2025 IEEE International Symposium on Workload Characterization (IISWC) | 979-8-3315-4917-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/IISWC66894.2025.00026

HALO: Hybrid Systolic Arrays via Logical Partitioning for Acceleration of Complex-Valued Neural Networks

Ji Yeong Yi[1*], Eunbi Jeong[1*], SungHee Yum[1], Jane Rhee[1], Sangun Choi[2], Gunjae Koo[3], Yunho Oh[2], Myung Kuk Yoon[1]

[1]*Division of Artificial Intelligence and Software, Ewha Womans University*

[2]*School of Electrical Engineering, Korea University*

[3]*Department of Computer Science and Engineering, Korea University*

Abstract

Complex-Valued Neural Networks (CVNNs) are an emerging class of deep learning models that process data with both real and imaginary components. By efficiently handling complex-valued representations, CVNNs have gained attention as a promising alternative to traditional Real-Valued Neural Networks (RVNNs), especially in domains such as signal processing and communications. A fundamental distinction between CVNNs and RVNNs lies in the use of Complex General Matrix-Matrix Multiplications (CGEMMs), each comprising four real-valued GEMMs along with additional operations. As such, CGEMMs impose substantial compute and memory burdens on CVNNs. Although modern systolic arrays have evolved to enhance GEMM performance with their high compute throughput, these architectures are suboptimal for CGEMMs due to two key limitations: (1) redundant data fetches and (2) underutilization of Processing Elements (PEs) within the array. To address these challenges, we propose HALO, a novel systolic array architecture that accelerates CGEMM execution through logical partitioning of the array. HALO divides a single array into logical sub-arrays, enabling concurrent execution of CGEMM sub-operations, thereby aggressively utilizing given PE resources with simple hardware modifications. We explore two execution modes of HALO: Half Mode and Quad Mode. Half Mode reduces duplicated data fetches and improves PE utilization, whereas Quad Mode offers even higher utilization but does not address the redundant memory access issue. To maximize performance across layers, HALO switches between the two modes on a per-layer basis, leveraging the proposed mode selection algorithm. Our evaluation demonstrates that HALO improves performance by 44.3% and achieves a 32.3% reduction in energy-delay product.

1. Introduction

With the end of Dennard Scaling and Moore's Law, domain-specific hardware accelerators have risen to meet the surging demands for computation and memory across various applications [23, 29, 33, 43, 48, 49]. Recent advances in deep learning have further fueled this development, driving the widespread adoption of accelerators—Google TPUs being a representative example [31, 55]. To process the massive amount of computations efficiently, these accelerators often employ large-sized systolic arrays, a design known as the *scale-up* architecture [2, 46, 62]. Due to their high compute throughput, such architectures are widely used for Deep Neural Network (DNN) workloads.

Traditionally, DNNs operate on real-valued data and are thus referred to as Real-Valued Neural Networks (RVNNs). While RVNNs have excelled in many tasks, they struggle to effectively handle inherently complex-valued data, which are prevalent in a wide range of application domains, such as telecommunications, bioinformatics, radar, and speech recognition [9, 24, 35, 53]. To overcome this limitation, recent studies have proposed Complex-Valued Neural Networks (CVNNs) designed to process complex numbers throughout their layers [9, 21, 24, 35]. CVNNs have demonstrated significant improvements for tasks where complex-valued representations are more natural and expressive. Consequently, CVNNs are emerging as a promising class of models in the field of deep learning.

Despite the growing success of CVNNs [6, 10, 25, 39], recent efforts in systolic array architectures have primarily focused on optimizing General Matrix-Matrix Multiplications (GEMMs), leaving Complex GEMMs (CGEMMs) relatively underexplored. Given that CGEMM follows a different computation procedure from GEMM, there is a clear need to optimize its execution on systolic arrays—an inherently difficult work due to substantial memory and computational overheads. CGEMM operations are more demanding than their real-valued counterparts; they involve four matrices, two each from the input and weight, thus doubling the data footprint. Moreover, a single CGEMM requires four real-valued GEMMs and additional operations, making the total computation nearly four times greater than that of a real-valued GEMM. As a result, these memory and computation hurdles make it challenging to deploy CVNNs.

Identifying challenges of executing CGEMM (§3). This work begins by identifying two key bottlenecks in executing CGEMM on systolic array architectures. First, due to the nature of complex arithmetic, CGEMM computations involve redundant data loading throughout execution. The conventional approach to computing CGEMM on systolic arrays decomposes a single CGEMM operation into four real-valued sub-operations [57]. Specifically, computing the imaginary output requires multiplying the real part of the input with the imaginary part of the weight, and

[*] Ji Yeong Yi and Eunbi Jeong contributed equally to this research.

2835-2238/25/$31.00 ©2025 IEEE
DOI 10.1109/IISWC66894.2025.00026

accumulating it with the product of the imaginary part of the input and the real part of the weight. The real output is computed through a similar process. As each input and weight matrix contributes to two different output components, the same matrix data must be reloaded across the four sub-operations. This redundancy results in excessive access to the power-hungry on-chip buffers, degrading performance and reducing energy efficiency.

