Graph Convolutional Network Acceleration Using Adiabatic Superconductor Josephson Devices

Zhengang Li
Northeastern University
Boston, USA
li.zhen@northeastern.edu

Hongwu Peng
University of Connecticut
Storrs, USA
hongwu.peng@uconn.edu

Xuan Shen
Northeastern University
Boston, USA
shen.xu@northeastern.edu

Masoud Zabihi
Northeastern University
Boston, USA
m.zabihi@northeastern.edu

Xi Xie
University of Connecticut
Storrs, USA
xi.xie@uconn.edu

Geng Yuan
University of Georgia
Athens, USA
geng.yuan@uga.edu

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

Olivia Chen
Kyushu University
Fukuoka, Japan
olivia.chen@ieee.org

Caiwen Ding
University of Minnesota Twin Cities
Minneapolis, USA
dingc@umn.edu

Abstract

Graph Convolutional Network (GCN) has gained popularity as it could lower the human expert's burden in making tactical real-time decisions. As Moore's law is reaching an end, the acceleration of the conventional GCN systems is limited. One promising alternative is the Adiabatic Quantum-Flux-Parametron (AQFP) superconducting computing as it can achieve extremely high energy efficiency compared to CMOS. In this paper, we propose an AQFP-aware GCN acceleration framework via co-optimizing AQFP hardware and GCN algorithms. More specifically, we first develop a regrowth-after- partitioning algorithm to enable the AQFP hardware parallelism and accelerate the aggregation computation while maintaining accuracy. Then, we propose two distinct AQFP-based architectures tailored specifically for each of the combination and aggregation stages. Furthermore, to unlock the extreme energy efficiency, we develop a hybrid binarized/low-bit GCN hardware/software co-design that can be efficiently executed on AQFP-based devices. Leveraging the AQFP randomized behavior, we adjust the AQFP buffer design to achieve multi-bit intermediate results and explore the bit-width at the output of the combination step. To mitigate the gap between the software model with hardware implementation, an AQFP logic-aware GCN Hybrid Quantization is proposed for the binarized GCN framework. Our framework demonstrates remarkable energy efficiency even when considering the additional cooling consumption. Specifically, compared with the representative FPGA-based framework GCoD [88], our framework achieves energy efficiency improvements of 1.9×10^4, 1.1×10^4, and 8.7×10^4 for the Cora, CiteSeer, and PubMed datasets, respectively, with a similar level of accuracy. To the best of our knowledge, this is the first attempt to implement an AQFP-based architecture specifically designed for GCNs.

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3730434

CCS Concepts

• **Hardware** → **Emerging technologies**; • **Computer systems organization** → **Architectures**; • **Computing methodologies** → **Machine learning**.

Keywords

GCN, AQFP, Quantization, BNN, Superconducting

ACM Reference Format:
Zhengang Li, Hongwu Peng, Xuan Shen, Masoud Zabihi, Xi Xie, Geng Yuan, Yanzhi Wang, Olivia Chen, and Caiwen Ding. 2025. Graph Convolutional Network Acceleration Using Adiabatic Superconductor Josephson Devices. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3721145.3730434

1 Introduction

As an emerging branch in deep learning research, graph neural networks (GNNs) aim to lower the human expert's burden in making tactical real-time decisions in applications, such as computer vision [70], traffic forecasting [44], autonomous systems [71], drug discovery [9], and social influence [33]. Recently, several GNN models have been developed to incorporate external features into graph structures, e.g., Graph Convolutional Networks (GCN) [49], GraphSAGE [35], Graph Isomorphism Networks (GIN) [84], and Graph Attention Networks [78]. GCN has become the most popular GNN among others due to its foundational architecture, effective performance on wide range of tasks [2, 36, 83, 92]. It comprises two primary phases - aggregation and combination, implicating iterative traversal of the graph nodes and edges.

In order to achieve high performance and energy efficiency for GCN systems, two research trends have emerged. The first one is graph input or weight sparsification algorithms that aim to reduce the computation and memory footprint [13, 14, 16, 89]. The second one is to address the workload imbalance caused by the irregularity of the graph inputs with highly unbalanced non-zero distributions according to AWB-GCN [28], EnGn [52], G-CoS [90] and [1, 5, 51].

As Moore's law is reaching an end [79], the potential for accelerating GCN systems using conventional Von-Neumann architecture remains limited. We are in urgent need of (i) a next-generation technology beyond CMOS, and (ii) the corresponding customized novel computing architectures for GCN accelerators to achieve ultra-high energy efficiency. One promising alternative is the Adiabatic Quantum-Flux-Parametron (AQFP) superconducting computing [15]. By leveraging magnetic flux quantization and quantum interference in Josephson-junction (JJ)-based superconductor loops, AQFP have emerged as promising candidates for future computing. Compared to state-of-the-art CMOS technology, AQFP can potentially achieve an energy-efficiency gain in the range of $10^4 \sim 10^5$ [15].

Despite the recent success of AQFP-based convolutional neural network (CNN) [50, 85], implementing GCNs on the AQFP superconducting computing platform presents unique challenges:

(i) *GCN computation complexity.* While the combination phase uses a computation pattern similar to CNN, the aggregation phase relies on the sparse and irregular graph structure. Such inherent irregularity necessitates specialized hardware designs to perform graph-related operations like aggregating neighbors and passing messages. The recent AQFP-based CNN frameworks, as evidenced by previous works such as [50, 85], predominantly emphasize dense matrix multiplication. This new emphasis presents a fresh challenge to existing AQFP-based frameworks.

(ii) *GCN mapping problem on AQFP devices.* Current accelerators, e.g., AWB-GCN [28], GROW [43], FlowGNN [67] and GNNAdvisor [82] process moderately sparse feature matrix (X) multiplication with a dense and small weight matrix (W), and then multiply the output with the highly sparse and irregular adjacency matrix (A). This enables a workload-efficient computation design that utilizes a unified SpMM (Sparse Matrix-Matrix Multiplication) engine. However, AQFP, as an emerging technique, currently has relatively limited scalability, and there is no AQFP-based architecture tailored for GNN to handle the large memory requirements from graph structures and node embeddings. Expansion of the AQFP scalability and efficient memory storage/access for sparse matrix become crucial for AQFP devices to deploy the GCN computation.

(iii) *The intermediate results prohibit the efficient mapping of GNN onto hardware.* We observe that a crucial challenge that remains unaddressed in GNN hardware design is the output of the combination step (producing a 32-bit intermediate result), even with binarized GNN frameworks [7, 81], where the weight and input feature matrices are binarized. This omission creates inefficiency and hinders the seamless mapping of binarized GCN frameworks to AQFP-based devices. In addition, (iv) *AQFP platforms present randomized behavior.* Because of the thermal noise and/or quantum fluctuation impact, the output of AQFP buffer presents randomized behavior when input current amplitude falls in a certain range, known as "gray-zone" ΔI_{in} [27]. In this case, it will be hard to detect the direction of the input current, resulting in a randomized output with a probability related to the input current, i.e., $0 < P(I_{in}) < 1$. This unique property is a double-sided sword that introduces inaccuracy but also makes it possible to be combined with stochastic computing.

To address the aforementioned escalating challenges, in this paper, we propose an AQFP-aware GCN acceleration framework via co-optimizing AQFP hardware and GCN algorithm. In addition, to unlock the extreme energy efficiency, we develop a GCN hybrid quantization software-and-hardware co-design that can be efficiently executed on AQFP-based devices. Even considering the cooling consumption for AQFP devices, we still achieve significant improvement in terms of energy efficiency while keeping good throughput and model accuracy, compared with the previous GCN acceleration frameworks on different hardware platforms. For instance, we achieve 1.35 times speedup and 6.1×10^3 times higher energy efficiency than REFLIP [42] ReRAM design on PubMed dataset. Our proposed framework surpasses FPGA-based and ASIC work by about four orders of magnitude, and ReRAM-based work by two to four orders of magnitude.

To the best of our knowledge, this paper is the first attempt to implement an AQFP-based architecture specifically

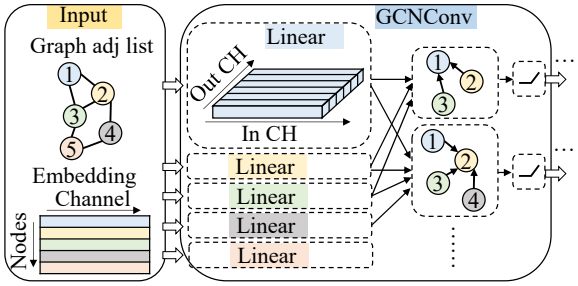

Figure 1: Single graph convolution layer computation.

