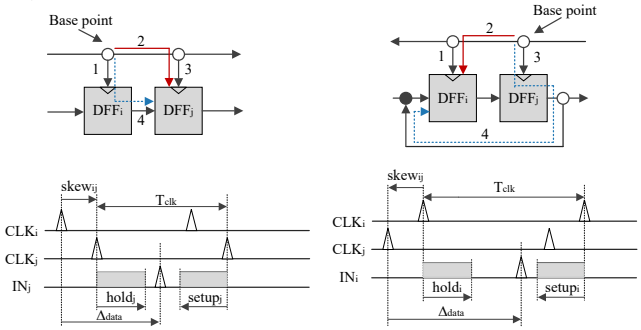

Figure 3. Typical SFQ clocking schemes: (a) concurrent-flow clocking, and (b) counter-flow clocking modified from [12].

2.1.3 Flow clocking. Due to the extremely high operating frequencies of SFQ circuits and the ubiquitous requirement for clock signal driving in most gates, SFQ's timing methodology diverges significantly from CMOS circuits. Two representative clocking schemes of SFQ circuits are concurrent-flow clocking and counter-flow clocking [12]. The feedback loop within flow clocking schemes affects the frequency of SFQ circuits. In the absence of a feedback loop, concurrent-flow clocking is recommended, where the clock pulse travels alongside the data to hide propagation delays (Fig. 3(a)). However, this clocking is not suitable when a feedback loop exists. In such cases, the preferable approach is to utilize counter-flow clocking (Fig. 3(b)), effectively concealing the data feedback delay. Therefore, for optimal performance, we employ concurrent-flow clocking for the PE and counter-flow clocking for the shift-register-based memory.

2.2 Bit-serial multiply-accumulate operation for signed numbers

The bit-serial MAC operation involves three steps: 1) performing bitwise AND operations to obtain 1-bit partial products for

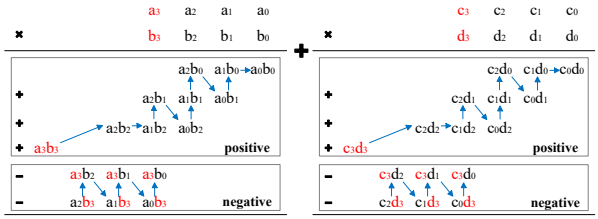

Figure 4. Example of bit-serial MAC operation: $A \cdot B + C \cdot D$, where A, B, C, and D are 4-bit signed numbers with a_3, b_3, c_3, and d_3 as sign bits.

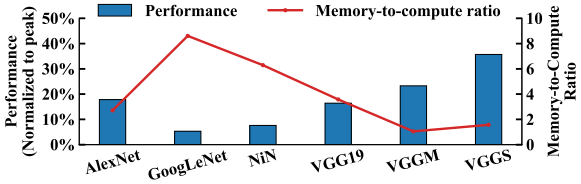

Figure 5. SuperNPU's performance (normalized to its peak performance) and the ratio of memory access time to computation time.

each bit position, 2) accumulating the partial products at the same bit position to obtain a partial sum (psum), and 3) shifting the psums from different bit positions based on their binary coefficients and accumulating them to obtain the final MAC result. For example, consider the MAC operation of $A \cdot B + C \cdot D$, where A, B, C, and D are 4-bit signed numbers, resulting in a 16-bit partial product as illustrated in Fig. 4. These partial products can be divided into positive and negative components, based on the principles of binary multiplication [4]. The positive partial products include bitwise AND operations between two sign bits, such as a_3b_3 and c_3d_3, as well as between two value bits, such as a_0b_0 and c_0d_0. The negative partial product is formed by the AND result of one sign bit and one value bit, such as a_3b_2 and c_3d_2. We can traverse the bit positions in a specific order, as suggested in [41], which involves shifting left by one position or maintaining the current position. This traversal order helps reduce the hardware resource overhead of shifters. For instance, the accumulation starts with a_3b_3 and c_3d_3 at the highest bit position, followed by two left-shift operations, and then the accumulation of a_2b_2 and c_2d_2. Notably, shifts and accumulations of positive and negative psums need to be performed separately. Finally, the positive psum is added to the 2's complement of the negative psum to obtain the final MAC result.

3 Motivation

In this section, we present the motivation behind the SFQ-based bit-serial DNN accelerator design. We first evaluate the performance, the JJ overhead, and the ifmap buffer utilization of the SFQ-based bit-parallel DNN accelerator, SuperNPU, under memory bandwidth constraints. To model SuperNPU, we develop a simulator based on SCALE-Sim [32], a cycle-accurate simulator targeting specifically designed systolic-array-based DNN accelerators. Moreover, we establish a Baseline SFQ-based bit-serial DNN design as described in Section 3.4. The design was evaluated using six different network models with varying topological structures, computational intensities, and off-chip memory bandwidth requirements.

3.1 Actual performance of SuperNPU

In the current landscape of SFQ-based bit-parallel DNN accelerators, the Von Neumann memory wall [44] poses a significant challenge. Despite assuming a memory bandwidth of 300 GB/s, equivalent to TPUv2 [2], the memory access times of SuperNPU are still 3× longer than the computation time on average, as depicted in Fig. 5. This discrepancy even reaches up to 8.6× for GoogLeNet. The large gap between the high computing speed of tens of GHz and the limited memory bandwidth significantly reduces the actual performance when inferring a large batch of images, amounting to only 15% of the peak performance, as shown in Fig. 5. This limitation is primarily attributed to the relatively low memory bandwidth, which imposes constraints on the achievable performance. Notably, GoogLeNet exhibits the poorest performance due to the most pronounced discrepancy between its computation time and memory access time.

3.2 Area overhead of SuperNPU

SuperNPU adopts the WS dataflow to eliminate feedback loops in PEs. To enhance the overall throughput, the roofline model suggests that increasing computational intensity, i.e., the number of MACs executed with one weight mapped onto the PE, is crucial [18]. In SuperNPU, this is achieved by efficiently processing a large batch of images concurrently for inference, reducing off-chip memory access by strategically expanding on-chip buffer capacity to accommodate ifmap data from multiple images simultaneously. However, this expansion results in a substantial overhead of JJ count being devoted to buffers instead of MACs. Specifically, the on-chip buffer requires 81% of the total JJs, while the PE array accounts for only 17%, as depicted in Fig. 6. The primary factor stems from the requirement that a single weight mapping encompasses feature map pixels spanning nearly the entire channel, thereby demanding a substantial volume of data to attain high data reuse efficiency. This is particularly evident in batch-based inference scenarios. This represents a major limitation in SFQ-based DNN designs that rely on the WS dataflow. In this paper, we adopt the OS dataflow commonly used in bit-serial architectures. By addressing the limitations of this dataflow, we improve its practicality and unlock its potential for advancing SFQ-based DNN accelerator designs.

3.3 Ifmap buffer underutilization of SuperNPU

The considerable capacity of SuperNPU's ifmap buffer results in a notable underutilization of resources. Experimental results

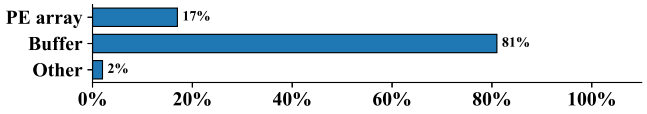

Figure 6. Breakdown of SuperNPU's JJ count: distribution among PEs, buffers, and other circuits.

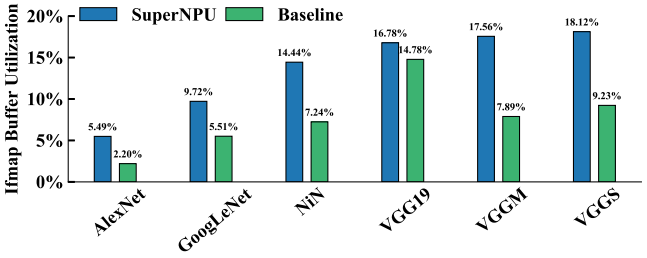

Figure 7. Ifmap buffer utilization of SuperNPU and Baseline.

show that the ifmap buffer utilization ratio is only 12.7% when inferring a large batch of images, as shown in Fig. 7. This is because the capacity of the ifmap buffer depends on the data volume of the largest DNN layer, while most layers are significantly smaller than this layer.