Second, systolic arrays are often underutilized during CGEMM execution, leaving substantial performance potential untapped. In general, the matrices involved in GEMM (or CGEMM) are divided into smaller tiles, as their sizes rarely match with the array size [8, 57]. Under this tiling scheme, tiles located near the edges or corners may be smaller than the array dimensions, causing portions of the Processing Elements (PEs) to remain idle during computation.

Such *PE underutilization* issue has been discussed in several prior studies [15, 38], particularly in the context of computing GEMMs. In this work, we analyze this problem for the first time in the context of CGEMMs. To mitigate underutilization, prior work has explored alternative design strategies that deviate from the conventional scale-up design. One such approach is the *scale-out* architecture, which employs multiple smaller systolic arrays instead of relying on a few large ones. Although scale-out designs deliver performance gains by enhancing PE utilization, they incur increased memory traffic due to extensive off-chip fetches, undermining energy efficiency. To assess the feasibility of scale-out architectures for CGEMM computations, we evaluate both the performance and memory usage of CVNNs when executing CGEMM on such designs. Our results reveal that scale-out designs offer improved PE utilization for CGEMMs, but they also lead to a substantial increase in memory accesses, which is critical due to the inherent duplication of data fetches. Thus, scale-out is not a viable solution to the underutilization problem in CGEMM and fails to address the redundant data fetch issue as well.

Exploring logical partitioning schemes (§4.1). To tackle these challenges, we explore a novel scheme that executes CGEMM on systolic arrays by logically partitioning a single array into multiple sub-arrays. Unlike scale-out designs, this approach requires no additional PE resources and/or separate SRAM buffers; instead, it makes optimal use of the existing systolic array while providing the illusion of multiple sub-arrays. Leveraging the fact that CGEMM is composed of four GEMM operations, we logically divide the array and assign each sub-array to compute a different sub-operation by loading appropriately rearranged matrix tiles. The benefits from this approach are twofold: (1) by emulating the behavior of scale-out designs, it achieves PE utilization comparable to physically scaled-out architectures; and (2) by executing multiple sub-operations concurrently, it can potentially avoid redundant fetches of matrices, thereby improving both performance and energy efficiency.

We specifically introduce two logical partitioning schemes. First, the *halving* scheme divides the array horizontally into two partitions, enabling two sub-operations to be computed in a single pass. Under this scheme, one CGEMM operation is completed over two computation phases—one for the real output and one for the imaginary output. Second, the *quartering* scheme splits the array into four partitions, allowing all four sub-operations to be executed concurrently in a single computation phase. While both schemes improve PE utilization by more finely leveraging the compute resources within the array, each has a distinct advantage. The halving scheme eliminates redundant weight fetches by reusing the same weights across the two computation phases. In contrast, the quartering scheme does not reuse weights, yet it achieves higher PE utilization due to its finer-grained partitioning. This leads to the question: which scheme should be used. To answer this, we compare the performance of two schemes across CVNNs, analyzing their impact on each layer. Interestingly, we observe that the optimal scheme varies across layers, suggesting that selectively using both could result in better overall performance. Furthermore, we find that both partitioning schemes can be simply implemented on conventional systolic arrays with only minor architectural supports and can even coexist within a unified architecture—without requiring separate arrays for each scheme.

Introducing hybrid systolic array (§4.2). Building on these insights, we propose `HALO`, a **H**ybrid Systolic **A**rray Architecture via **LO**gical Partitioning to maximize CGEMM performance on systolic arrays. We first introduce two execution modes—`Half Mode` and `Quad Mode`—based on the proposed halving and quartering schemes, respectively. To support CGEMM computations through logical partitioning, we present a negation logic that enables seamless accumulation of the imaginary product with its corresponding real counterpart. We then analyze the root causes behind the distinct advantages of the two schemes and introduce a mode selection strategy that determines the optimal execution mode on a per-layer basis. The optimal mode for each layer is identified using the proposed selection algorithm, and the corresponding configuration is provided to the hardware, enabling the array to operate in either `Half Mode` or `Quad Mode` accordingly.

To validate the effectiveness of our proposed architecture, we evaluate `HALO` on a set of CVNN workloads, including several complex-valued CNNs and the state-of-the-art Transformer model. Our evaluation shows that `HALO` achieves a speedup of 44.3% over the baseline and improves energy efficiency, reducing energy consumption and energy-delay product by 10.0% and 32.3%, respectively.

2. Background

In this section, we first introduce CVNNs, highlighting their benefits over RVNNs and their distinctive computational core, CGEMM. Then, we describe the baseline systolic array architecture.

2.1. Complex-Valued Neural Network

A CVNN is a type of neural network designed to process complex-valued data, where both the weights and inputs are complex numbers. In contrast to RVNNs, which can

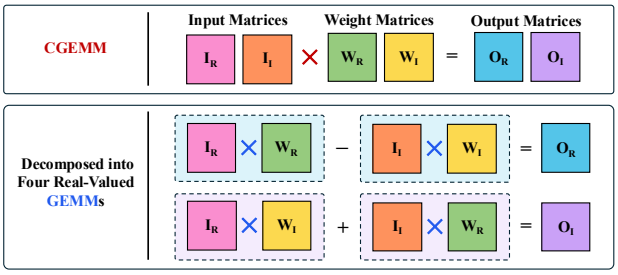


Figure 1: Computational process of CGEMM, decomposed into four real-valued GEMMs.