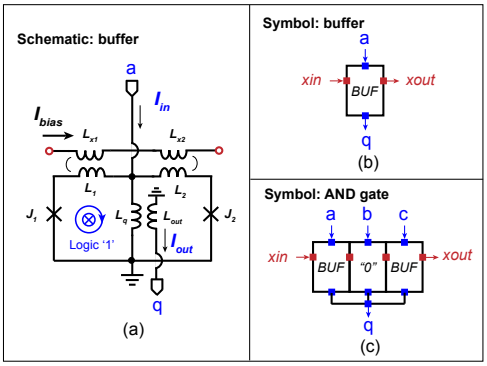

Figure 2: (a) Schematic of an AQFP buffer. (b) Symbol view of an AQFP buffer. (c) Symbol view of an AQFP AND gate, consisting of 3 AQFP buffers and a 3-to-1 branch.

designed for GNNs to manage various types of computation. We summarize our contributions as follows:

- We propose AQFP-based architecture tailored for GCN computation considering both the combination and the aggregation stages.
- A regrowth-after-partitioning algorithm is developed to divide large graphs into sub-graphs, to enable the AQFP hardware parallelism while maintaining accuracy.
- We propose an AQFP logic-aware GCN hybrid quantization to handle the intermediate result and AQFP randomized behavior problems in our framework.
- Hardware-and-software co-design. Adjusting the AQFP buffer configuration to achieve multi-bit output and fully leverage its randomized behavior.

Overall, compared with the representative FPGA-based framework GCoD [88], our framework achieves energy efficiency improvements of 4 orders of magnitude for the Cora, CiteSeer, and PubMed datasets with a similar level of accuracy, even considering the additional cooling consumption.

2 Background and Motivation

2.1 Preliminaries of GCN Accelerators

Graph neural networks analyze the graph's structure and learn the characteristics of nodes, edges, or even the entire graph. The Graph Convolutional Networks (GCNs) [49] apply graph convolutions (GCNConv) recursively to extract meaningful information from the graphs. A representative example of single GCNConv layer can be found in Fig. 1. A given graph is denoted by $G = (\mathcal{V}, \mathcal{E}, A)$, which incorporates $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges and adjacent list A. Each node within the graph is affiliated with an C-dimensional feature vector, and the matrix $X \in \mathbb{R}^{|\mathcal{V}| \times C}$ represents the collection of these feature vectors for all nodes, serving as the feature embedding matrix. The forward propagation within the single GCNConv layer can be conceptually divided into two distinct stages. The initial stage involves a linear transformation, represented by the equation $Y = XW$. The second stage incorporates a feature aggregation operation, given by $X' = \sigma(AY)$. This two-step process illustrates the core functionality of the GCNConv layer within the GCN.

In the aggregation stage of graph convolutional networks (GCNs), the adjacency list A is typically characterized by extreme sparsity and follows a power-law distribution [28], such as 0.144% for Cora [57], 0.082% for CiteSeer [31], and 0.023% for Pubmed [69]. HyGCN [86] employs a design strategy that separates aggregation and combination stages using sparse-sparse (SpGEMM) and sparse-dense (SpMM) matrix multiplication engines. However, this approach leads to under-utilization and workload imbalance between the engines due to their reliance on graph input characteristics.

To address the under-utilization issue, AWB-GCN [28], GROW [43], and GPU accelerators, such as GNNAdvisor [82], present unified hardware designs for both aggregation and combination phases. Moreover, later work AWB-GCN [28] and GCoD [88] recognize the processing of nodes with larger degree as a major bottleneck, particularly when utilizing thousands of processing elements, and thus focus on exploiting parallelism in GCNs. Furthermore, I-GCN [29] leverages the clustering nature of graphs, having developed an island (cluster) detector and island consumer to enhance graph processing locality. FlowGNN [66] exploits a message passing based dataflow implementation of GNNs acceleration on FPGA platform, similar to that of PyG [26] on GPU platforms. However, the design doesn't address the workload imbalance issue and lacks performance scalability to process large graphs compared to existing GPU platform with large number of core count (6912 CUDA cores for A100 GPU). GROW [43] deploys the row-wise product to accelerate the GNN workload and employ the graph partition algorithms to enhance the locality of SpMM process.

2.2 Cryogenic Devices and AQFP Superconducting Logic

AQFP, originally stemming from quantum-flux-parametron (QFP) logic, is a superconducting logic family initially proposed in 1985 [55]. The adiabatic version of QFP, introduced in [75], achieves substantially lower energy dissipation (5-6 orders lower than CMOS) by re-parameterizing the device to operate in an adiabatic mode. Furthermore, recent research on the reversible version of QFP (RQFP) suggests that bit-level information transfer energy may even surpass the Shannon limit (k_BTln2) [74]. Like other superconducting logic families, AQFP utilizes Josephson Junctions (JJ) as the core switching element for state transitions in logic encoding.

The fundamental AQFP circuit structure is the AQFP buffer, comprising a double-Josephson-Junction SQUID (J_1, J_2) [21], as illustrated in Figure 2. Primarily driven by AC power, which functions as both excitation current and power supply, this configuration utilizes fluxes generated by an applied AC current I_{bias} (trapezoidal or sinusoidal). The direction of the input current I_{in} determines the presence of a single flux quantum in the left or right loop, thereby dictating the direction of the output current I_{out} and representing the logical state '1' or '0'. A transformer consisting of L_q and L_{out} amplifies and delivers I_{out} to the next logic level. The AQFP buffer forms the basis for a suite of AQFP logic gates, including INVERTER, AND, OR, MAJORITY, and SPLITTER.

Unlike conventional CMOS technology, both combinational and sequential AQFP logic cells are driven by AC power, which also serves as a synchronization mechanism or clock signal. Consequently, AQFP circuits inherently exhibit deep-pipelining due to the requirement of overlapping clock signals. A detailed design methodology for the AQFP standard cell library can be found in [77].

Due to the principle of AQFP buffer, the output is sensitive to the direction of the input current. When the amplitude of input current is very small, which falls in the "grayzone" ΔI_{in} [27] of an AQFP buffer, the stochastic switching behavior (caused by the thermal or quantum fluctuation) exists in an AQFP comparator will make the AQFP hard to detect the direction of the input current, resulting in a randomized output with a probability related to input current, i.e., $0 < P(I_{in}) < 1$. This unique property is a double-sided sword that introduces inaccuracy but also makes it possible to be combined with stochastic computing. SC-AQFP [12], an AQFP-based DNN acceleration framework, employs stochastic computing but is only effective for simple tasks on small networks (e.g., MNIST). Another study [85] proposes a BNN model-based crossbar synapse array architecture tailored for AQFP logic. However, current attenuation, limited scalability, and the randomized behavior of AQFP buffers challenge the true implementation of this architecture. Our proposed framework addresses these issues, offering a feasible solution. To address these concerns, SupeRBNN [50] proposes an AQFP-aware BNN training framework that accounts for AQFP characteristics. Nevertheless, as it only takes into account the general convolutional layers, it falls short of meeting the computational demands associated with GCN.

Besides the superconducting devices, CMOS-based cryogenic devices have been investigated as an optional solution as they can improve computer devices' energy efficiency due to the lower leakage current and wire latency [68, 91]. Multiple cryogenic CMOS-based works [3, 8, 60, 61, 64, 65] are proposed to improve the overall hardware performance. Different from superconducting computation applied under 4K temperature, 77K temperature is more actively considered for cryogenic CMOS-based design to save the cooling consumption. The corresponding comparison is incorporated into the experimental evaluation.

2.3 Stochastic Computing

Stochastic computing (SC) is a paradigm that represents a stochastic number (SN) by counting the number of ones in a uniformly distributed bitstream. For instance, the bitstream 0100110100 denotes a real number $x = P_X = 4/10 = 0.4$, where X represents the stochastic bitstream and x its associated real value. An SC with a bipolar encoding format can accommodate numbers in the range of [-1, 1], with a real number x processed through $P(X = 1) = (x + 1)/2$. For example, 0.4 can be represented as 1011011101 and -0.6 as 0100100000. Figure 3 illustrates various SN representation formats. SC boasts low hardware-cost implementation for arithmetic operations. Unipolar and bipolar multiplication can be handled using a single AND and XNOR gate, respectively, while addition operations can leverage the OR gate, multiplexer, or approximate parallel counter (APC) [47]. Typically, SNs are generated via hardware-based random number generators; however, in our design, we directly utilize neuron circuit output results as SNs due to the true random property of the AQFP buffer [30, 73].

2.4 Model Quantization and Binary Neural Network

Model quantization enables deep neural network (DNN) inference acceleration on edge devices by compressing model size and improving inference speed. This technique maps the 32-bit floating-point weight and/or activation values in a DNN model to lower bit-width values by partitioning the data range into a specified number of levels. Quantization research can be classified into different schemes, from the extremely low precision seen in Binary (BNN) [23, 24, 53, 63]

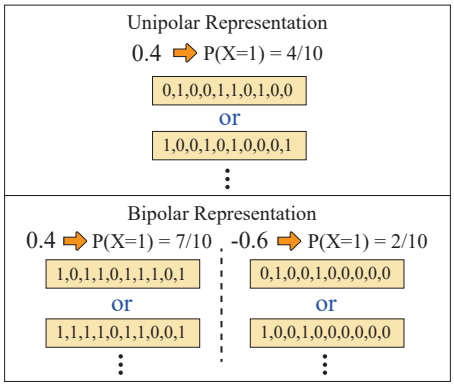

Figure 3: Examples of the unipolar and bipolar representations of stochastic numbers.

and Ternary Neural Networks (TNN) [39, 94] to low-bit-width fixed-point networks [20, 93] which uniformly quantize models.