3.4 Baseline SFQ-based bit-serial DNN design

A straightforward approach to implementing a superconducting bit-serial DNN accelerator is to draw inspiration from the architecture of CMOS-based bit-serial DNN accelerators and implement it using SFQ logic since extensive research on CMOS-based bit-serial DNN accelerators exists. Thus, we first architect an SFQ-based bit-serial DNN accelerator by adopting the OS dataflow, implementing Loom's bit-serial PE using SFQ logic together with SuperNPU's systolic on-chip network, and employing shift-register-based on-chip memory. The bit-serial PE derived from Loom includes a feedback loop consisting of a 32-bit bit-parallel adder for the accumulation of psums, leading to a significant degradation in the PE's operational frequency. Consequently, this inherent drawback considerably compromises the performance of Baseline.

In the Baseline design, we adopt SuperNPU's work distribution model which each shift register is exclusively allocated to store the ifmap data for a given channel, subsequently providing this data to 4 rows of PEs. As a result, only 4 rows of PEs are mapped with the same image's output feature map (ofmap) data. Considering an ofmap size of 56×56, a total of 784 mapping rounds is required to complete the computation. Due to the sequential memory access of the shift registers, each mapping operation compels the complete readout of all ifmap data from the buffer. This also implies that the computation time is directly proportional to the product of the ifmap and ofmap sizes. This undoubtedly leads to extremely low efficiency since a convolution operation only requires a small amount of ifmap data, and the majority of ifmap data

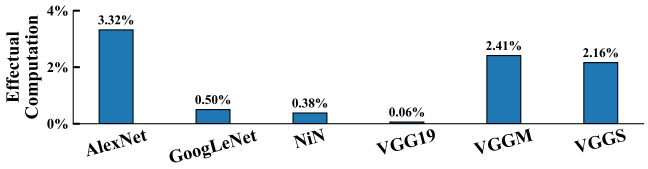

Figure 8. Baseline's effectual computation.

fed into the PEs is ineffectual. Experimental results show that the effectual MACs are only 0.76% as shown in Fig. 8.

Furthermore, in bit-serial computing, the reduction in the precision of the ifmap and weight corresponds to a decrease in data volume. Correspondingly, similar to SuperNPU, the oversized ifmap buffer of Baseline also suffers from underutilization. Fig. 7 shows that the ifmap buffer utilization ratio is only 6.8% when inferring a large batch of images, even lower than that of SuperNPU.

3.5 Research challenges and goals

As the Baseline design described above suffers from various performance and efficiency constraints, we are in dire need of convincing architectures to effectively circumvent the bottlenecks.

Removing feedback loops in PEs. PEs with the OS dataflow typically include feedback loops consisting of registers and bit-parallel adders, which significantly degrade the PEs' operating frequency within the concurrent-flow clocking scheme. Architects must remove the feedback loop inherent in the PE to harness the high frequency of concurrent-flow clocking, thereby enhancing the overall performance of the design.

Absence of a work distribution model for bit-serial computation using SFQ logic. As mentioned above, directly applying the work distribution model of SuperNPU to bit-serial computation would result in a low effective computation ratio. Therefore, to improve the utilization of the PE and on-chip buffer, an efficient work distribution model for SFQ-based bit-serial computation should be designed.

This paper proposes JBSA to resolve these challenges. We first implement a concise bit-serial PE that performs MAC operations using SFQ logic cells (Section 4.2). Next, we propose a novel work distribution model to match the bit-serial computation (Section 4.3). Finally, the effectiveness of JBSA is evaluated (Section 5).

4 JBSA Architecture

In this section, we first introduce the overall architecture of JBSA, including an overview of its main components. Next, we provide a detailed explanation of the SFQ-based bit-serial PE design and the work distribution model, aiming to address the challenges mentioned above. Finally, we conduct a comprehensive analysis of various architectural configuration parameters to illustrate the design decisions made for JBSA.

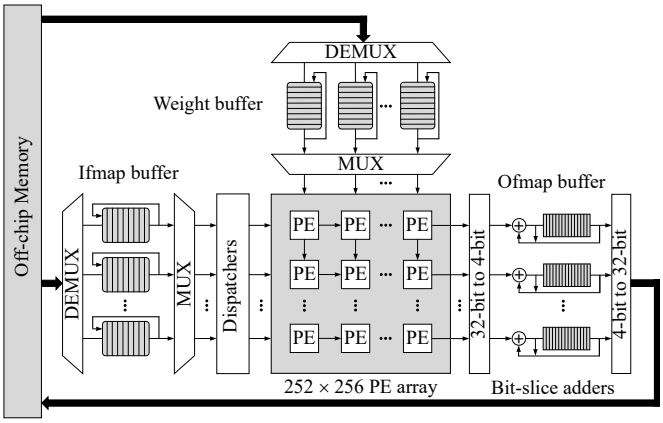

Figure 9. The overview of JBSA.

4.1 Overall architecture

Fig. 9 shows the overview of JBSA. The major components of JBSA can be summarized as follows.

252×256 systolic PE array: The PE array adopts a systolic on-chip network structure, as it offers the highest operating frequency and minimal area overhead under the flow clocking of SFQ technology [18]. Additionally, the systolic array facilitates the timing alignment of data arrivals between the sequential buffer and PEs. The PE array scale is 252×256.

Three on-chip buffers: The on-chip buffer is divided into three areas for ifmaps, ofmaps, and weight filters, respectively. The capacity of the ifmap buffer is 2.2 MB, which is 10× smaller than that of superNPU. This is because JBSA employs the OS dataflow, in which each ofmap pixel is assigned to a given PE and only a portion of ofmap pixels is allowed to be computed concurrently due to the limited size of the PE array. As a result, the ifmap buffer needs to retain only the ifmap data required for computing the current ofmap pixels, which is further constrained by the size of weight filters. This eliminates the necessity to concurrently store all ifmap data on-chip, thereby leading to a reduction in ifmap buffer capacity requirements and a minimization of ifmap buffer area overhead. The capacity of the ofmap buffer is 252 KB, which is only 1% of that of SuperNPU, yet it is sufficient to store all positive psums or final ofmap data mapped to PEs. The weight buffer has a capacity of 4.5 MB, double that of the ifmap buffer. This allows it to store as many weight filters as possible, reducing the need for off-chip memory access, since weights are highly reusable in convolution operations.

To minimize data movement overhead and underutilization in the shift register, JBSA partitions a larger buffer into multiple smaller chunks, similar to SuperNPU [18]. The ifmap buffer is partitioned into 128 chunks and connected by the multiplexer (MUX) and demultiplexer (DEMUX) trees, with different chunks storing ifmap data of different bit positions from different images. The DEMUX tree selects a chunk to store data, while the MUX tree activates an appropriate chunk to provide data for computation. Each chunk consists of 252 shift registers corresponding to 252 PE rows. The length of each ifmap shift register is 36. The weight buffer is also partitioned into 128 chunks, with different chunks storing weights of different bit positions. Each chunk consists of 256 shift registers corresponding to 256 weight filters fed into 256 PE columns. The length of each weight shift register is 72. The ofmap buffer consists of 504 shift registers, each of which stores positive psums or final ofmap data for half a PE row.

504 4-bit bit-slice adders: To accumulate the psums produced by 8 Shift counters (as described in Section 4.2) in each PE, as well as to add the positive and negative psums, we assign an adder to each half row of PEs, resulting in a total of 504 adders. To mitigate area overhead, we employ a 4-bit bit-slice carry look-ahead adder structure, based on a 4-bit bit-slice arithmetic logic unit [38], which enables the operation of a 32-bit addition every 8 cycles. To support the 4-bit bit-slice adders, JBSA integrates circuits for data conversion between 32-bit and 4-bit formats.

Additionally, JBSA includes dispatchers to send ifmap data to the appropriate PE rows at the precise time. To achieve a high operating frequency, JBSA employs concurrent-flow clocking for the PE array and dispatchers, while employing counter-flow clocking for on-chip buffers and bit-slice adders.