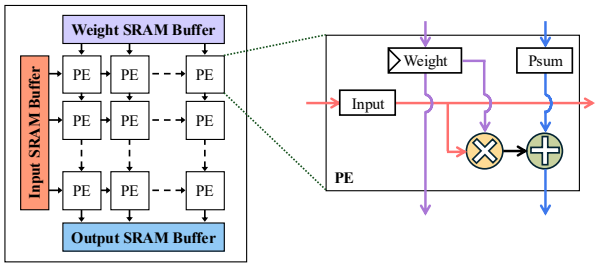


Figure 2: Weight stationary systolic array architecture.

only represent magnitude, CVNNs provide a richer representational capacity that is particularly effective for data in the complex domain [9, 11, 14, 44]. Due to these benefits, CVNNs have gained popularity in domains such as telecommunications, signal processing, speech recognition [6, 7, 41]. A key advantage of CVNNs lies in the arithmetic properties of complex multiplication, which inherently combines phase rotation and amplitude attenuation. This allows CVNNs to jointly process directional and intensity information, enabling more compact and expressive representations of complex-valued data. As a result, CVNNs can represent complex-valued relationships using fewer trainable parameters, thereby reducing the model's degrees of freedom and improving generalization ability [25]. Moreover, compared to RVNNs, training CVNNs on complex-valued data has been demonstrated to offer faster convergence and noise-robust memory mechanisms [5, 70].

The core computation in CVNNs is a CGEMM, whereas in RVNNs it is the standard GEMM [4, 8, 32, 45, 47, 60]. As depicted in Figure 1, the CGEMM is a multiplication between complex-valued input and weight matrices, each composed of two real-valued matrices—one for the real part and one for the imaginary part—denoted as $I = I_R + iI_I$ and $W = W_R + iW_I$. A single CGEMM operation is decomposed into four real-valued GEMMs: $I_R \times W_R$, $I_R \times W_I$, $I_I \times W_R$, and $I_I \times W_I$. The final output is computed by combining these intermediate results; the real part of the output, O_R, is obtained by subtracting $I_I \times W_I$ from $I_R \times W_R$, and the imaginary part, O_I, is obtained by adding $I_R \times W_I$ and $I_I \times W_R$. Such a CGEMM operation enables CVNNs to preserve the phase information inherent in complex-valued representations [1, 56]. This benefit, however, comes at the cost of significantly higher computational complexity due to the arithmetic structure of complex numbers.

2.2. Systolic Array

A systolic array is a parallel hardware architecture widely adopted as the core compute engine in accelerators [12, 37, 68]. It consists of a two-dimensional grid of PEs, each responsible for performing Multiply-Accumulate (MAC) operations. Systolic arrays support spatial data reuse by passing data between neighboring PEs in a pipeline fashion, effectively reducing off-chip memory accesses. The mapping of GEMM operations onto a systolic array is governed by the choice of dataflow that determines which data is retained within each PE and which is propagated across the array. There are three representative dataflows: *weight stationary*, *input stationary*, and *output stationary*—each optimized for reusing weights, input activations, and partial sums, respectively [46, 59, 62]. In this work, we adopt the weight stationary dataflow as the baseline, as it is widely used in systolic array architectures, including TPUs, which serve as representative hardware for deep learning workloads [2, 19, 26, 27, 30].

Figure 2 shows a systolic array architecture that retains weights locally within each PE to enable weight reuse during computation. Before the computation begins, weights are prefilled into the PEs, starting from the topmost row. At each clock cycle, weight values are fetched from the weight buffer to the top row of PEs and propagated downward along the columns; each PE receives its weight either directly from the weight SRAM buffer (top row) or from the PE directly above (lower rows), and stores it in its local register. The locally stored weights are then reused throughout the computation. After the prefill step, inputs are streamed into the leftmost column and propagated rightward across the array, where each PE performs a MAC operation at every cycle using the stored weight and the incoming input. The partial sums generated by each PE are propagated downward along the columns and accumulated. Once the final outputs are produced at the bottom row, they are stored into the output buffer and PEs are updated with new weights for the next layer's computation.

In typical GEMM workloads, the operand matrices often exceed the capacity of a single systolic array. When the dimensions of input and weight matrices exceed the physical size of a systolic array, the matrices are partitioned into tiles that match the array's spatial dimensions [69]. In weight stationary architectures, the weight matrix W ($K \times N$) is spatially mapped onto the array, requiring it to be divided into tiles of size $S_R \times S_C$, where S_R and S_C denote the number of rows and columns of the array, respectively. Because the inner dimension K is shared between the input matrix I ($M \times K$) and the weight matrix W, both matrices must be partitioned along this axis. The operations on these tiles can be executed sequentially on a single array or in parallel across multiple arrays, depending on available hardware resources and the system scheduler.