BNNs, with their weights constrained to $\{-1, 1\}$, simplify hardware implementation and reduce operations by replacing multiplications with additions/subtractions or eliminating them using XNOR and AND operations when activations are binary. This renders BNNs suitable for low-power consumption scenarios. However, the limited representational ability of BNNs results in accuracy degradation, prompting research to explore mitigations, such as introducing scaling factors [63], fusing scaling factors [10], and utilizing multiple scaling factors [53]. Furthermore, methods to minimize gradient information loss [32, 62, 87] and model modifications to preserve accuracy after quantization [54] have been proposed.

Some recent works developed Binarized Graph Convolutional Network (GCN) frameworks [7, 81] that extend weight and feature binarization into the GCN domain. These approaches, however, primarily focus on feature binarization across layers, inadequately considering intermediate graph features within a single GCNConv layer. Consequently, such frameworks cannot be directly adapted to Adiabatic Quantum-Flux-Parametron (AQFP) devices.

3 Architecture Design for AQFP-based GCN accelerator

The overall architecture of the AQFP-based GCN accelerator is shown in Fig. 4, in which the computation mainly contains two parts: combination computation and aggregation computation. In this section, we discuss the corresponding details of our proposed architecture paradigms. All components for GCN computation, i.e., memory tile, crossbar tile, and register are implemented using the AQFP technology.

3.1 GCN Computation Initialization

For the initialization, the vectors representing the graph are stored in AQFP memory tiles. Each tile stores a sub-graph derived by graph boundary edge re-growth partition algorithm (details are illustrated in Section 4.3), thus, enabling the hardware parallelism to improve the performance of throughput. To effectively represent the graph, we utilize two vectors: the vector "indices" stores the column indices of the non-zero elements, and the vector "indptr" stores the index pointers in "indices" indicating where each row starts. We employed 8 KB AQFP memory tiles with the architecture presented in [72] for storing the graph vectors. During the mapping process, we use the indices as addresses to properly map the relevant elements into the memory crossbar tiles. Additionally, the weights are stored in the crossbar tiles allocated for combination computation. After initialization, the computation for each layer of the GNN is performed in three phases: (1) Combination computation phase, (2) Interim storage phase, and (3) Mapping and aggregation phase.

3.2 GCN Combination Computation

GCN combination computation is mainly a dense matrix multiplication incorporating both GCN weight matrix and input node feature matrix. Considering the binarization, the whole matrix computation can be converted into XNOR operation and accumulation. Inspired by recent work [85], we can use an AQFP-based crossbar to handle the combination computation and address the challenges arising from the data movement between memory and computing units in conventional Von Neumann architectures. However, the direct implementation of AQFP crossbar still faces two problems: 1) the limited crossbar size due to the current attenuation phenomenon restricts the ability to accommodate the entire computation within the structure. 2) the original design of the AQFP-based crossbar supports only 1-bit output provided by the classic AQFP buffer, whereas the accuracy requirements of GCN computations necessitate multi-bit intermediate results.

To address these problems, we propose our AQFP-based architecture paradigm for combination computation in GCN consisting of three components. First, following [85], we employ an AQFP-based crossbar as the main body for performing the binarized weight matrix multiplication computation (Section 3.4). This leverages the inherent parallelism and computational capabilities of AQFP technology. Second, we adjust the functionality of the AQFP buffer with a linear probability distribution to serve as a stochastic computation bit-stream generator (Section 3.6). This modified buffer generates the stochastic bit-streams required for the stochastic computing process. Finally, we incorporate a stochastic computing module that accumulates the generated

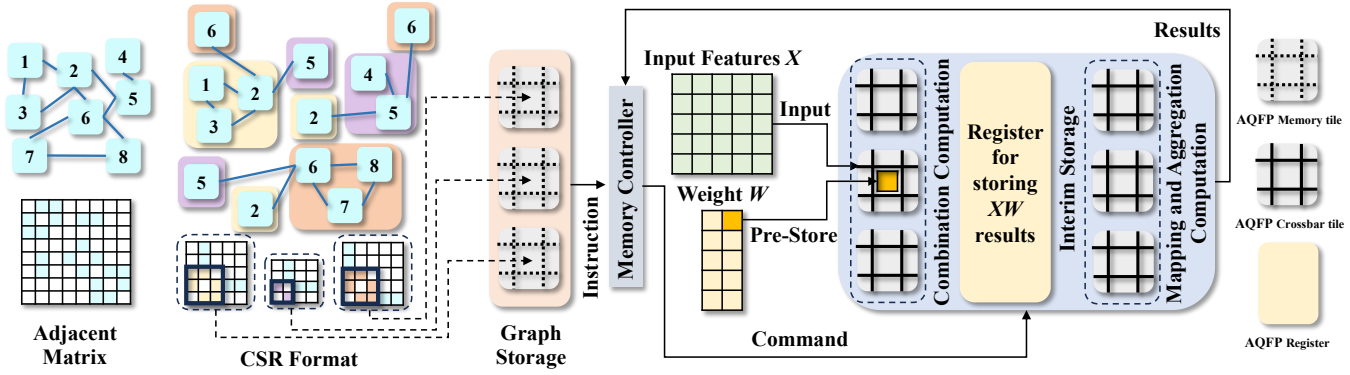

Figure 4: The overall architecture for AQFP-based GCN implementation.

bit-streams and provides the desired multi-bit intermediate result (Section 3.5). By introducing these three components, our proposed AQFP-based architecture paradigm enables efficient and accurate combination computation in GCNs. It overcomes the limitations of crossbar size and output precision, making it feasible to leverage the advantages of AQFP technology for GCN computations while maintaining the required accuracy.

3.3 GCN Aggregation Computation

During the combination computation phase, the results for $Y = XW$ are calculated. These results are then stored in registers during the interim storage phase. In the next step, the relevant data is read from the registers and mapped into the crossbar arrays using the addresses provided by the graph. Once the data is properly placed in each AQFP crossbar tile for the aggregation computation, all crossbar tiles associated with the aggregation computation perform the computation in parallel, resulting in the generation of outputs. If these results are for the first layer, they are written back to the crossbar associated with the combination computation through an AQFP buffer to convert it back to binary values. This enables the computation for the second layer to be started and performed in a similar manner to that of the first layer, as discussed earlier.

The architecture size can be adapted flexibly to the graph dataset that is being processed. And the crossbar size is designed according to the number of GCN hidden futures.

3.4 AQFP-based Crossbar Architecture

Figure 5 illustrates the circuit architecture of AQFP-based crossbar. Following the work [85], the binarized weights are pre-stored in 1-bit AQFP logic-in-memory (LiM) cells and multiplied by an in-cell XNOR macro. The resulting output of each LiM cell corresponds to the multiplication of the input feature matrix input from each row of the crossbar, and the

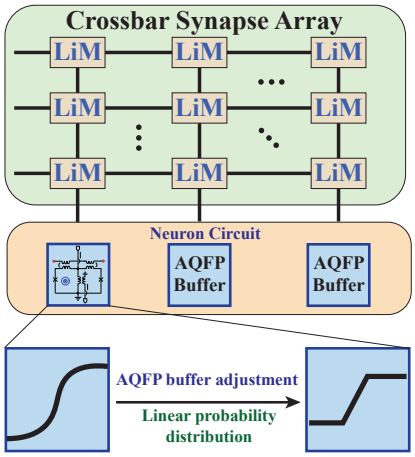

Figure 5: AQFP-based Crossbar Architecture.

corresponding pre-stored weight. Notably, this approach deviates from the conventional popcount-based accumulation method used in BNNs. Instead, we adopt an analog summation technique that directly adds up all the outputs, taking advantage of the fact that AQFP represents logic '1' and '0' using positive and negative current pulses. Consequently, the accumulated result, represented by the current sum-up of each column in the synapse array, is subsequently transmitted to the adjusted AQFP buffer for next-level computation.

3.5 Stochastic Computing-based Accumulation

According to [50, 85], the randomized behavior that appears in the AQPF buffer presents an output probability dependence on the input current amplitude, which can provide a sufficient level of stochastic numbers (SNs) through a certain observation window with almost no hardware overhead. The output AQFP buffer can efficiently convert the attenuated

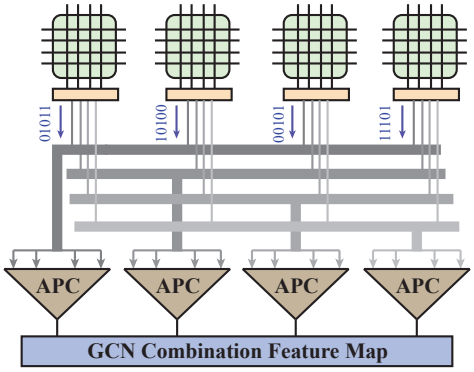

Figure 6: Architecture design of SC-based accumulation module.