4.2 SFQ-based bit-serial PE design

We design a feedback-free bit-serial PE with the OS dataflow, as shown in Fig. 10. During each clock cycle, the PE performs bitwise AND operations between the i-th bits of 16 consecutive ifmap pixels and the j-th bits of 16 consecutive weight pixels along the channel dimension, resulting in 16 partial products at the specific bit position. Accumulating these single-bit partial products is achieved through counting operations.

As described in Section 2.2, the psums from different bit positions necessitate either shifting left or maintaining the current position before accumulation. To facilitate these operations, we introduce an SFQ-based module called *Shift counter*, as shown in Fig. 10. The Shift counter comprises a series of T1 gates, collectively forming a 32-bit counter. Each T1 gate functions as a modulo-two counter, generating an asynchronous output pulse as a carry signal into the higher T1 gate for every two input pulses, as illustrated in Fig. 2. Upon completion of counting the AND results of all pixels for a particular bit position, the psum remains in the counter if the binary coefficient of its bit position is equal to that of the next bit position (e.g. a_1b_2 and a_2b_1 in Fig. 4). Otherwise, a *read* signal is triggered, causing the psum to be destructively read from the counter and stored in a set of D2FFs. In the next clock cycle, a *shift* signal is activated to left-shift the psum by one position and transfer it back to the counter. If the psum needs to be shifted by two positions (e.g., a_3b_3 to a_2b_2 in Fig. 4), an additional set

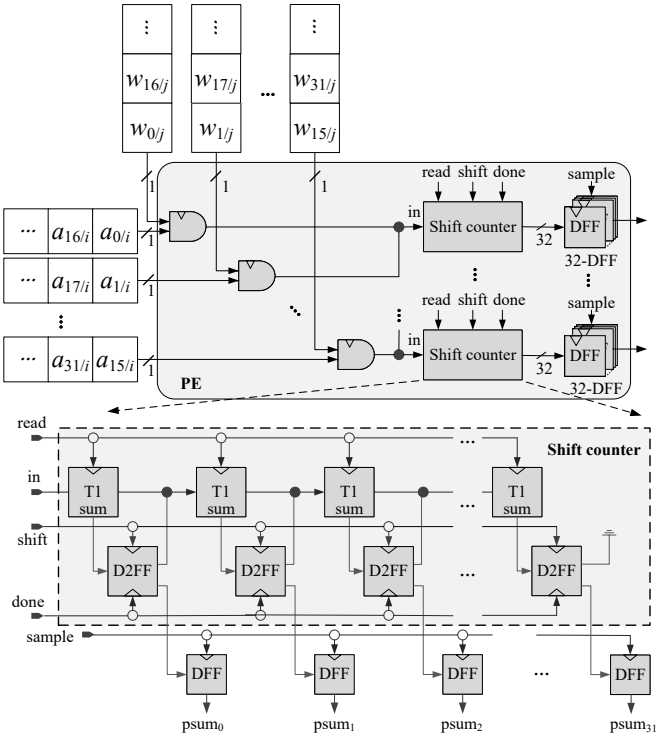

Figure 10. The proposed SFQ-based 16-bit bit-serial PE with OS dataflow, where the Shift counter circuit is designed for partial products accumulation.

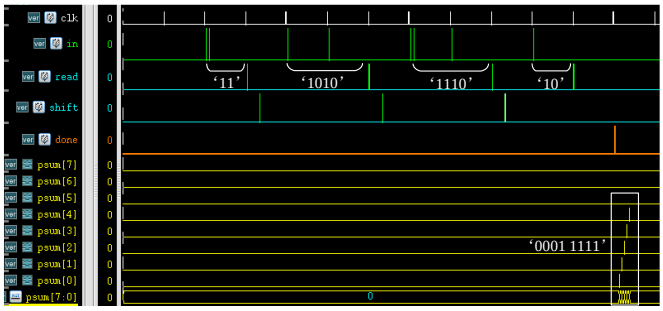

Figure 11. Simulation result of the Shift counter.

of *read* and *shift* signals are required. This process ensures that the counter value is reset to the updated psum, enabling the counting of partial products for the subsequent bit position. We adhere to the traversal order of bit positions i and j as outlined in Section 2.2. Once the counting of positive or negative psums concludes, the *read* and *done* signals store the psum in a register composed of 32 DFFs.

A 16-bit PE comprises 8 Shift counters. The pulse sequences from two AND gates are processed through a CB gate and then input into each Shift counter. The *sample* signal transfers the psums from the 8 Shift counters to a bit-slice adder for accumulation. To verify the functionality of the Shift counter, we first create its layout using Cadence Virtuoso, extract the netlist, and simulate it using Synopsys VCS. For example, when using the proposed PE to execute the unsigned MAC 111×011+101×010, after passing through the AND and CB gates, the top Shift counter receives the input sequence 111010111010 at its *in* port, as illustrated in Fig. 11. The *done* signal is asserted at the 11th clock cycle, resulting in the correct psum of 00011111.

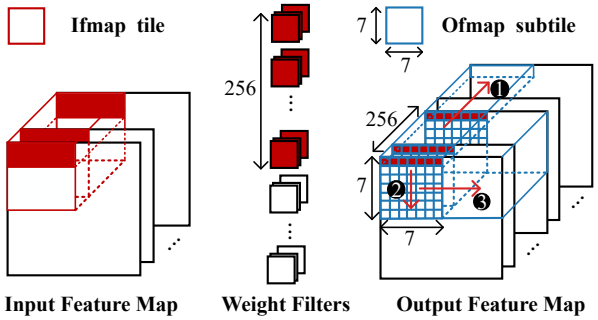

Figure 12. Data partitioning and scheduling of the convolutional layer.

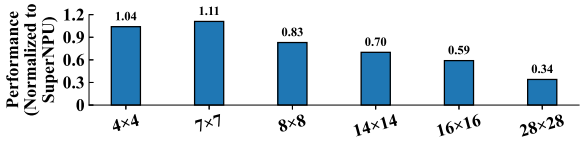

Figure 13. JBSA's performance with various spatial sizes of the ofmap tile, normalized to SuperNPU.

4.3 Work distribution model design

4.3.1 Data partitioning scheme. We employ a data partitioning scheme for ifmaps and ofmaps as shown in Fig. 12. For any given convolutional layer, we first partition the ofmap along the spatial dimension. Each segment is designated as a *tile*, with a specific spatial size. Selecting an appropriate tile size is crucial, as a small tile size can lead to redundant off-chip memory access, while a large tile size would cause excessive ineffectual computation. Moreover, given the prevalent use of 224×224 images as inputs in DNNs, it is observed that the majority of feature map sizes are multiples of 7. Consequently, adopting a tile size that is also a multiple of 7 can achieve evenly partitioned feature maps. We evaluate the impact of different tile sizes on performance, including options that are powers of 2 and multiples of 7, as shown in Fig. 13. In this work, we select the performance-optimal 7×7 tile size. Each tile is further partitioned into multiple portions, called *subtiles*, along the channel dimension. Each subtile includes 256 channels corresponding to the width of the PE array.

Simultaneously, we partition the ifmap along the spatial dimension into tiles. The size of each ifmap tile depends on the volume of ifmap data required to compute a single ofmap tile, which is determined by the spatial size of the ofmap tile, the convolutional kernel's size, and the stride. For example, with a 3×3 kernel and a stride of 1, the spatial size of the ifmap tile would be 9×9, spanning all channels.

The ifmap buffer chunk is divided into 36 buffer tiles, each formed by 7 buffer rows (shift registers). Storage configurations vary by kernel type. For 1×1 kernel, each buffer tile

stores a single ifmap tile row, and each shift register holds all channels per spatial position within the row. For larger kernels, each buffer tile stores an entire ifmap tile, and each shift register holds all channels per row. A buffer tile does not need to store the entire ifmap tile data simultaneously.

Correspondingly, we partition the PE array along the height dimension, with a set of 7 rows of PEs forming a PE tile, resulting in a total of 36 PE tiles. Each PE tile is tasked with computing one ofmap tile and receives data corresponding to the ifmap tile. Consequently, JBSA can compute 36 ofmap tiles concurrently.