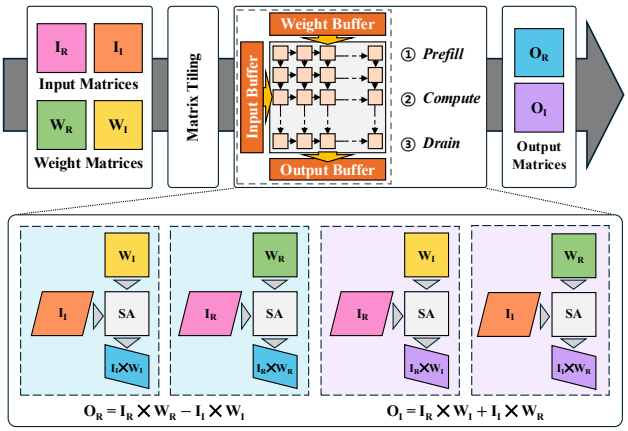


Figure 3: Execution flow and sub-operation breakdown of CGEMM on a systolic array.

3. Motivation

In this section, we first describe the baseline CGEMM computation method on systolic arrays in detail. Then, we identify two key challenges in executing CGEMM on conventional systolic arrays. Finally, we motivate the need for a new architectural solution to efficiently accelerate CGEMM on systolic arrays.

3.1. CGEMM Execution on Systolic Arrays

To perform CGEMM using systolic arrays similar to that employed in TPUs, four real-valued GEMM operations must be executed: $I_R \times W_R$, $I_I \times W_I$, $I_R \times W_I$, and $I_I \times W_R$, as illustrated in Figure 3. Each of these GEMMs may require *matrix tiling* when the matrix dimensions exceed the size of the systolic array. As all four GEMMs share the same dimensions, the number of tiles required is identical across all operations. To produce the final complex-valued output, the results of the four real-valued GEMMs are combined following the CGEMM arithmetic rules: subtraction for the real part ($I_R \times W_R - I_I \times W_I$) and addition for the imaginary part ($I_R \times W_I + I_I \times W_R$). For the imaginary part, the addition between $I_R \times W_I$ and $I_I \times W_R$ can be efficiently handled by storing the intermediate result of $I_R \times W_I$ in the output buffer. Then, during the execution of the $I_I \times W_R$, the stored result can be fetched and added on-the-fly, allowing the final imaginary output to be computed without requiring an additional post-processing accumulation step. For the real part, to eliminate the need for a separate subtraction stage and to leverage the addition logic already present in the PEs, we first compute the $I_I \times W_I$, negate its output, and store it in the output buffer before initiating the $I_R \times W_R$. Then, during the execution of $I_R \times W_R$, the negated intermediate result is fetched and added on-the-fly. This approach enables the final real output to be computed without necessitating an explicit post-processing subtraction.

3.2. Challenges

Redundant data access. While the aforementioned approach intuitively follows the structure of complex multiplication, it requires each component matrix of the inputs and weights to be loaded repetitively from the SRAM buffer into the array. As shown in the bottom box of Figure 3, the imaginary weight matrix (W_I; yellow) is loaded for the first and third sub-operations, while the real weight matrix (W_R; green) is fetched for the second and fourth. Hence, each weight matrix is loaded twice, leading to redundant memory accesses. The rationale behind this redundancy is that CGEMM produces two output matrices—one for the real part and another for the imaginary part—and all four component matrices contribute to both outputs, thus requiring *double loads*. To sum up, the first challenge in computing CGEMM on systolic arrays is the redundant data fetches from the energy-intensive SRAM buffer, which degrades both performance and energy efficiency.

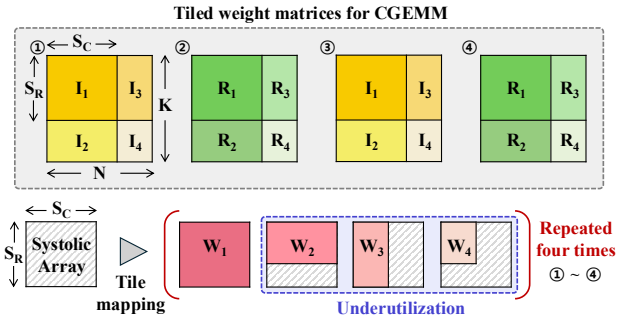


Figure 4: Underutilization of systolic arrays caused by matrix tiling during CGEMM execution.

PE underutilization. Conventional systolic array architectures typically consist of a small number of large-sized pods. For instance, TPUv3 and v4i employ four 128×128 arrays [69]. This type of systolic array design, characterized by large pods, is called *scale-up* architecture [2, 28, 54]. Scale-up designs offer high throughput by executing MAC operations on large matrix tiles within a single large pod. The larger array allows each fetched input or weight tile to be used in more computations across the PEs within the array, amortizing the cost of memory accesses. However, such designs have one critical drawback: as operand matrix sizes do not often align with the array's spatial dimension, many PEs within each array often remain idle, resulting in *PE underutilization*. While this issue has been noticed in prior work [15, 38], we observe that it becomes further exacerbated in CGEMM computations, which require four sub-operations per a single computation.