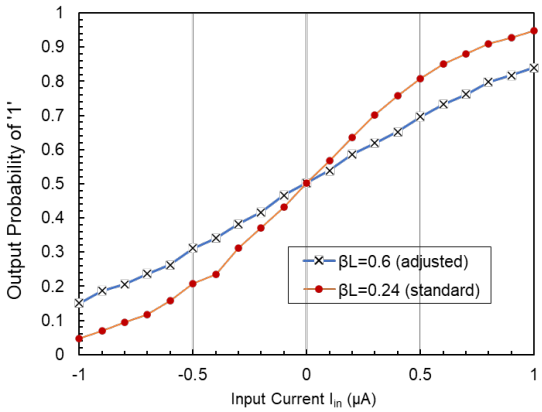

Figure 7: The relationship between output probability of "1" with input current on AQFP buffer based on different β_L.

output current into SNs to resolve the possible accuracy loss introduced by the analog current addition.

For example, as shown in Figure 6, for each clock phase, the AQFP buffer of the crossbar will generate a 1-bit output with the probability of '1' or '0' depending on the accumulated current from the corresponding crossbar column. When we use an observation window for the output, we can obtain an output bit-stream, which is naturally a stochastic number. To perform the accumulation of stochastic numbers (SNs) across different crossbars, we utilize approximate parallel counters (APCs) [47] at the output of the module. Note that all logic cells and circuits, including APCs and comparators, are designed using the AQFP standard cell library. This library comprises AQFP logic gates such as AND, OR, buffer, inverter, majority, splitter, and read-out interfaces. By utilizing the AQFP standard cell library, we ensure compatibility and seamless integration of all logic elements in our proposed architecture.

In contrast to the approach presented in [50], which exclusively employs the stochastic computing-based accumulation module to address the limited scalability of AQFP crossbars, our method harnesses the power of stochastic computing to achieve a broader bit-width precision in the intermediate results, thereby meeting the accuracy requirements of Graph Convolutional Networks (GCN). The detailed algorithm is elaborated upon in Section 4.2, and the corresponding accuracy analysis is provided in Section 5.5.

However, the standard AQFP buffer exhibits a non-linear randomized distribution in terms of input current, which can introduce distortion and mismatches in the generated stochastic numbers. To overcome this challenge and fully exploit the randomized behavior of the AQFP buffer, we adjust the device parameter in the standard AQFP buffer to achieve a nearly linear probability distribution as shown in Section 3.6.

3.6 AQFP Buffer Adjustment

In order to use AQFP-buffer to generate multi-bit intermediate results, we need to modify the AQFP design to increase the geometric parameter β_L and achieve a more linear distribution in the "grayzone". This parameter is proportional to the number of flux quanta that can be screened by the Josephson junctions' maximum critical current. Implementing a larger β_L value decreases the switching sensitivity to the input current, resulting in a more stable slope, as shown in Figure 7.

To address the non-linear randomized distribution of the classic AQFP buffer, we deploy an adjustment mechanism to achieve a linear probability distribution. This adjustment is crucial for fully leveraging the randomized behavior of the AQFP buffer as a stochastic computing bit-stream generator.

3.7 AQFP Adjusted Buffer Behavior Analysis

As mentioned above, we use adjusted AQFP buffers to function as a stochastic number generator. We adjust the randomized distribution into the linear format to satisfy the stochastic computing requirement and achieve integer intermediate results in combination computation. In our research, conducted at a temperature of 4.2K, the probability distribution incurred by the randomized behavior is shown in Figure 7, which presents the output probability of '1' for adjusted AQFP buffer corresponding to a given input current amplitude at the micro-ampere level. The formulation can be denoted as:

$$P\left(I_{in}\right)=\begin{cases}0, & I_{in}<-\Delta I_{in}/2\\ 0.5+2I_{in}/\Delta I_{in}, & -\Delta I_{in}/2\le I_{in}\le\Delta I_{in}/2\\ 1, & I_{in}>\Delta I_{in}/2\end{cases}\quad(1)$$

where I_{in} is the input current amplitude of the AQFP buffer, and ΔI_{in} means the length of the "gray-zone".

Considering the impact of superconductive inductance which incurs the current attenuation, the accumulated current inside each column of the crossbar would decrease when the crossbar size becomes larger (has additional inputs in the merging circuits). Assuming the accumulated current amplitude in each column of the crossbar representing the value of "1" (unit current) be denoted as $U(C)$, which is negatively correlated with the crossbar number of inputs C, then the value presented by the accumulated current (which is also the input current of AQFP buffer) is denoted as $V_{in}=I_{in}/U(C)$. Thus, we can rewrite the probability distribution equation (1) into the value form (when I_{in} drops into ΔI_{in}):

$$\mathcal{P}\left(V_{in}|\Delta I_{in},C\right)=0.5+\frac{2V_{in}U(C)}{\Delta I_{in}},\quad(2)$$

which bridges the software training algorithm with the hardware configuration.

4 AQFP-aware GNN Hybrid Quantization Training Algorithm

4.1 Graph Convolutional Network

Graph Convolutional Network (GCN) [49] is a very popular member of the graph neural network family. Here, we show a brief review of GCN. Given an undirected graph G, one layer of the graph convolution operation can be described as:

$$X^{(l+1)}=\sigma\left(\tilde{A}X^{(l)}W^{(l)}\right),\quad(3)$$

where σ expresses the activation function, $\tilde{A}$ denotes the adjacency matrix, $X^{(l)}$ means the input node feature map of the l-th layer, $W^{(l)}$ means the weight matrix of the i-th layer which contains the learnable parameters.

From a spatial methods perspective, the graph convolution layer in Graph Convolutional Networks (GCN) can be decomposed into two distinct steps: combination and aggregation. The first step is the combination step, denoted as $Y^{(l)}=X^{(l)}W^{(l)}$, performing simple matrix multiplications of weights and computed feature vectors associated with vertices. The second step aggregation, denoted as $\tilde{A}Y^{(l)}$, requires graph traversal using an adjacency matrix that represents connections between vertices. Notably, the adjacency matrix of a graph structure is extremely sparse (with a density around 0.1% level or even lower), it requires compression to perform an efficient hardware implementation.

4.2 AQFP logic-aware GCN Hybrid Quantization Co-design

4.2.1 Binarization of Weight Matrix and Node Feature. Following the conventional GCN binarization framework, such as [7, 81], we perform binarization quantization for both weight matrix $W\in\mathbb{R}^{F^{(l)}\times F^{(l+1)}}$ and input node feature $X\in\mathbb{R}^{N\times F^{(l)}}$, where $F^{(l)}$ means the feature dimension of the i-th layer, N means the number of nodes. The binarization can be described as:

$$W^{(l)}_{:,j}\Rightarrow\alpha^{(l)}_{j}\,\mathrm{sign}\left(W^{(l)}_{:,j}\right)\quad(4)$$

where $W^{(l)}_{:,j}$ means the j-th column of $W^{(l)}$, and $\alpha^{(l)}_{j}$ is the per-column scaling factor computed by the L_1 norm of the corresponding column:

$$\alpha^{(l)}_{j}=\frac{1}{F^{(l+1)}}\left\|W^{(l)}_{:,j}\right\|_1,\quad(5)$$

After the binarization, the binarized weight $\tilde{W}$ contains two components: the binary matrix $\hat{W}^{(l)}=\mathrm{sign}\left(W^{(l)}\right)$ which be computed in the crossbar and the scaling factor vector $\alpha^{(l)}$ which can be considered in the adjusted AQFP buffer. The input node feature map $X^{(l)}$ is binarized in a similar manner. The only difference is that we use one single scaling factor $\beta^{(l)}$ for the whole matrix, preventing mismatches on AQFP devices. Thus, the entire combination computation can be converted to:

$$Y^{(l)}_{i,j}=\alpha^{(l)}_{j}\beta^{(l)}\hat{X}^{(l)}_{i,:}\hat{W}^{(l)}_{:,j}.\quad(6)$$

where $\hat{X}^{(l)}_{i,:}\hat{W}^{(l)}_{:,j}=\sum_{k=1}^{F(l)}\hat{X}^{(l)}_{i,k}\cdot\hat{W}^{(l)}_{k,j}$ presents the matrix multiplication between binarized input with weight. The binarized weight $\hat{W}^{(l)}_{k,j}$ is pre-stored in the k-th row, j-th column of AQFP crossbar LiM, while the binarized input value $\hat{X}^{(l)}_{i,k}$ is presented by the direction of the i-th input current at the k-th row of the crossbar. For a toy example, if $\hat{X}^{(l)}_{i,:}=\{0,1,1\};\hat{W}^{(l)}_{k,j}=\{1,1,0\}^T$, then the value of the combination result $Y^{(l)}_{i,j}=\alpha^{(l)}_{j}\beta^{(l)}(0+1+0)=\alpha^{(l)}_{j}\beta^{(l)}$. For the gradient approximation in back-propagation, we follow the setting of [81].

4.2.2 AQFP-aware Multi-bit Quantization for Combination Computation Result. As mentioned earlier, conventional GCN binarization frameworks, such as [81], do not consider the intermediate result's optimization. This omission creates inefficiency and hinders the seamless mapping of binarized GCN frameworks to hardware, especially for AQFP-based devices. Due to the significant accuracy degradation observed when directly binarizing the combination computation results (please refer to the ablation study in Section 5.5), we propose to use a hybrid quantization scheme and reserve

low-bit precision for combination computation results to preserve the accuracy.