4.3.2 Data scheduling strategy. In our approach, the pixels of a row in an ofmap subtile are concurrently mapped onto 7 PEs of the corresponding PE tile. The ifmap buffer tile only stores the necessary data for computation. Once the computation is done, the same row from the subsequent subtile of the same ofmap tile is mapped onto the PE tile. Since the ifmap data required for both mappings is identical, this avoids extra off-chip memory access.

Once the computation of the same row from different subtiles of the same ofmap tile is finalized, the PE tile proceeds to compute the succeeding row of the ofmap tile. Due to the data reuse property of the convolutional layer, the ifmap data required for computing different rows of the ofmap tile partially overlaps. In this scenario, the ifmap buffer tile only needs to load the distinct portion of ifmap data, thereby avoiding extra off-chip memory access. Those outdated ifmap data, which will not be utilized in subsequent computations, can be replaced.

The computation sequence of the ofmap is as follows: Initially, compute ofmap pixels of the first row from different subtiles of an ofmap tile (Fig. 12 ❶). Then, compute the second row from different subtiles of the same ofmap tile (Fig. 12 ❷), and continue in this manner until all 7 rows of this ofmap tile have been computed. Thereafter, transition to the next ofmap tile (Fig. 12 ❸).

4.3.3 Dispatcher. The *Dispatcher* serves two functions: timing alignment and data dispatch. Each dispatcher is tasked with dispatching data from an ifmap buffer tile to the corresponding PE tile. Therefore, there are 36 Dispatchers in total, each corresponding to one of the 36 PE tiles. Fig. 14 illustrates the structure of the Dispatcher and provides an example of a 3×3 convolution with a stride of 1. To compute 7 pixels of the first row of an ofmap subtile, the first three rows of the ifmap tile are stored in the first three shift registers of the buffer tile (Fig. 14 ❶).

Timing alignment: Firstly, each shift register is connected with at least 6 cascaded DFFs to regulate the timing of data arrival. The number of cascaded DFFs for each shift register increases by 7 gradually across different Dispatchers, since weights require 7 cycles to propagate down to the succeeding PE tile. Additionally, some DFFs should be bypassed depending

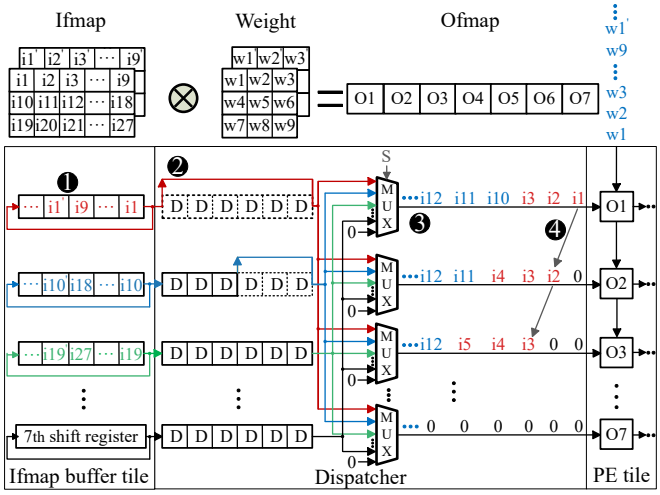

Figure 14. Dispatcher with a working example.

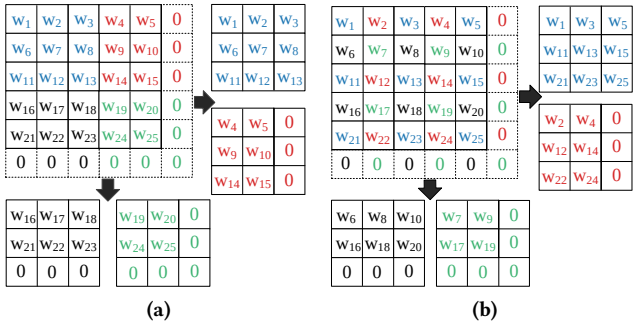

Figure 15. The partition of a 5×5 convolutional kernel using (a) stride = 1 and (b) stride = 2, respectively.

on the size of the convolutional kernel. For example, in a 3×3 convolution, the ifmap data of the first row must arrive 3 cycles prior to the second row and 6 cycles prior to the third row at the PEs. Therefore, the first row's 6 DFFs should be bypassed, and the second row's 3 DFFs should be skipped (Fig. 14 ❷).

Data dispatch: After passing through the cascaded DFFs, data of each shift register is sent to 7 rows of PEs through splitter trees. Each row of PEs is equipped with an 8:1 MUX that selects the required data from the 7 shift registers or 0 (Fig. 14 ❸). The selection signal depends on the convolutional kernel size, stride, and row index of the ofmap pixels mapped onto the PE tile, following a regular pattern. For example, in a 3×3 convolution, the first 3 pixels of the ifmap data in the convolutional window are selected first, succeeded by the next 3 pixels of the second row, then the next 3 pixels of the third row, and back to the first row's next 3 pixels, repeating this pattern until the computation is complete. Due to the systolic nature of the PE array, the selection signal originates from the top of the PE array and propagates downward (Fig. 14 ❹).

The Dispatcher supports both 1×1 and 3×3 convolutional kernels. Larger convolutional kernels can be partitioned into multiple 3×3 convolutional kernels. For example, a 5×5 kernel can be padded with an additional row and an extra column of

Table 1. Diverse architectural configurations investigated for SFQ-based bit-serial DNN accelerators.

Configuration	JBSA	Config1	Config2	Config3	Config4	Config5	Config6	Config7	Config8	Config9	Config10
PE array height	252	252	504	504	126	252	252	252	252	252	252
PE array width	256	64	512	128	512	256	256	256	256	256	256
Number of 4-bit adders	504	504	2016	504	504	504	504	504	504	504	504
Ifmap buffer size	2.2 MB	2.2 MB	4.4 MB	2.2 MB	2.2 MB	1.1 MB	4.4 MB	2.2 MB	2.2 MB	2.2 MB	2.2 MB
Ifmap buffer partition	128	128	128	64	256	64	256	128	128	64	256
Weight buffer size	4.5 MB	4.5 MB	4.5 MB	4.5 MB	4.5 MB	4.5 MB	4.5 MB	2.25 MB	9 MB	4.5 MB	4.5 MB
Weight buffer partition	128	512	64	256	64	128	128	64	256	64	256

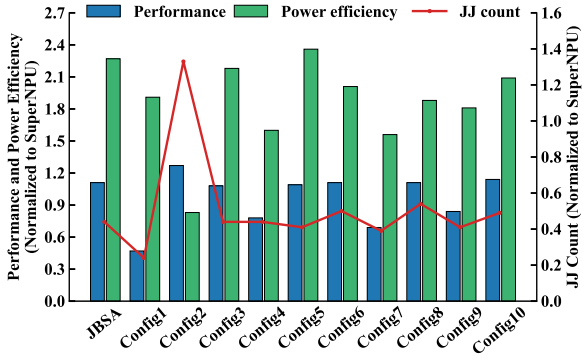

Figure 16. Performance, power efficiency and JJ count comparison of different architectural configurations for SFQ-based bit-serial DNN accelerators.

zeros, and subsequently partitioned into four 3×3 kernels as shown in Fig. 15. The number of pixels in a single spatial row of an ifmap tile is 9 for a 3×3 kernel, which aligns with the pixel count of the kernel as shown in Fig. 14. Consequently, this alignment significantly reduces ineffective computations and bubble insertions for filling the pipeline. This elucidates one of the reasons behind the performance-optimality of the 7×7 ofmap tile size.

4.4 Design parameter selection

We conduct a comprehensive exploration of the design space of the SFQ-based bit-serial DNN accelerator architecture to elucidate the rationale behind the selection of architecture parameters in JBSA. Specifically, four types of parameters are analyzed, including computational resource quantity, PE array shape, buffer size, and buffer partition degree. By varying these parameters, we create 11 distinct architectural configurations, as summarized in TABLE 1. We evaluate the performance, power efficiency, and JJ count of these configurations (Fig. 16).