Figure 4 illustrates an example execution scenario where the tiling of matrices in CGEMM leads to PE underutilization due to a mismatch between the matrix and array sizes. Under the matrix tiling scheme, each GEMM sub-operation is executed through multiple passes, with each pass processing a specific pair of input and weight tiles. The upper part of the figure shows four tiled weight matrices (①, ②, ③, and ④), each corresponding to a sub-operation in CGEMM, while the lower part depicts how these tiles are mapped onto the array during a single sub-operation. As indicated by the four red boxes—each corresponding to W_1, W_2, W_3,

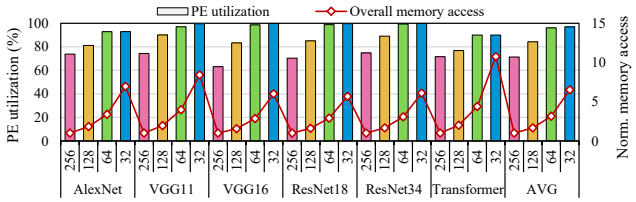


Figure 5: PE utilization and overall memory access across varying systolic array sizes.

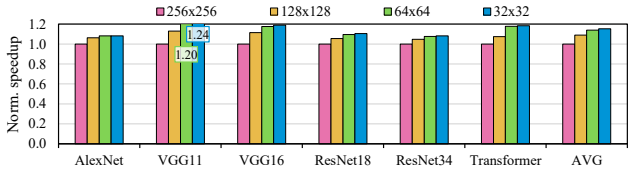


Figure 6: Speedup across varying systolic array sizes.

and W_4—portions of the second to fourth pods (hatched areas) remain unassigned by the tiled matrix elements, leaving the PEs in those regions *inactive*. Since this process repeats for all four sub-operations, the total number of idle cycles across all PEs increases by approximately four times compared to a real-valued GEMM. Consequently, the second challenge is the inability of existing architectures to fully harness existing PE resources, resulting in significant underutilization and suboptimal performance.

Can scaling-out help? The redundant data access problem is clearly unavoidable under the baseline CGEMM computation, whereas the PE underutilization may be alleviated by adopting *scale-out* architectures instead of conventional scale-up designs. Scale-out designs employ many small-sized pods rather than a few large ones [13, 69]. They enable *fine-grained* mapping of tiled matrices onto arrays, thereby achieving heightened PE utilization and improved performance. However, scaling out incurs substantial memory traffic, because multiple pods often fetch the same matrix tiles within a short time window, resulting in increased energy consumption and bandwidth pressure [13, 66, 69].

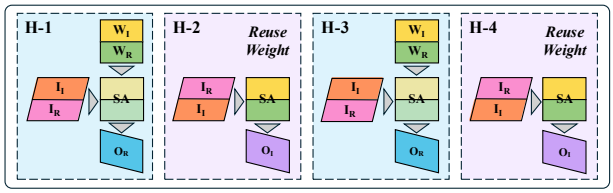
To analyze the trade-off between scale-up and scale-out designs for CVNNs, we evaluate both PE utilization and overall memory access across four systolic array configurations: 256×256 (1 pod), 128×128 (4 pods), 64×64 (16 pods), and 32×32 (64 pods), all of which are set to have the same total number of PEs (65,536). See §5.1 for a detailed evaluation methodology. Figure 5 shows the average number of active PEs over the total number of PEs within active pods (PE utilization) and overall memory access, including both SRAM and off-chip accesses, normalized to the baseline 256×256 configuration. The 256×256 array exhibits an average PE utilization of 71.3%, indicating that many PEs within pods are wasted throughout execution. Reducing the array size to 128×128 improves PE utilization by 12.9 Percentage Points (PP), but results in 1.67× increase in memory access. Further scaling down to 64×64 enhances PE utilization by 24.8 PP through finer-grained matrix-to-array mapping, but incurs 3.15× more memory accesses. Notably, such improvements in PE utilization begin to saturate beyond the 64×64 configuration; the 32×32 provides only an additional 0.8 PP of utilization improvement over the 64×64. Moreover, 32×32 configuration suffers from a significant increase in memory accesses, resulting in 6.52× higher memory overhead compared to the baseline. Note that this overhead worsens in CVNNs, where both weight and input matrices must be redundantly loaded across the four GEMM operations in a single CGEMM computation (Figure 3). Therefore, although scale-out architecture significantly improves PE utilization, it sacrifices off-chip memory bandwidth efficiency, making it infeasible for CVNN acceleration.

One step further, we assess the performance across the same array configurations used in Figure 5, to better understand the resulting performance under the aforementioned trade-off. Figure 6 shows the speedup results normalized to the baseline 256×256 configuration. Reducing the array dimension to 128×128 while quadrupling the number of pods offers a 1.09× speedup by enabling more efficient PE usage with reduced idle cycles. The 64×64 configuration further improves performance, achieving a 1.14× speedup. Performance improvements start to plateau beyond the 64×64 configuration, consistent with the PE utilization results; the 32×32 only offers an additional 1.4 PP of speedup over the 64×64. This is attributed to the substantial increase in memory accesses and the negligible improvement in PE utilization, as presented in Figure 5.