For m-bit quantization, we derive the quantized values from the set:

$$\mathbb{S}(m) = \gamma^{(l)} \times \{-1, (\frac{2}{2^m-1} - 1), (\frac{4}{2^m-1} - 1), \ldots, 1\}, \quad (7)$$

where γ is the scaling factor, which is a learnable value fixed in the inference period. Then the quantizer function from the original floating-point matrix Y to an m-bit value matrix $\hat{Y}$ is expressed as

$$\hat{Y}^{(l)} = \gamma^{(l)} \cdot h^{-1}\left(\frac{1}{2^m-1} \cdot \text{round}\left((2^m-1) \cdot h\left(\lceil Y^{(l)}, \gamma^{(l)} \rfloor\right)\right)\right), \quad (8)$$

where $h(\cdot)$ shifts a value within $[-1, +1]$ into the range of $[0, 1]$ (e.g., $h(x) = x/2 + 0.5$), and $\lceil Y^{(l)}, \gamma^{(l)} \rfloor$ clips each element in $Y^{(l)}$ by:

$$\lceil Y_{i,j}^{(l)}, \gamma^{(l)} \rfloor = \begin{cases} -1, & Y_{i,j}^{(l)} < -\gamma^{(l)} \\ Y_{i,j}^{(l)}/\gamma^{(l)}, & -\gamma^{(l)} \le Y_{i,j}^{(l)} \le \gamma^{(l)} \\ 1, & Y_{i,j}^{(l)} > \gamma^{(l)} \end{cases}. \quad (9)$$

To fully leverage the randomized behavior of AQFP buffers for generating the stochastic numbers, we need to map the multi-bit quantization levels into the randomization distribution presented by equation (2). Comparing equation (2) with (9), we find the clipping and the randomization distribution in the value domain have the same format. For combination computation in l-th layer, given the i-th row of the input node feature, the value represented by the accumulated current in the j-th column in the crossbar $V_{in,i,j}^{(l)} = Y_{i,j}^{(l)}/(\alpha_j^{(l)}\beta^{(l)})$. Consequently, AQFP-based device and the quantized model can be bridged by setting the hardware configuration ΔI_{in} for l-th layer and j-th column as:

$$\Delta I_{in,j}^{(l)} = \frac{2\gamma^{(l)}U(C)}{\alpha_j^{(l)}\beta^{(l)}} \quad (10)$$

which makes it feasible to generate the multi-bit combination result with scaling factors by AQFP buffer.

4.3 Graph Boundary Edge Re-growth Partition Algorithm

In the field of hardware parallelism, handling large graph datasets requires breaking them down into manageable subgraphs that can be individually processed. Traditional graph partitioning methods, while valuable, have presented specific challenges. Various methods have been developed for graph training and inference partitioning, including neighborhood sampling [34], graph partitioning [18, 29, 46], and boundary sampling [80]. However, shortcomings have been identified in these methods. Neighborhood sampling-based approaches, for instance, can increase inference latency and processing time due to the graph sampling process [34]. Moreover, the METIS library [46, 80], while offering a basic graph partitioning solution, can create subgraphs devoid of inter-cluster connections, thereby resulting in decreased inference accuracy. BNS-GCN [80] attempted to address this issue by sampling first-degree neighbors during training, yet the analysis of the tradeoff between inference efficiency and partition accuracy was found to be lacking.

Algorithm 1 Graph Boundary Edge Re-growth Partition Algorithm

Require: A, X {Graph input as adjacent list and embedding}

1: $A_0, A_1, ..., A_k \leftarrow$ METIS_PARTITION(A) {Conduct a METIS partition algorithm on A}
2: **for** $i = 0$ to k **do**
3: $E_i \leftarrow$ DETECT_BOUNDARY_EDGES(A_i) {Detect all boundary edges of partition A_i}
4: $A_i \leftarrow A_i \cup E_i$ {Regrow the boundary edges}
5: **end for**
6: **return** $A_0, A_1, ..., A_k$ {Return the graph partitions}

To construct the partitions of graph datasets, the METIS algorithm [46, 80] has been employed, yielding n partitions of nodes: $\mathcal{V} = [\mathcal{V}_1, \cdots \mathcal{V}_n]$, where $\mathcal{V}_i$ represents the nodes in the i-th partition. These partitions are represented as $\bar{G}_i = [G_1, \cdots, G_n]$, where G_i consists of the nodes and edges within $\mathcal{V}_i$. Subsequently, the nodes are reorganized, and the adjacency matrix is partitioned into i^2 sub-matrices. Each diagonal sub-block A_{ii} represents an adjacency matrix containing intra-connections within $\mathcal{V}_i$, while diagonal block matrix $\bar{A}$ is the adjacency matrix for $\bar{G}$, with $A_{jk}(j \neq k)$ denoting inter-connections between two different partitions $\mathcal{V}_j$ and $\mathcal{V}_k$. The METIS algorithm, however, has been observed to cause accuracy degradation during the inference stage. The root cause of this problem is the removal of inter-cluster edges, resulting in feature loss at the nodes and inhibiting message passing between distinct clusters.

To overcome these challenges, we develop a regrowth-after-partitioning algorithm, as detailed in Algorithm 1, which subdivides the graph into smaller sub-clusters and facilitates the reconnection of edges and nodes between clusters. This approach emphasizes the regeneration of boundary edges between disconnected clusters to mitigate feature loss and enable efficient message passing between inter-cluster nodes.

The regrowth-after-partitioning method features 4 significant benefits:

- The partition algorithm can be conducted in parallel with training, and the partition result can be reused during inference.

- At inference time, the partitioned graph improves enables parallel graph processing, where each sub-graph can be process separately by each AQFP device.
- The partition algorithm only leads to minor inference FLOPs overhead, observing that the normal graph datasets examined in this study contain approximately only 10% to 25% 1-hop boundary edges (nodes) between clusters, depending on graph structure and number of partitions.
- With the proposed partition algorithm, the graph processing accuracy is more robust when we increase the number of partitions, as evidenced in Figure 8.

It demonstrates performance in terms of enhanced inference speed, reduced single-device memory usage, and minimal accuracy degradation, thus overcoming the limitations of existing methods.

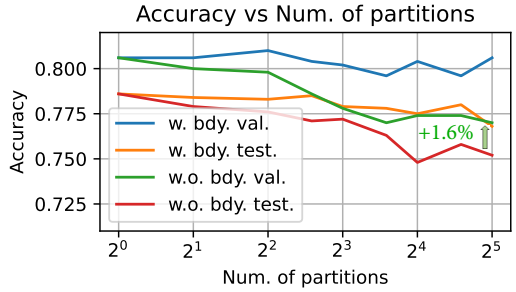

Figure 8: Partition algorithm evaluation on Pubmed

Partition Algorithm Evaluation. Figure 8 presents an evaluation of the partition algorithm on the PubMed dataset. The plot provides a comparison of the accuracy between our algorithm and the METIS [46, 80] algorithm, considering cases with boundary edge recovery (w. bdy.) and without boundary edge recovery (w.o. bdy.). It can be observed that the test accuracy experiences significantly less degradation when boundary edge recovery is employed, a phenomenon that is particularly evident at higher partition numbers.

5 Experimental Evaluation

5.1 Experimental setup

AQFP hardware implementation employs a semi-automated design method geared towards the AIST 4-layer 10 kA/cm^2 niobium process (HSTP) [59]. Analog cells and circuits are manually optimized at the Josephson-junction (JJ) level, while logic cells and circuits utilize the AQFP standard cell library. The entire circuit operates on a delay-line (microstrip line) based clocking scheme [38] with a frequency of 5 GHz and a 5 ps inter-stage delay. Verification at the circuit level employs a modified Josephson simulator, Jsim [25], accounting for thermal noise.

For the GNN training, we focus on the transductive learning task. A two-layer GCN with 64 hidden features is used as our model architecture. AQFP-aware hybrid quantization is performed on this architecture to evaluate our overall performance. Besides that, Adam [48] optimizer is used with a learning rate of 0.001. The total training epoch is 1000, and the dropout layers are utilized in the training process with a dropout rate of 0.4.

Three datasets are used in our experimental evaluation: Cora, CiteSeer, and PubMed, which are shown in Table 1. The training strategies and hyperparameters keep the same the same for all of them.

Table 1: Detailed information of three datasets: Cora, CiteSeer, and PubMed. Density (A) means the density of the adjacency matrix.

Dataset	Cora	CiteSeer	PubMed
#Nodes	2708	3327	19717
#Edges	5429	4732	44338
#Features	1433	3703	500
Density (A)	0.144%	0.082%	0.023%
# Classes	7	6	3

5.2 Visualization of Graph Boundary Edge Re-growth Partition

In the context of graph partitioning and hardware acceleration, it is essential to understand the partitioning and regrowth methodologies' effect on the original sparse graph structure. We illustrate an example of utilizing a partition and regrowth technique in Algorithm 1, utilizing the Cora dataset as an exemplar. As depicted in Fig. 9(a), the original graph features irregular sparsity and comprises edge connections dispersed across the entire matrix.

To facilitate the partitioning process, we employ the METIS library [46, 80] to reorder the graph into four primary clusters. This reordered graph is presented in Fig. 9(b). We subsequently extract the second graph cluster along with its boundary nodes and connections, resulting in the graphical representation seen in Fig. 9(c). Upon inspection, we discern that the cluster maintains few boundary edge connections with all other clusters.