Initially, we investigate the impact of different computational resource quantities by comparing JBSA, Config1, and Config2. Theoretically, a larger PE array size contributes to improved performance, albeit with increased JJ consumption. To this end, we start with Config1, which possesses a PE array size similar to that of SuperNPU, and proceed to enlarge the PE array size by four times to attain JBSA, resulting in a significant 2.4× performance enhancement with only a modest 87% increase in JJ count. In contrast, Config2, with four times the computational resources of JBSA, yields only a 14% performance improvement, while exhibiting a JJ count 3× that of JBSA and a power efficiency of merely 37% compared to JBSA. This observation indicates that the gains in performance diminish as the PE array size becomes excessively large.

Next, we delve into the influence of various PE array shapes. Despite possessing identical computational resource quantities and buffer capacities, JBSA outperforms both Config3 and Config4 in terms of performance and power efficiency, highlighting the importance of the specific PE array shape.

Subsequently, we analyze the effect of different ifmap buffer sizes by comparing JBSA, Config5, and Config6. Despite Config6's ifmap buffer size being twice that of JBSA, it fails to demonstrate performance improvement. This is because JBSA's ifmap buffer capacity adequately stores all the required ifmap data for computing concurrently mapped ofmap pixels within the PE array, thereby eliminating the need for additional off-chip memory access. Consequently, increasing the buffer capacity would merely escalate JJ consumption and reduce power efficiency. Contrarily, Config5, equipped with an ifmap buffer capacity of only 1.1 MB, faces buffer capacity limitations during the execution of certain DNN layers, leading to a 3% performance reduction compared to JBSA. Nevertheless, Config5 exhibits 7% fewer JJ counts and 4% higher power efficiency than JBSA. Considering performance and scalability, we opt for a 2.2 MB ifmap buffer size for JBSA.

Similarly, we explore the impact of different weight buffer sizes by comparing JBSA, Config7, and Config8. Notably, Config7, with half the weight buffer size of JBSA, experiences a substantial 38% performance decline. The reduction is attributed to the insufficiency of weight buffer capacity, which results in frequent off-chip memory access due to the high reusability of weights in convolutions. Conversely, JBSA's weight buffer effectively stores all weight filters, making the larger weight buffer in Config8 practically unbeneficial.

Moreover, the shift register lengths within the buffers in the aforementioned configurations remain consistent, resulting in equivalent data movement overheads. Lastly, we investigate

Table 2. Hardware specifications for the proposed JBSA and comparative accelerators.

	SuperNPU	Baseline	JBSA	Loom@300K	CryoLoom@77K
Dataflow	WS	OS	OS	OS	OS
PE array height	256	256	252	16	16
PE array width	64	256	256	128	128
PE bit width	16-bit	16-bit	16-bit	16-bit	16-bit
Ifmap buffer size	24 MB	24 MB	2.2 MB	1 MB	1 MB
Weight buffer size	128 KB	4.5 MB	4.5 MB	2 MB	2 MB
Ofmap buffer size	24 MB	252 KB	252 KB	8 KB	8 KB
Frequency (GHz)	23.3	23.3	23.3	1	2.2
Peak perf. (TMAC/s)	382	\	94	0.13	0.28
RSFQ-based power w.o.c.	1108 W	\	545 W	\	\

the influence of different shift register lengths, corresponding to varying buffer partition degrees. A comparative analysis involving JBSA, Config9, and Config10 reveals that finer partitioning leads to shorter shift register lengths, thereby contributing to enhanced performance. However, Config10's performance improvement is restricted, with only a marginal 2% advantage over JBSA. Ultimately, JBSA attains the highest power efficiency, surpassing Config10 by 9%.

5 Evaluation

This section evaluates JBSA with three DNN accelerators, SuperNPU, Loom, and the cryogenic-CMOS-based implementation of Loom. The evaluation methodology is first described, then JBSA is evaluated on performance, power efficiency, and JJ overhead.

5.1 Evaluation methodology

In this work, we simulate JBSA using the SIMIT fabrication process [13]. We use Cadence Virtuoso to generate the layout of JBSA and extract the netlist from the layout, which is simulated using Synopsys VCS. Finally, JBSA achieves a maximum operating frequency of 23.3 GHz, bounded by on-chip buffers with feedback loops. The bit width of PE is set to 16 bits. A comparative analysis is performed on the performance and power efficiency of JBSA against three distinct DNN accelerator implementations, including:

- **SuperNPU**. Due to the unavailability of the AIST 1.0 μm fabrication process technology used by SuperNPU, we utilize the SIMIT fabrication process to implement SuperNPU, ensuring a fair comparison. The bit width of PE in SuperNPU is set to 16 bits. The maximum operating frequency of the implemented SuperNPU is also 23.3 GHz.
- **Loom@300K**. Loom is implemented by standard integrated circuit design tools with the 45 nm TSMC CMOS process. The maximum operating frequency reported by the Synopsys Design Compiler (DC) is 1 GHz.
- **CryoLoom@77K**. Research has shown that CMOS designs operating at cryogenic temperatures (e.g., 77 K) can enhance both performance and power efficiency, primarily due to reductions in leakage power and wire resistance at cryogenic temperatures [6]. To implement the CryoLoom design running at 77 K, we utilize an open-source 45 nm low-temperature library [17] to synthesize it. To show the potential of cryogenic computing, we apply voltage scaling to CryoLoom, reducing V_{dd} and V_{th} to 0.75 V and 0.25 V, respectively, as suggested in [6]. The reported maximum frequency by DC is 2.2 GHz.

Table 3. Setup of six DNN models cited from [36].

Network	Data Precisions of Convolutional Layers		Batch Size
	Ifmap / Per Layer	Weight	
AlexNet	9-8-5-5-7	11	30
GoogLeNet	10-8-10-9-8-10-9-8-9-10-7	11	30
NiN	8-8-8-9-7-8-8-9-9-8-8-8	11	30
VGG19	12-12-12-11-12-10-11-11-13-12-13-13-13-13-13-13	12	7
VGGM	7-7-7-8-7	12	30
VGGS	7-8-9-7-9	12	30

The hardware specifications of the accelerators above are summarized in TABLE 2, where "w.o.c." denotes "without cooling cost". To model these accelerators, we develop a simulator based on SCALE-Sim [32]. Across all designs, the off-chip memory bandwidth is set to 300 GB/s, equivalent to the HBM value used by SuperNPU [18]. To calculate the JJ overhead and power consumption of JBSA and SuperNPU, we utilize the data provided for each SFQ gate in [33][1]. The cooling cost at 4 K and 77 K is assumed to be 400× [15] and 9.65× [20] of the design's power consumption, respectively. We choose six DNN models, namely AlexNet, GoogLeNet, NiN, VGG19, VGGM, and VGGS, as our workloads. Using the lower effectual precision attributes inherent in bit-serial computation, we employ the profile-derived expected precision of ifmaps for each convolutional layer and a shared weight precision across all convolutional layers, as described in [36]. This approach is applied to bit-serial-based designs, namely JBSA, Loom@300K, and CryoLoom@77K. For bit-parallel SuperNPU, a uniform data precision of 16 bits is employed for all convolutional layers. To evaluate the performance of the four designs, we conduct batch-based inferences. The precision levels and batch sizes used in the evaluation are summarized in TABLE 3.

5.2 Performance evaluation

The speedup of JBSA is evaluated using the normalized throughput (TMAC/s) relative to Loom@300K. As depicted in Fig. 17, JBSA consistently outperforms Loom@300K (CryoLoom@77K) across all DNN models, achieving speedups of 316× (144×) on average and 810× (362×) in VGGS. This is mainly due to the high-frequency characteristics of SFQ logic and a significantly larger PE array of JBSA. Fig. 17 also demonstrates that JBSA exhibits a performance improvement of 11% compared to SuperNPU. This enhancement attributes to three factors.

[1]The JJ counts of T1 and D2FF are from [28] and [48], respectively, while their static and dynamic powers are calculated using the method in [27].