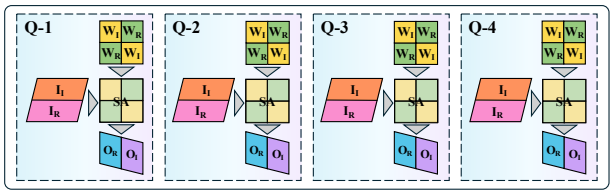
To summarize, CGEMM is a fundamental operation in CVNNs, yet existing systolic array architectures fall short in executing it efficiently. Conventional scale-up designs suffer from poor PE utilization due to *coarse-grained* tiling, limiting their performance. In contrast, scale-out designs improve PE utilization through *fine-grained* mapping but incur significant off-chip memory access overhead. Furthermore, neither architecture addresses the redundant data fetching problem inherent in the baseline CGEMM computation method. To overcome these limitations, we propose a novel systolic array architecture that enables a scale-up array to behave like a scale-out design without physically partitioning the large array into smaller pods. The proposed architecture aims to maximize PE utilization while reducing redundant data loads, thereby delivering both high performance and energy efficiency for CGEMM computations.

4. HALO Architecture

In this section, we propose `HALO`, a novel systolic array architecture for executing CGEMMs through logical partitioning of arrays. We first explore two design spaces, each leveraging a different logically partitioned layout. To maximize performance across various CGEMM layers, we introduce a hybrid systolic array supporting two execution modes—`Half Mode` and `Quad Mode`—and present architectural support and a mode selection algorithm to determine an optimal mode per layer.



(a) Halving systolic array



(b) Quartering systolic array

Figure 7: Overview of computation flows under two logical partitioning schemes, halving and quartering systolic array.

4.1. Logical Partitioning of Systolic Array

Scale-out architectures may offer a potential solution to address the limitations of conventional systolic arrays in executing CGEMM efficiently. However, we observe that adopting such architectures would not be practical, as they incur excessive memory accesses in both on-chip and off-chip memory and still fail to eliminate redundant data fetches. Instead of physically dividing a large array into smaller arrays with dedicated buffers, as in scale-out designs, we explore a novel alternative approach: logically partitioning a single large array into multiple independent sub-arrays. Considering a single CGEMM computation consists of four GEMMs, we propose executing multiple sub-operations concurrently within the array by assigning each to a different logical partition. The proposed approach is driven by two key insights: (1) PE utilization can be improved by rearranging the layout of weight and input matrix tiles to logically partition the array into independent sub-arrays, each responsible for a different sub-operation; and (2) redundant weight loads can potentially be avoided by simultaneously loading all four component matrices involved in a CGEMM into the PEs, enabling multiple sub-operations to be merged and executed in parallel. Leveraging these insights, we introduce two logical partitioning schemes: one that divides the array into two partitions, and another that splits it into four.

Halving systolic array. This scheme logically divides the array horizontally into two partitions and uses them to compute partial terms for a single output component (either real or imaginary). Figure 7a illustrates the overall process of CGEMM computation using a halved systolic array. We begin by prefilling the array with weights such that the upper partition stores the imaginary weights (W_I), and the lower partition stores the real weights (W_R). The array first computes the real output by streaming in both the real and imaginary components of the input matrix (**H-1**). Specifically, the real input (I_R) is paired with the lower partition ($I_R \times W_R$), and the imaginary input (I_I) is paired with the upper partition ($I_I \times W_I$). The two resulting partial outputs are then accumulated within the array to produce the real output ($I_R \times W_R - I_I \times W_I$), with a negation applied to the upper partition—this mechanism will be described in detail in §4.2. Next, the array computes the imaginary output without flushing the weights (**H-2**). Retaining and reusing the prefilled weights in both partitions, the array streams in a new input. That is, I_I is paired with the lower partition ($I_I \times W_R$), and I_R is paired with the upper partition ($I_R \times W_I$). This is done simply by modifying the input stream fed into the array. Note that the figure shows the steps required to compute the same throughput as in Figure 3, and thus the two-step process is repeated in **H-3** and **H-4**. To sum up, this halving scheme computes the real and imaginary outputs sequentially using the same weights, thereby improving PE utilization and eliminating redundant weight fetches from the SRAM buffer.

Quartering systolic array. This scheme partitions the array into four sub-arrays, each computing one of the four sub-operations of the CGEMM computation. By executing all four sub-operations concurrently, the array can compute both the real and imaginary output component matrices in a single pass using the prefilled weights. As shown in Figure 7b, weights are again loaded in a *mixed-component* format, with each weight duplicated diagonally across partitions. The imaginary input (I_I) is streamed into the upper half of the array, while the real input (I_R) is streamed into the lower half, forming the necessary operand combinations for each sub-operation. That is, I_R is paired with both the imaginary weights (W_I) and the real weights (W_R), allowing the two lower partitions to compute $I_R \times W_I$ and $I_R \times W_R$. Likewise, I_I is also paired with both the W_I and W_R, enabling two upper partitions to compute $I_I \times W_I$ and $I_I \times W_R$, respectively. Similar to the halved systolic array, this scheme also requires a negation step for the product of two imaginary operands (*i.e.,* the result of $I_I \times W_I$ must be negated). With this negation mechanism, all partial products are accumulated within the array, resulting in the real and imaginary output components being computed on the left and right sides, respectively (**Q-1**). As the figure illustrates a CGEMM computation with the same output throughput as the halving scheme in Figure 7a, this quartered execution process is repeated in the following three steps (**Q-2**, **Q-3**, and **Q-4**). Note that, unlike the halving scheme, the quartering scheme does not reuse weights across phases, as the same real and imaginary components must be duplicated to fill different partitions. Nonetheless, this design utilizes PE resources more aggressively by dividing the array into four small partitions. Thus, quartering the systolic array enables more efficient use of PEs by concurrently computing both output components through fine-grained logical partitioning.