To further optimize the partition, we aggregate the boundary nodes and edges while disregarding other unused nodes. The outcome is illustrated in Fig. 9(d). Through this process, we discover that only a minimal number of boundary nodes (112 nodes) are incorporated into the original cluster of 677 nodes. Consequently, the overall computational workload remains largely unaffected.

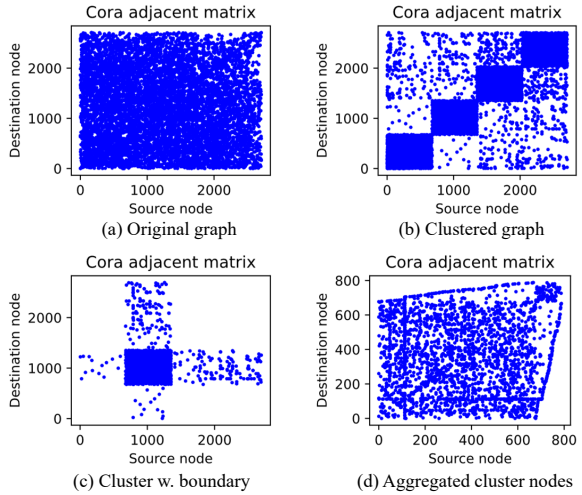

Figure 9: Graph partition and boundary regrowth process on Cora dataset

The significance of the aforementioned partition and boundary partition algorithm extends beyond structural rearrangement. In the preprocessing stage, this approach enables us to allocate distinct hardware to process each cluster, thereby obtaining the classification label of each cluster's inner node. Notably, this strategy is executed without incurring substantial hardware overhead, underscoring its efficiency and potential utility in hardware-accelerated graph processing.

5.3 Hardware Performance Analysis

As shown in Table 2, we compare our AQFP-based design with multiple representative GNN frameworks based on different platforms and optimizations, including I-GCN [29], AWB-GCN [28], GCoD [88], FlowGNN [66], HyGCN [86], CoDG [56], REFLIP [42], and Cryo-CMOS [11]. Since the existing ASIC-based GCN accelerators can also benefit from 77K cryogenic computing. We estimate the energy efficiency of the 77K implementation of HyGCN according to [17] and cooling consumption according to [11]. Our work shown here uses 4 bits for combination results with 6 graph partitions & boundary regrowth.

Without considering the cooling consumption, compared with the CMOS-based framework I-GCN and AWB-GCN, FPGA-based framework HyGCN, I-GCN, and FlowGNN, and ASIC-based framework HyGCN, our AQFP-based implementation achieves significant improvement on both latency and energy-efficiency perspectives in all three datasets. Compared with the ReRAM-based framework CoDG-ReRAM, although the latency is a little bit longer, our proposed framework achieves 4 to 6 degrees of magnitude better energy efficiency if we only consider the chip power dissipation. This huge improvement comes from binarization, efficient partition and mapping, and AQFP high-efficiency benefits.

5.4 Energy Consumption Comparison Considering Cooling Consumption

Modern cryogenic computing often aims for two target low temperatures, 77K and 4K, because the two temperatures can be easily achieved by applying Liquid Nitrogen (LN) and Liquid Helium (LHe), respectively. To satisfy the superconducting cooling requirement, our Low-level circuits are measured at the 4K level within a liquid helium Dewar. Notably, the cooling cost for typical superconducting digital circuits stands at approximately 400 times the chip power dissipation [41]. Despite considering cryo energy, our proposed framework still exhibits significantly higher energy efficiency compared to previous GNN acceleration approaches, surpassing FPGA-based and ASIC work by about four orders of magnitude, and ReRAM-based work by two to four orders of magnitude.

Besides the superconducting devices, CMOS-based cryogenic devices have been investigated as an optional solution as they can improve computer devices' energy efficiency due to the lower leakage current and wire latency [68, 91]. Multiple cryogenic CMOS-based works [3, 8, 60, 61, 64, 65] are proposed to improve the overall hardware performance.

In addition to the conventional devices, our accelerator is compared against Cryo-CMOS [11], which employs a CMOS-based cryogenic technique to enhance hardware performance under low operating temperatures. Since the cooling consumption accounts for a major portion of energy dissipation for cryogenic devices, to achieve better overall energy efficiency, Cryo-CMOS is applied under 77K temperature to save the cooling consumption. According to the total energy consumption, our results indicate that our framework achieves 1.6×10^3, 8.4×10^3, and 1.7×10^4 times better energy efficiency under Cora, CiteSeer, and PubMed datasets, respectively.

Regarding the 77K implementation of HyGCN, as discussed in Section 5.3, we estimate performance based on prior studies [11, 17]. The cooling consumption of 77K is about 9.65 times the device consumption [11]. Considering the cooling consumption, 77K Cryo-CMOS can achieve about 1.5 times the energy efficiency of the conventional room temperature CMOS. Even when accounting for cooling consumption, our approach achieves energy efficiency improvements ranging from 9.8×10^3 to 1.9×10^4 for various graph datasets, consistently delivering faster inference speeds.

As we delve into the emerging field of AQFP-based devices, it is essential to consider the potential for further advancements, such as new production processes and new superconducting materials. This could lead to even lower

Table 2: Comparison of latency in μs, and energy efficiency in Graph/kJ on Cora, CiteSeer, and PubMed dataset. Our work uses 4 bits with 6 graph partitions & boundary regrowth. Representative works I-GCN [29], AWB-GCN [28], FlowGNN [66], HyGCN [86], GCoD [88], CoDG [56], REFLIP [42] and Cryo-CMOS [11] are compared. Energy efficiency: graphs/kJ.

Work	Platform	Optimization	Cora		CiteSeer		PubMed	
			Latency	Energy-efficiency	Latency	Energy-efficiency	Latency	Energy-efficiency
Without Considering Cooling Consumption								
I-GCN	D5005 FPGA	Graph reorder	1.3	7.1E6	1.9	3.7E6	15.1	5.3E5
AWB-GCN	D5005 FPGA	Load balancer	2.3	3.1E6	4.0	1.9E6	30	2.5E5
GCoD	VCU128 FPGA	quant, reorder	4.41	9.6E5	7.05	5.8E5	56.38	6.8E4
FlowGNN	U50 FPGA	Message passing	6.91	7.8E6	8.33	6.44E6	53.22	1.01E6
REFLIP	ReRAM	fixed point	0.91	2.1E7	1.14	1.1E7	14.55	9.7E5
CoDG-ReRAM	ReRAM	quant, reorder	0.18	1.2E8	0.38	4.9E7	3.48	4.8E6
HyGCN	ASIC	N/A	21	7.2E6	300	4.9E5	640	2.3E5
HyGCN	ASIC (77K)	N/A	~21	9.9E6	~300	6.7E5	~640	3.2E5
Cryo-CMOS	ASIC (77K)	N/A	-	1.2E8	-	7.9E6	-	3.8E6
Ours	AQFP	Hybrid Quant, SC	1.32	**7.0E12**	1.40	**2.5E12**	10.8	**2.4E12**
Considering Cooling Consumption								
HyGCN	ASIC (77K)	N/A	~21	9.3E5	~300	6.3E4	~640	3.0E4
Cryo-CMOS	ASIC (77K)	N/A	-	1.1E7	-	7.4E5	-	3.5E5
Ours	AQFP	Hybrid Quant, SC	1.32	**1.8E10**	1.40	**6.2E9**	10.8	**5.9E9**

overall energy dissipation for AQFP-based devices in the future, making them even more promising for energy-efficient hardware solutions.

5.5 Combination Computation Bit-width Ablation Study

Here, we provide the ablation study for both graph boundary edge re-growth partition and different bit-widths of the combination computation result Y. We also list the model performance result of the representative FPGA-based framework GCoD [56] to be the baseline and have the comparison.

As shown in Table 3, when bit-width of Y equals 1, we achieve a fully binarized GCN model, in which we binarize both the activation, weight, and intermediate result. This one achieves super higher energy efficiency and the shortest latency but incurs a huge accuracy degradation, i.e., about 3~7 % lower accuracy compared with GCoD. When the number of bits in Y reaches 4, our framework achieves a similar level or even higher model accuracy compared with the baseline.

When increasing the bit-width of the combination result (Y), we observe that higher model accuracy is achieved at the cost of lower energy efficiency and longer inference latency. using Cora as an example, transitioning from 1-bit to 4-bit results in a notable accuracy improvement, from 76.4% to 80.0%. However, energy efficiency decreases from 4.7×10^{13} graphs/kJ to 7.2×10^{12} graphs/kJ, and latency increases from 1.14μs to 7.66μs. Moreover, the incorporation of a partition algorithm proves to be a valuable strategy. It significantly reduces inference latency while maintaining a similar level of accuracy and energy efficiency. In the case of the 4-bit combination result version, the use of the partition algorithm decreases latency from 7.66μs to 1.32μs. As a result, our preference leans toward the adoption of the '4-bit + partition' version. This choice places a primary emphasis on preserving high accuracy while simultaneously striking a good trade-off between accuracy and energy efficiency and maintaining lower latency. Compared with the baseline GCoD, our '4-bit + partition' version achieves 3.3× to 5.2× faster inference speed with super higher energy efficiency whether considering the cooling consumption or not, while preserving a similar accuracy level. Regarding overall system accuracy, our hardware-software co-design integrates AQFP's inherent randomness into the stochastic computing training algorithm, resulting in promising accuracy. Furthermore, the GCN architecture only has two layers, ensuring that SC error accumulation remains fully controllable. Previous studies [12, 50] have also demonstrated the reliability of combining AQFP with SC in deep neural networks, further validating our approach. Combined with graph boundary edge re-growth partition, the latency is significantly improved with the help of hardware parallelism, while preserving both the model accuracy and the energy efficiency.