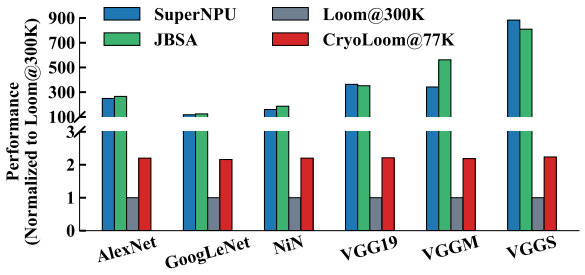

Figure 17. Normalized performance comparison of JBSA, SuperNPU, Loom@300K, and CryoLoom@77K.

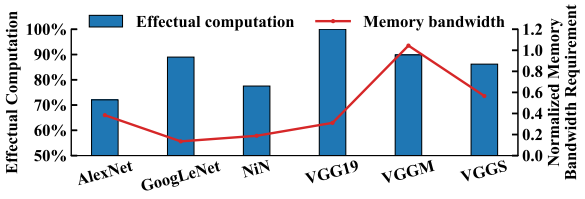

Figure 18. JBSA's effectual computation and off-chip memory bandwidth requirement (normalized to SuperNPU).

Table 4. Power efficiency normalized to Loom@300K.

	RSFQ-based JBSA	ERSFQ-based JBSA	RSFQ-based SuperNPU	ERSFQ-based SuperNPU	Loom @300K	CryoLoom @77K
w.o.c.	33	491	14.56	215	1	4.87
w.c.	0.08	1.23	0.04	0.54	1	0.46

Firstly, architectural optimizations in JBSA increase the effectual MAC operations to an average of 85.3%, as shown in Fig. 18. This percentage is 112× higher than the Baseline described in Section 3.4. Secondly, the precision reduction of bit-serial technology leads to a lower off-chip memory bandwidth requirement, which is only 35% of what SuperNPU requires, as shown in Fig. 18. Additionally, the larger PE array in JBSA contributes to its improved performance. In terms of maximum speedup, JBSA achieves a 1.6× improvement over SuperNPU in VGGM, primarily due to the lower effective precision of VGGM. For other DNN models, JBSA's performance generally remains on par with that of SuperNPU.

5.3 Power efficiency evaluation

We evaluated JBSA's power efficiency using two SFQ device technologies, rapid SFQ (RSFQ) and ERSFQ. These two technologies differ in terms of power consumption characteristics. In RSFQ logic, static power dominates the total power consumption, while ERSFQ has zero static power and dynamic power is assumed to be twice that of RSFQ [18]. The power efficiency for JBSA, SuperNPU, and the CMOS-based designs is shown in TABLE 4, where "w.c." denotes "with cooling cost". JBSA outperforms SuperNPU in power efficiency, with an average improvement of 2.3× (2.3×), and up to 3.3× (3.4×) in VGGM with RSFQ (ERSFQ) technology. This enhancement is attributed to the smaller on-chip buffer capacity of JBSA and its significantly higher utilization ratio of the ifmap buffer, as

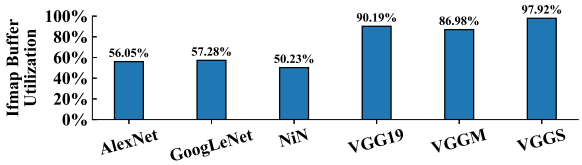

Figure 19. JBSA's ifmap buffer utilization.

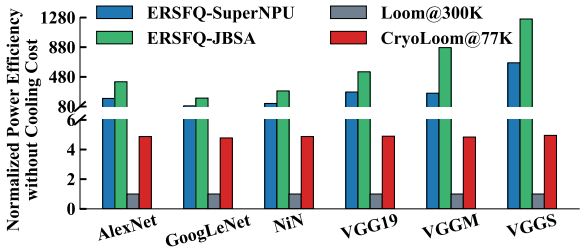

Figure 20. Power efficiency comparison of ERSFQ-based JBSA, ERSFQ-based SuperNPU, Loom@300K, and CryoLoom@77K considering no cooling cost, normalized to Loom@300K.

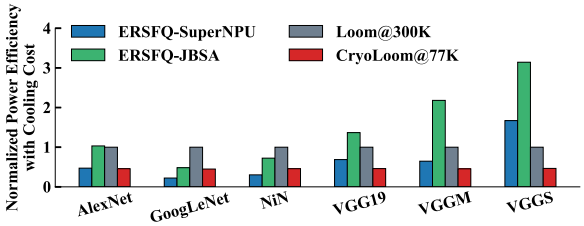

Figure 21. Power efficiency comparison of ERSFQ-based JBSA, ERSFQ-based SuperNPU, Loom@300K, and CryoLoom@77K considering cooling cost, normalized to Loom@300K.

shown in Fig. 19, which is 70.6% on average and 5.6× higher than that of SuperNPU. In comparison to CMOS-based accelerators, the power efficiency of RSFQ-based JBSA is 33× (6.78×) higher than Loom@300K (CryoLoom@77K) when considering free cooling. However, when accounting for the cooling cost, the normalized power efficiency decreases to 0.08 (0.18), which is deemed unacceptable due to the substantial static power dissipation of RSFQ logic. Conversely, ERSFQ-based JBSA achieves an average power efficiency that is 491× (100.86×) higher than Loom@300K (CryoLoom@77K), considering free cooling. In VGGS, the maximum power efficiency improvement reaches 1260× (254×) over Loom@300K (CryoLoom@77K), as depicted in Fig. 20. Even when factoring in the cooling cost, the normalized power efficiency of ERSFQ-based JBSA still attains 1.23× (2.68×) higher than Loom@300K (CryoLoom@77K) on average, as shown in Fig. 21.

Fig. 21 indicates that JBSA's power efficiency varies across different model architectures, particularly excelling with large-scale workloads. This is attributed to three key factors: 1) PE array scale. The scale of JBSA's PE array is four times that of SuperNPU. As the workload increases, the likelihood of fully utilizing the PE array also rises, leading to enhanced PE efficiency. 2) Dataflow limitation. SuperNPU's use of WS dataflow becomes less advantageous with larger workloads, as

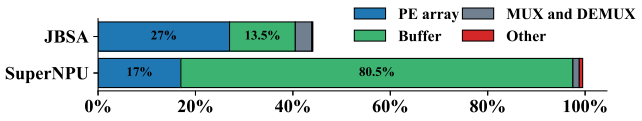

Figure 22. The JJ count breakdown of SuperNPU and JBSA.

it requires substantial data reuse and larger buffers. In contrast, JBSA's data reuse is contingent upon the size of the convolution filter, which is controllable, allowing for significantly smaller buffer sizes. 3) Memory access impact. Larger workloads result in greater memory access requirements. JBSA leverages the variable precision characteristics of data at each layer of the neural network model to compress the precision of memory access data. This effectively mitigates the significant disparity between computation speed and memory access speed, resulting in higher PE utilization.

5.4 JJ count evaluation

To unveil the area efficiency of JBSA, we conducted an in-depth analysis of the JJ composition between JBSA and SuperNPU. We implement gate-level RTL netlists for both of them and determine the JJ count based on the number of gates in the netlists. We include the PE array, three on-chip buffers, MUX and DEMUX trees, as well as other on-chip modules shown in Fig. 9. We also account for the JJ count of splitters and DFFs for fan-out and path balancing. Additionally, we integrate PTL transmitters and receivers with each SFQ logic gate to calculate the JJ count after placement and routing. Fig. 22 shows the JJ count breakdown of both designs. As the figure clearly indicates, JBSA exhibits a remarkable 56% reduction in JJ overhead compared to SuperNPU. This reduction primarily stems from JBSA's smaller on-chip buffer capacity, which is 6.9× smaller than that of SuperNPU. Specifically, the JJ count of JBSA's on-chip buffer accounts for only 31% of the total JJ count, while the JJ count of SuperNPU's on-chip buffer dominates the total JJ overhead, comprising 80% of it. Furthermore, the JJ count of the PE array in JBSA is only 1.6× larger than that of SuperNPU's PE array, considering that the size of JBSA's PE array is approximately 4× larger than that of SuperNPU's PE array. This is primarily due to the fact that a single PE in JBSA consumes fewer JJs than SuperNPU's PE, specifically only 41% of the number used in SuperNPU's PE.