Halving or quartering? Both halving and quartering the systolic array effectively enhance PE utilization by logically partitioning a single array into multiple sub-arrays. The halved array not only improves PE utilization but also

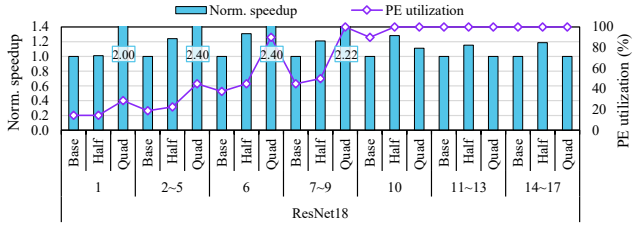


Figure 8: Normalized speedup and PE utilization across CGEMM layers in ResNet18.

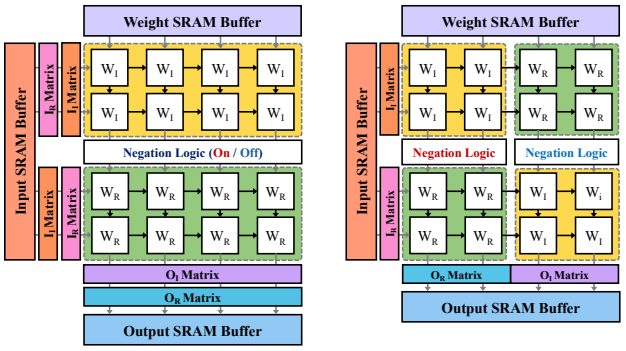
removes redundant weight fetches from the SRAM buffer, while the quartered array achieves even higher utilization by leveraging twice as many logical partitions. To further explore the design of HALO architecture, we compare the performance of CGEMM computations under these two logical partitioning schemes, referred to as Half and Quad.

Figure 8 presents the normalized speedup and PE utilization results for 17 CGEMM layers in ResNet18, where consecutive layers with same matrix dimensions are grouped within the same column. Across layers 1 to 9, Quad consistently outperforms Half, achieving 2.00×–2.40× speedups and showing 14.4–55.0 PP higher PE utilization over the Base (baseline). In contrast, for layers 10 to 17, where all configurations offer around maximum PE utilization, Half surpasses Quad, delivering 1.15×–1.28× speedups. As a result, while Quad achieves the highest average performance with improved PE utilization, the best-performing scheme varies across layers. Specifically, Quad performs best in layers with low to moderate PE utilization under the baseline (layers 1 to 9). On the other hand, Half is more effective in layers where the baseline already exhibits high utilization (layers 10 to 17). In such cases, Half enhances performance by eliminating redundant data fetches—an advantage that Quad cannot provide. These results indicate that the effectiveness of each scheme is largely dependent on the matrix dimensions of CGEMM layers. Note that although we provide results from ResNet18, similar trends are consistently observed across all CVNNs evaluated in this work.

To this end, we propose a hybrid systolic array architecture that switches between Half and Quad Mode to maximize CGEMM performance across the layers of CVNNs. In the following subsection, we provide the microarchitectural details for Half Mode and Quad Mode based on the previously discussed partitioning schemes, and then introduce the hybrid mechanism for selecting between the two modes.

4.2. Hybrid Systolic Array

To support CGEMM computations through logical partitioning on conventional systolic arrays, we incorporate two key modifications. First, the layout of input and weight matrices loaded into the array must be rearranged, ensuring that each operand tile is steered to its corresponding logical partition within the array. Second, to support partitioned CGEMM execution with minimal hardware overhead, HALO integrates Negation Logic, which enables the accumulation of intermediate results within the array.



(a) Half mode (b) Quad mode

Figure 9: Hybrid systolic array architecture.

Half mode. Half Mode computes the real and imaginary output matrices over two computation phases. To do this, the array is logically divided along the horizontal axis into two partitions, as depicted in Figure 9a. In the baseline CGEMM execution (§3.1), the real and imaginary weight matrices, W_R and W_I, each of size $K \times N$, are partitioned into tiles of size $S_R \times S_C$ to match the dimensions of the array. In Half Mode, however, each of W_R and W_I is divided into tiles of size $(S_R/2) \times S_C$. A pair of sub-tiles, one from W_R and one from W_I, is then stacked vertically to form a single $S_R \times S_C$ tile, which is prefilled into the array. Note that the sub-tile W_I must be placed in the upper half of the array to enable the negation of the imaginary product $(W_I \times I_I)$ via the Negation Logic placed along the middle row. Additionally, in cases where a tile from W_R or W_I contains fewer than $S_R/2$ rows or fewer than S_C columns, the tile is aligned to the top-left corner of its designated half. That is, W_I tiles are mapped to the upper half, and W_R tiles to the lower half, again ensuring the negation is applied to the imaginary product.