5.6 AQFP-based Application Discussion

Compared to the well-established CMOS technology, superconducting electronics stands out as an emerging field, with anticipated enhancements in scalability and performance forthcoming through future advances in fabrication technologies. The current scalability

Table 3: Ablation study for the graph boundary edge regrowth partition (part.) and bit-width of combination computation result Y. GCoD [88] is listed as a baseline for comparison. Here, model accuracy (%), inference delay (μs), and energy efficiency (EE, Graph/kJ) are compared for different implementations evaluated on Cora, CiteSeer, and PubMed datasets.

	GCoD [88]	Our Framework (1-bit for weight and activation)				
Y bit	-	1 bit	2 bit	3 bit	4 bit	4 bit + part.
Cora						
Acc.	79.5	76.4	77.7	79.7	80.0	**80.2**
Delay	4.41	**1.14**	2.23	3.86	7.66	1.32
EE	9.6E5	**4.7E13**	2.4E13	1.4E13	7.2E12	7.0E12
CiteSeer						
Acc.	**69.6**	62.9	64.6	67.8	68.2	68.3
Delay	7.05	1.67	2.81	4.42	8.31	**1.40**
EE	5.8E5	**1.7E13**	8.3E12	5.2E12	2.6E12	2.5E12
PubMed						
Acc.	**78.6**	75.6	76.4	77.3	77.9	77.8
Delay	56.38	**7.25**	15.6	28.4	58.0	10.8
EE	6.8E4	**1.7E13**	8.3E12	5.1E11	2.5E12	2.4E12

For the future application of the proposed work, we target the AI acceleration component of the next-generation heterogeneous supercomputer. The proposed neural network can serve as an integral component within a high-performance superconducting-based general computing system [6], facilitating the acceleration of AI-based computing. With developed superconducting-CMOS interface technology [19, 37, 58], and the ongoing development of superconducting-quantum interface [76], this system shows great promise for advancing the development of the next-generation supercomputer through heterogeneous integration with both cryogenic CMOS memory systems [3, 40, 64] and quantum computing systems [4, 45]. The proposed neural network can serve as an integral component within a high-performance superconducting-based general computing system [6], facilitating the acceleration of AI-based computing. With developed superconducting-CMOS interface technology, including Josephson Latching Driver (JLD) [19, 37] and nanocryotron (nTron) [58], which can convert small superconducting pulses (usually several microvolts to millivolts) to sufficient voltage-level signals, this system shows great promise for advancing the development of the next-generation supercomputer through heterogeneous integration with cryogenic CMOS memory systems [3, 40, 64]. On the other hand, the ongoing development of superconducting-quantum interface [76] shows that the AQFP-based computing system presents a compelling interface for controlling superconducting qubits [22, 45] in quantum computing, owing to its extremely low energy dissipation. Furthermore, their operation in a congruent thermal environment with superconducting qubits not only mitigates thermal discrepancies but also fosters a streamlined integration on a singular platform. This cohabitation underpins a symbiotic relationship between the computation and control units, potentially facilitating the amalgamation of efficient computation and precise control, which is paramount in the scalable deployment of quantum information systems.

6 Conclusion

In this paper, we propose a GCN acceleration framework based on AQFP technology, systematically addressing the multifaceted challenges in realizing ultra-efficient GCN accelerators. We first develop a regrowth-after-partitioning algorithm to enable the AQFP hardware parallelism and accelerate the aggregation computation while maintaining accuracy. We propose two distinct AQFP-based architectures tailored specifically for each of the combination and aggregation stages. To unlock the extreme energy efficiency, we develop a hybrid binarized/low-bit GCN hardware/software codesign that can be efficiently executed on AQFP-based devices. Leveraging the AQFP randomized behavior, we adjust the AQFP buffer design to achieve multi-bit intermediate results and explore the bit-width at the output of the combination step. To mitigate the gap between the software model with hardware implementation, we propose an AQFP logic-aware GCN Hybrid Quantization. A comparative evaluation with existing GNN frameworks substantiates marked enhancements in latency and energy efficiency. The study further underscores the significance of AQFP as a next-generation device/circuit technology, offering ultra-high energy efficiency in GCN accelerators, with gains ranging from 1×10^4 to 1×10^6 compared to conventional CMOS technology. Our framework demonstrates remarkable energy efficiency even when considering the additional cooling consumption.

Acknowledgments

This research was supported in part by Semiconductor Research Corporation (SRC) Artificial Intelligence Hardware program (CD), and JST FOREST Program (Grant Number JPMJFR226W, Japan) (OC).

References

[1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. 2021. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–38.

[2] Sami Abu-El-Haija et al. 2020. N-gcn: Multi-scale graph convolution for semi-supervised node classification. In *uncertainty in artificial intelligence*.

[3] Shamiul Alam, Md Shafayat Hossain, Srivatsa Rangachar Srinivasa, and Ahmedullah Aziz. 2023. Cryogenic memory technologies. *Nature Electronics* 6, 3 (2023), 185–198.

[4] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[5] Adam Auten et al. 2020. Hardware acceleration of graph neural networks. In *DAC*.

[6] Christopher L Ayala, Tomoyuki Tanaka, Ro Saito, Mai Nozoe, Naoki Takeuchi, and Nobuyuki Yoshikawa. 2020. Mana: A monolithic adiabatic integration architecture microprocessor using 1.4-zj/op unshunted superconductor josephson junction devices. *IEEE Journal of Solid-State Circuits* 56, 4 (2020), 1152–1165.

[7] Mehdi Bahri et al. 2021. Binary graph neural networks. In *CVPR*.

[8] Arnout Beckers, Farzan Jazaeri, Andrea Ruffino, Claudio Bruschini, Andrea Baschirotto, and Christian Enz. 2017. Cryogenic characterization of 28 nm bulk CMOS technology for quantum computing. In *2017 47th European Solid-State Device Research Conference (ESSDERC)*. IEEE, 62–65.

[9] Pietro Bongini et al. 2021. Molecular generative graph neural networks for drug discovery. *Neurocomputing* (2021).

[10] Adrian Bulat and Georgios Tzimiropoulos. 2019. Xnor-net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863* (2019).

[11] Ilkwon Byun, Dongmoon Min, Gyu-hyeon Lee, Seongmin Na, and Jangwoo Kim. 2020. CryoCore: A fast and dense processor architecture for cryogenic computing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 335–348.

[12] Ruizhe Cai et al. 2019. A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology. In *ISCA*.

[13] Yixin Cao et al. 2019. Multi-Channel Graph Neural Network for Entity Alignment. In *ACL*.

[14] Cen Chen et al. 2021. DyGNN: Algorithm and Architecture Support of Dynamic Pruning for Graph Neural Networks. In *DAC*.

[15] Olivia Chen et al. 2019. Adiabatic quantum-flux-parametron: Towards building extremely energy-efficient circuits and systems. *Scientific reports* (2019).

[16] Tianlong Chen et al. 2021. A unified lottery ticket hypothesis for graph neural networks. In *ICML*.

[17] HL Chiang, TC Chen, JF Wang, S Mukhopadhyay, WK Lee, CL Chen, WinSan Khwa, B Pulicherla, PJ Liao, KW Su, et al. 2020. Cold CMOS as a power-performance-reliability booster for advanced FinFETs. In *2020 IEEE Symposium on VLSI Technology*. IEEE, 1–2.

[18] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 257–266.

[19] Fumihiro CHINA, Naoki TAKEUCHI, Hideo SUZUKI, Yuki YAMANASHI, Hirotaka TERAI, and Nobuyuki YOSHIKAWA. 2022. A High-Speed Interface Based on a Josephson Latching Driver for Adiabatic Quantum-Flux-Parametron Logic. *IEICE Transactions on Electronics* E105.C, 6 (06 2022), 264–269. doi:10.1587/transele.2021sep0002

[20] Jungwook Choi, Zhuo Wang, et al. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv:1805.06085* (2018).

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

[22] John Clarke and Frank K Wilhelm. 2008. Superconducting quantum bits. *Nature* 453, 7198 (2008), 1031–1042. doi:10.1038/nature07128

[23] Matthieu Courbariaux et al. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. (2016).

[24] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NeurIPS*.

[25] Emerson S. Fang. 1989. A Josephson integrated circuit simulator (JSIM) for superconductive electronics application.

[26] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[27] T.V. Filippov et al. 1995. Signal resolution of RSFQ comparators. *IEEE Transactions on Applied Superconductivity* (1995).