6 Related Work

Superconducting neural network accelerators. Research on superconducting neural network (NN) accelerators has primarily followed three distinct approaches: integrating superconducting technology into traditional frameworks, exploring unconventional computing paradigms, and developing neuromorphic systems. The first approach involves incorporating superconducting technology into conventional architectures, such as hybrid SFQ-CMOS memory architectures [49] and binarized neural network accelerators utilizing novel SFQ counters [11]. The second approach delves into unconventional computing paradigms, including stochastic computing accelerators based on adiabatic quantum flux parametron logic [7], temporal logic for bioinformatics algorithms [39, 40], and unary SFQ encoding for signal processing [14]. A more challenging endeavor is to develop superconducting neuromorphic systems, such as spiking NN and novel neuron devices [16, 34]. Unlike previous studies, this paper focuses on the SFQ bit-serial NN accelerator, achieving superior performance and power efficiency compared to the state-of-the-art SFQ NN accelerator and CMOS designs, even with the cooling penalty.

CMOS bit-serial neural network accelerators. A lot of researchers have proposed bit-serial CMOS NN accelerators for power-efficient processing. A representative approach is to exploit effective bits, specifically non-zero bits, in activation or weight data [1, 25]. Another attempt is to employ a compression technique to reduce the precision of activations in each convolutional layer to achieve superior throughput [19]. Sharify et al. introduced Loom, which leverages precision reduction and bit-serial processing for both activations and weights [36]. Another bit-serial PE design based on a lookup table was proposed in [21]. However, these studies investigate only CMOS-based NN accelerator architectures. No prior work designs a bit-serial architecture for SFQ DNN accelerators at the 4 K temperature. Our target is to explore and optimize the bit-serial architecture of the SFQ-based DNN accelerator and demonstrate its potential in performance and power efficiency.

7 Conclusion

It is evident that the explosive growth of AI conflicts with the end of Moore's Law, making the search for CMOS alternative technologies a major challenge. This paper employs superconducting single flux quantum technology in the design of DNN accelerators and explores novel circuit architectures and computational paradigms to address design incompatibilities. The proposed JBSA adopts a bit-serial architecture and an innovative data scheduling scheme, offering advantages such as low resource consumption and reduced memory bandwidth requirements. Furthermore, evaluations demonstrate that JBSA achieves significantly higher energy efficiency and performance compared to the state-of-the-art SFQ-based and CMOS-based solutions. JBSA has the potential to serve as a design paradigm for DNN accelerators based on emerging circuit technologies.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No.62302477), and the CAS Project for the Youth Innovation Promotion Association.

References

[1] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. 2017. Bit-Pragmatic Deep Neural Network Computing. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 382–394.

[2] Paul Alcorn. 2017. Hot Chips 2017: A Closer Look At Google's TPU v2. http://www.tomshardware.com/news/tpu-v2-google-machine-learning,35370.html.

[3] Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, and Jörg Henkel. 2022. Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey. *Comput. Surveys* 55, 4, Article 83 (2022), 36 pages. https://doi.org/10.1145/3527156

[4] Parhami Behrooz. 2000. Computer Arithmetic: Algorithms and Hardware Designs. *Oxford University Press* 19 (2000), 512583–512585.

[5] Ali Bozbey, Mustafa Altay Karamuftuoglu, Sasan Razmkhah, and Murat Ozbayoglu. 2020. Single Flux Quantum Based Ultrahigh Speed Spiking Neuromorphic Processor Architecture. *arXiv preprint arXiv:1812.10354* (2020).

[6] Ilkwon Byun, Dongmoon Min, Gyu-hyeon Lee, Seongmin Na, and Jangwoo Kim. 2020. CryoCore: A Fast and Dense Processor Architecture for Cryogenic Computing. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 335–348. https://doi.org/10.1109/ISCA45697.2020.00037

[7] 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 *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 567–578. https://doi.org/10.1145/3307650.3322270

[8] Ke Chen, Yue Gao, Haroon Waris, Weiqiang Liu, and Fabrizio Lombardi. 2023. Approximate Softmax Functions for Energy-Efficient Deep Neural Networks. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 31, 1 (2023), 4–16. https://doi.org/10.1109/TVLSI.2022.3224011

[9] Ran Cheng, Uday S. Goteti, and Michael C. Hamilton. 2019. Superconducting Neuromorphic Computing Using Quantum Phase-Slip Junctions. *IEEE Transactions on Applied Superconductivity* 29, 5 (2019), 1–5. https://doi.org/10.1109/TASC.2019.2892111

[10] Mohammed E. Elbtity, Peyton S. Chandarana, Brendan Reidy, Jason K. Eshraghian, and Ramtin Zand. 2022. APTPU: Approximate Computing Based Tensor Processing Unit. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 12 (2022), 5135–5146. https://doi.org/10.1109/TCSI.2022.3206262

[11] Rongliang Fu, Junying Huang, Haibin Wu, Xiaochun Ye, Dongrui Fan, and Tsung-Yi Ho. 2022. JBNN: A Hardware Design for Binarized Neural Networks Using Single-Flux-Quantum Circuits. *IEEE Trans. Comput.* 71, 12 (2022), 3203–3214. https://doi.org/10.1109/TC.2022.3215085

[12] Kris Gaj, Eby G Friedman, and Marc J Feldman. 1997. Timing of Multi-Gigahertz Rapid Single Flux Quantum Digital Circuits. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology* 16 (1997), 247–276. https://doi.org/10.1023/A:1007903527533

[13] Xiaoping Gao, Qi Qiao, Mingliang Wang, Minghui Niu, Huanli Liu, Masaaki Maezawa, Jie Ren, and Zhen Wang. 2021. Design and verification of SFQ cell library for superconducting LSI digital circuits. *IEEE Transactions on Applied Superconductivity* 31, 5 (2021), 1–5. https://doi.org/10.1109/TASC.2021.3062570

[14] Patricia Gonzalez-Guerrero, Meriam Gay Bautista, Darren Lyles, and George Michelogiannakis. 2022. Temporal and SFQ Pulse-Streams Encoding for Area-Efficient Superconducting Accelerators. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 963–976. https://doi.org/10.1145/3503222.3507765

[15] D Scott Holmes, Andrew L Ripple, and Marc A Manheimer. 2013. Energy-Efficient Superconducting Computing—Power Budgets and Requirements. *IEEE Transactions on Applied Superconductivity* 23, 3 (2013), 1701610–1701610. https://doi.org/10.1109/TASC.2013.2244634

[16] Junying Huang, Rongliang Fu, Xiaochun Ye, and Dongrui Fan. 2022. A survey on superconducting computing technology: circuits, architectures and design tools. *CCF Transactions on High Performance Computing* 4 (2022), 1–22. https://doi.org/10.1007/s42514-022-00089-w

[17] HPCS Lab in Seoul National University. 2023. CryoModel. https://github.com/SNU-HPCS/CryoModel.

[18] Koki Ishida, Ilkwon Byun, Ikki Nagaoka, Kosuke Fukumitsu, Masamitsu Tanaka, Satoshi Kawakami, Teruo Tanimoto, Takatsugu Ono, Jangwoo Kim, and Koji Inoue. 2020. SuperNPU: An Extremely Fast Neural Processing Unit Using Superconducting Logic Devices. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 58–72. https://doi.org/10.1109/MICRO50266.2020.00018

[19] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-Serial Deep Neural Network Computing. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783722

[20] Gyu-Hyeon Lee, Dongmoon Min, Ilkwon Byun, and Jangwoo Kim. 2019. Cryogenic Computer Architecture Modeling with Memory-Side Case Studies. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 774–787. https://doi.org/10.1145/3307650.3322219

[21] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun Yoo. 2019. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision. *IEEE Journal of Solid-State Circuits* 54, 1 (2019), 173–185. https://doi.org/10.1109/JSSC.2018.2865489

[22] Yuan Li, Ahmed Louri, and Avinash Karanth. 2022. SPACX: Silicon Photonics-Based Scalable Chiplet Accelerator for DNN Inference. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 831–845. https://doi.org/10.1109/HPCA53966.2022.00066

[23] 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 (1991), 3–28. https://doi.org/10.1109/77.80745

[24] Evan McKinney, Mingkang Xia, Chao Zhou, Pinlei Lu, Michael Hatridge, and Alex K Jones. 2023. Co-Designed Architectures for Modular Superconducting Quantum Computers. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 759–772. https://doi.org/10.1109/HPCA56546.2023.10071036

[25] Andreas Moshovos, Jorge Albericio, Patrick Judd, Alberto Delmas, Sayeh Sharify, Mostafa Mahmoud, Tayler Hetherington, Milos Nikolic, Dylan Malone Stuart, Kevin Siu, Zissis Poulos, Tor Aamodt, and Natalie Enright Jerger. 2018. Identifying and Exploiting Ineffectual Computations to Enable Hardware Acceleration of Deep Learning. In *IEEE International New Circuits and Systems Conference (NEWCAS)*. 356–360. https://doi.org/10.1109/NEWCAS.2018.8585656

[26] Oleg A. Mukhanov. 2011. Energy-Efficient Single Flux Quantum Technology. *IEEE Transactions on Applied Superconductivity* 21, 3 (2011), 760–769. https://doi.org/10.1109/TASC.2010.2096792

[27] Thomas Ortlepp, Olaf Wetzstein, Sonja Engert, Juergen Kunert, and Hannes Toepfer. 2011. Reduced Power Consumption in Superconducting Electronics. *IEEE Transactions on Applied Superconductivity* 21, 3 (2011), 770–775. https://doi.org/10.1109/TASC.2011.2117410

[28] S.V. Polonsky, V.K. Semenov, and A.F. Kirichenko. 1994. Single Flux Quantum B Flip-Flop and Its Possible Applications. *IEEE Transactions on Applied Superconductivity* 4, 1 (1994), 9–18. https://doi.org/10.1109/77.273059

[29] G S Priyanka, M Venkatesan, and P Prabhavathy. 2023. Advancements in Quantum Machine Learning and Quantum Deep Learning: A Comprehensive Review of Algorithms, Challenges, and Future Directions. In *International Conference on Quantum Technologies, Communications, Computing, Hardware and Embedded Systems Security (iQ-CCHESS)*. 1–8. https://doi.org/10.1109/iQ-CCHESS56596.2023.10391745

[30] Nitin Rathi, Indranil Chakraborty, Adarsh Kosta, Abhronil Sengupta, Aayush Ankit, Priyadarshini Panda, and Kaushik Roy. 2023. Exploring Neuromorphic Computing Based on Spiking Neural Networks: Algorithms to Hardware. *Comput. Surveys* 55, 12, Article 243 (mar 2023), 49 pages. https://doi.org/10.1145/3571155

[31] Wojciech Romaszkan, Tianmu Li, and Puneet Gupta. 2022. SASCHA—Sparsity-Aware Stochastic Computing Hardware Architecture for Neural Network Acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 41, 11 (2022), 4169–4180. https://doi.org/10.1109/TCAD.2022.3197503

[32] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv preprint arXiv:1811.02883* (2018).

[33] L. Schindler and T. Hall. 2023. ColdFlux RSFQ Logic Cell Library for MIT-LL SFQ Process. https://github.com/sunmagnetics/RSFQlib. Version: 3.0.

[34] Michael Schneider, Emily Toomey, Graham Rowlands, Jeff Shainline, Paul Tschirhart, and Ken Segall. 2022. SuperMind: a survey of the potential of superconducting electronics for neuromorphic computing. *Superconductor Science and Technology* 35, 5 (2022), 053001. https://doi.org/10.1088/1361-6668/ac4cd2

[35] Michael L. Schneider, Christine A. Donnelly, Stephen E. Russek, Burm Baek, Matthew R. Pufall, Peter F. Hopkins, and William H. Rippard. 2017. Energy-efficient single-flux-quantum based neuromorphic computing. In *IEEE International Conference on Rebooting Computing (ICRC)*. 1–4. https://doi.org/10.1109/ICRC.2017.8123634

[36] Sayeh Sharify, Alberto Delmas Lascorz, Kevin Siu, Patrick Judd, and Andreas Moshovos. 2018. Loom: Exploiting Weight and Activation Precisions to Accelerate Convolutional Neural Networks. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465915

[37] Ourania Spantidi, Georgios Zervakis, Iraklis Anagnostopoulos, and Jörg Henkel. 2022. Energy-Efficient DNN Inference on Approximate Accelerators Through Formal Property Exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 41, 11 (2022), 3838–3849. https://doi.org/10.1109/TCAD.2022.3197522

[38] Guang-Ming Tang, Kensuke Takata, Masamitsu Tanaka, Akira Fujimaki, Kazuyoshi Takagi, and Naofumi Takagi. 2015. 4-bit Bit-Slice Arithmetic Logic Unit for 32-bit RSFQ Microprocessors. *IEEE Transactions on Applied Superconductivity* 26, 1 (2015), 1–6. https://doi.org/10.1109/TASC.2015.2507125

[39] Georgios Tzimpragos, Dilip Vasudevan, Nestan Tsiskaridze, George Michelogiannakis, Advait Madhavan, Jennifer Volk, John Shalf, and Timothy Sherwood. 2020. A Computational Temporal Logic for Superconducting Accelerators. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 435–448. https://doi.org/10.1145/3373376.3378517

[40] Georgios Tzimpragos, Jennifer Volk, Dilip Vasudevan, Nestan Tsiskaridze, George Michelogiannakis, Advait Madhavan, John Shalf, and Timothy Sherwood. 2021. Temporal Computing With Superconductors. *IEEE Micro* 41, 3 (2021), 71–79. https://doi.org/10.1109/MM.2021.3066377

[41] Yaman Umuroglu, Davide Conficconi, Lahiru Rasnayake, Thomas B Preusser, and Magnus Själander. 2019. Optimizing Bit-Serial Matrix Multiplication for Reconfigurable Computing. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12, 3 (2019), 1–24. https://doi.org/10.1145/3337929

[42] Di Wu, Jingjie Li, Ruokai Yin, Hsuan Hsiao, Younghyun Kim, and Joshua San Miguel. 2020. UGEMM: Unary Computing Architecture for GEMM Applications. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 377–390. https://doi.org/10.1109/ISCA45697.2020.00040

[43] Di Wu and Joshua San Miguel. 2022. uSystolic: Byte-Crawling Unary Systolic Array. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 12–24. https://doi.org/10.1109/HPCA53966.2022.00010

[44] Wm A Wulf and Sally A McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News* 23, 1 (1995), 20–24. https://doi.org/10.1145/216585.216588

[45] Haipeng Zha, Naveen Kumar Katam, Massoud Pedram, and Murali Annavaram. 2022. HiPerRF: A Dual-Bit Dense Storage SFQ Register File. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 415–428. https://doi.org/10.1109/HPCA53966.2022.00038

[46] Huilin Zhang, Chen Gang, Chen Xu, Guoliang Gong, and Huaxiang Lu. 2023. Brain-Inspired Spiking Neural Network Using Superconducting Devices. *IEEE Transactions on Emerging Topics in Computational Intelligence* 7, 1 (2023), 271–277. https://doi.org/10.1109/TETCI.2021.3089328

[47] Jiadi Zhu, Teng Zhang, Yuchao Yang, and Ru Huang. 2020. A comprehensive review on emerging artificial neuromorphic devices. *Applied Physics Reviews* 7, 1 (2020), 011312. https://doi.org/10.1063/1.5118217

[48] D. Zinoviev, P. Bunyk, A. Rylyakov, K. Likharev, and P. Litskevitch. 2023. SUNY RSFQ Cell Library. http://www.physics.sunysb.edu/Physics/RSFQ/Lib/index.html.

[49] Farzaneh Zokaee and Lei Jiang. 2021. SMART: A Heterogeneous Scratchpad Memory Architecture for Superconductor SFQ-Based Systolic CNN Accelerators. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 912–924. https://doi.org/10.1145/3466752.3480041