[28] Tong Geng et al. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *MICRO*.

[29] Tong Geng et al. 2021. I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *MICRO*.

[30] James E Gentle. 2003. *Random number generation and Monte Carlo methods*. Vol. 381. Springer.

[31] C Lee Giles et al. 1998. CiteSeer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*.

[32] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *ICCV*. 4852–4861.

[33] Zhiwei Guo and Heng Wang. 2020. A deep graph neural network-based mechanism for social recommendations. *IEEE Transactions on Industrial Informatics* (2020).

[34] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[35] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[36] Peng Han et al. 2019. GCN-MF: disease-gene association identification by graph convolutional networks and matrix factorization. In *KDD*.

[37] N. Harada, N. Yoshikawa, A. Yoshida, and N. Yokoyama. 2004. Josephson latching driver with a low bit-error rate. *IEEE Transactions on Applied Superconductivity* 14, 4 (2004), 2031–2036. doi:10.1109/TASC.2004.837112

[38] Yuxing He, Naoki Takeuchi, and Nobuyuki Yoshikawa. 2020. Low-latency power-dividing clocking scheme for adiabatic quantum-flux-parametron logic. *Applied Physics Letters* (2020). arXiv:https://doi.org/10.1063/5.0005612 https://doi.org/10.1063/5.0005612

[39] Zhezhi He and Deliang Fan. 2019. Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation. In *CVPR*.

[40] Yuki Hironaka, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2020. Demonstration of a single-flux-quantum microprocessor operating with Josephson-CMOS hybrid memory. *IEEE Transactions on Applied Superconductivity* 30, 7 (2020), 1–6.

[41] 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.

[42] Yu Huang et al. 2022. Accelerating graph convolutional networks using crossbar-based processing-in-memory architectures. In *HPCA*.

[43] Ranggi Hwang et al. 2023. GROW: A Row-Stationary Sparse-Dense GEMM Accelerator for Memory-Efficient Graph Convolutional Neural Networks. In *HPCA*.

[44] Weiwei Jiang and Jiayun Luo. 2022. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications* (2022), 117921.

[45] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. 2011. Quantum annealing with manufactured spins. *Nature* 473, 7346 (2011), 194–198.

[46] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[47] Kyounghoon Kim et al. 2015. Approximate de-randomizer for stochastic circuits. In *ISOCC*.

[48] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[49] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[50] Zhengang Li et al. 2023. SupeRBNN: Randomized Binary Neural Network Using Adiabatic Superconductor Josephson Devices. *arXiv:2309.12212* (2023).

[51] Shengwen Liang et al. 2020. Deepburning-gl: an automated framework for generating graph neural network accelerators. In *ICCAD*.

[52] Shengwen Liang et al. 2020. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Trans. Comput.* (2020).

[53] Xiaofan Lin et al. 2017. Towards accurate binary convolutional neural network. *NeurIPS* (2017).

[54] Zechun Liu et al. 2020. Reactnet: Towards precise binary neural network with generalized activation functions. In *ECCV*.

[55] K. Loe and E. Goto. 1985. Analysis of flux input and output Josephson pair device. *IEEE Transactions on Magnetics* (1985).

[56] Yixuan Luo et al. 2022. CoDG-ReRAM: An Algorithm-Hardware Co-design to Accelerate Semi-Structured GNNs on ReRAM. In *ICCD*.

[57] Andrew Kachites McCallum et al. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* (2000).

[58] Adam N. McCaughan and Karl K. Berggren. 2014. A Superconducting-Nanowire Three-Terminal Electrothermal Device. *Nano Letters* 14, 10 (2014), 5748–5753. doi:10.1021/nl502629x arXiv:https://doi.org/10.1021/nl502629x PMID: 25233488.

[59] S. Nagasawa et al. 2005. Reliability evaluation of Nb 10 kA/cm2 fabrication process for large-scale SFQ circuits. *Physica C: Superconductivity and its Applications* (2005).

[60] SS Teja Nibhanupudi, Siddhartha Raman Sundara Raman, Mikaël Cassé, Louis Hutin, and Jaydeep P Kulkarni. 2021. Ultra-low-voltage UTBB-SOI-based, pseudo-static storage circuits for cryogenic CMOS applications. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 7, 2 (2021), 201–208.

[61] Divya Prasad, Manoj Vangala, Mudit Bhargava, Arnout Beckers, Alexander Grill, Davide Tierno, Krishnendra Nathella, Thanusree Achuthan, David Pietromonaco, James Myers, et al. 2022. Cryo-computing for infrastructure applications: A technology-to-microarchitecture co-optimization study. In *2022 International Electron Devices Meeting (IEDM)*. IEEE, 23–5.

[62] Haotong Qin et al. 2020. Forward and backward information retention for accurate binary neural networks. In *CVPR*.

[63] Mohammad Rastegari et al. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

[64] Rakshith Saligram, Suman Datta, and Arijit Raychowdhury. 2021. CryoMem: A 4K-300K 1.3 GHz eDRAM macro with hybrid 2T-gain-cell in a 28nm logic process for cryogenic applications. In *2021 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1–2.

[65] Rakshith Saligram, Divya Prasad, David Pietromonaco, Arijit Raychowdhury, and Brian Cline. 2021. A 64-bit arm CPU at cryogenic temperatures: Design technology co-optimization for power and performance. In *2021 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1–2.

[66] Rishov Sarkar, Stefan Abi-Karam, Yuqi He, Lakshmi Sathidevi, and Cong Hao. 2022. FlowGNN: A Dataflow Architecture for Universal Graph Neural Network Inference via Multi-Queue Streaming. *arXiv preprint arXiv:2204.13103* (2022).

[67] Rishov Sarkar, Stefan Abi-Karam, Yuqi He, Lakshmi Sathidevi, and Cong Hao. 2023. FlowGNN: A Dataflow Architecture for Real-Time Workload-Agnostic Graph Neural Network Inference. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1099–1112.

[68] Oleg Semenov, Arman Vassighi, and Manoj Sachdev. 2002. Impact of technology scaling on thermal behavior of leakage current in sub-quarter micron MOSFETs: perspective of low temperature current testing. *Microelectronics Journal* 33, 11 (2002), 985–994.

[69] Prithviraj Sen et al. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[70] Weijing Shi and Raj Rajkumar. 2020. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *CVPR*.

[71] Xiaowu Sun et al. 2019. Formal verification of neural network controlled autonomous systems. In *HSCC*.

[72] Hiroshi Takayama, Naoki Takeuchi, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2018. A random-access-memory cell based on quantum flux parametron with three control lines. In *Journal of Physics: Conference Series*, Vol. 1054. IOP Publishing, 012063.

[73] Naoki Takeuchi et al. 2013. Measurement of 10 zJ energy dissipation of adiabatic quantum-flux-parametron logic using a superconducting resonator. *Applied Physics Letters* (2013).

[74] N. Takeuchi et al. 2014. Reversible logic gate using adiabatic superconducting devices. *Scientific Reports* 4 (2014).

[75] 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* (2013).

[76] Naoki Takeuchi, Taiki Yamae, Taro Yamashita, Tsuyoshi Yamamoto, and Nobuyuki Yoshikawa. 2023. Scalable quantum-bit controller using adiabatic superconductor logic. arXiv:2310.06544 [physics.app-ph]

[77] Naoki Takeuchi, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2015. Adiabatic quantum-flux-parametron cell library adopting minimalist design. *Journal of Applied Physics* 117, 17 (5 2015), 173912.

[78] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[79] M Mitchell Waldrop. 2016. The chips are down for Moores law. *Nature News* 530, 7589 (2016).

[80] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. *Proceedings of Machine Learning and Systems* 4 (2022), 673–693.

[81] Junfu Wang et al. 2021. Bi-gcn: Binary graph convolutional network. In *CVPR*.

[82] Yuke Wang et al. 2021. GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on GPUs. In *OSDI*.

[83] Zonghan Wu et al. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[84] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[85] Tomoharu Yamauchi et al. 2023. Design and Implementation of Energy-Efficient Binary Neural Networks Using Adiabatic Quantum-Flux-Parametron Logic. *TAS* (2023).

[86] Mingyu Yan et al. 2020. Hygcn: A gcn accelerator with hybrid architecture. In *HPCA*.

[87] Jiwei Yang et al. 2019. Quantization networks. In *CVPR*.
[88] Haoran You et al. 2022. Gcod: Graph convolutional network acceleration via dedicated algorithm and accelerator co-design. In *HPCA*.
[89] Sixing Yu et al. 2022. Topology-Aware Network Pruning using Multi-stage Graph Embedding and Reinforcement Learning. In *ICML*.
[90] Yongan Zhang et al. 2021. G-CoS: Gnn-accelerator co-search towards both better accuracy and efficiency. In *ICCAD*.
[91] Yan Zhang, Dharmesh Parikh, Karthik Sankaranarayanan, Kevin Skadron, and Mircea Stan. 2003. *Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects.* Technical Report. Citeseer.
[92] Ling Zhao et al. 2019. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems* (2019).
[93] Shuchang Zhou, Yuxin Wu, et al. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160* (2016).
[94] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2017. Trained ternary quantization. In *ICLR*.