Once the weight tiles are loaded into the array, the lower half of the array (green box) contains W_R, and the upper half (yellow box) contains W_I, creating the illusion of two independent sub-arrays. The CGEMM operation is then executed in two computation phases, reusing the same weights while streaming input tiles with different layouts in each phase. To enable this, the input matrix of size $M \times K$ is partitioned column-wise into tiles of size $M \times h$, where h is the number of active rows in each half of the array. When all rows are filled with weights, h becomes $(S_R/2)$. Across the two computation phases, the positions of the input tiles from I_R and I_I are swapped when streamed into the array. In the first phase, the tile from I_I is streamed into the upper half of the array (prefilled with W_I), while the tile from I_R is streamed into the lower half (prefilled with W_R). This layout allows the computation of the real part of the output using $I_R \times W_R$ and $I_I \times W_I$, where the subtraction between the two products is performed by the Negation Logic at the center row (*i.e.,* Negation Logic is On). In the second phase, the positions of the input tiles are swapped; the tile from I_R is steered into the upper half of the array, and the tile from I_I into the lower half. This

enables to compute the imaginary part of the output using $I_I \times W_R$ and $I_R \times W_I$, where both terms are added directly without negation (*i.e.,* `Negation Logic` is `Off`).

Quad mode. `Quad Mode` computes all four CGEMM sub-operations concurrently by using four logical partitions of the systolic array in a single computation phase. The array is logically divided into four quadrants by splitting it evenly along both rows and columns, as shown in Figure 9b. In contrast to the `Half Mode`, which partitions weights into two halves of size $(S_R/2) \times S_C$, `Quad Mode` divides the real and imaginary weight matrices into smaller tiles of size $(S_R/2) \times (S_C/2)$. These tiles are then combined and loaded onto the array, with W_I (yellow boxes) placed in the upper-left and lower-right quadrants, and W_R (green boxes) in the upper-right and lower-left quadrants.

After the array is filled with these weights, the CGEMM operation is performed in a single computation phase. Input matrices are tiled into $M \times h$ tiles, as in `Half Mode`, with the imaginary input (I_I) streamed into the upper half of the array and the real input (I_R) into the lower half. This layout enables all four sub-operations to be computed in parallel and accumulated within the array. Specifically, the left half of the array, comprising the upper-left (W_I) and lower-left (W_R) quadrants, computes the real output as $I_R \times W_R - I_I \times W_I$. Similarly, the right half, composed of the upper-right (W_R) and lower-right (W_I) quadrants, computes the imaginary output as $I_I \times W_R + I_R \times W_I$. Since subtraction is required only for the real output (specifically $I_I \times W_I$ in the upper-left quadrant), the `Negation Logic` is selectively enabled only in the left half of the array (*i.e.,* `Negation Logic` is `On` in the left half and `Off` in the right half).

Selecting execution mode. To maximize performance across CGEMM layers in CVNNs, `HALO` selects between `Half Mode` and `Quad Mode` based on the matrix dimensions of each layer. The selection is guided by estimating the total cycles required to execute each mode. Here, *total cycles* refers to the sum of cycles needed for prefill, computation, and drain phases to complete the CGEMM execution. Given the same computational throughput, a lower total cycle count implies that the same amount of operations can be performed in less time, thereby improving performance. Note that memory stall cycles are not considered in this estimate, as they are highly variable and difficult to predict accurately. However, their impact on performance is negligible because systolic arrays effectively mitigate such stalls using *double-buffering* techniques [30, 50], which allow data preloading to overlap with ongoing computation.

Algorithm 1 details the procedure for selecting the execution mode yielding the lower total cycles. To measure the total cycles, two terms are first calculated: (1) the number of cycles required to process a single tile in each mode ($Cycle_{\text{mode}}$) and (2) the total number of tiles needed to cover the entire CGEMM layer ($Tile_{\text{mode}}$). Regarding the first, for `Half Mode`, S_R cycles are required for the prefill phase, and $2 \times (S_R + S_C + M - 2)$ cycles for computation and drain phases; likewise, for `Quad Mode`, S_R cycles are needed for prefill and $(S_R + S_C + M - 2)$ for computation and drain. The compute and drain cycles in `Half Mode` is doubled because its two-phase computation reuses prefilled weights. Regarding the second, both the row (S_R) and column (S_C) dimensions of the array are considered. For `Half Mode`, the K dimension is divided by S_R and then doubled ($2K/S_R$), reflecting the partitioned sub-tiles of size $(S_R/2) \times S_C$ for each real and imaginary component. Then, the N dimension is divided by S_C without doubling, and these two values are multiplied together. For `Quad Mode`, the calculation is similar, however, the column dimension is also doubled ($2N/S_C$), as this mode halves both the row and column dimensions (*i.e.,* sub-tiles of $(S_R/2) \times (S_C/2)$). Finally, the total cycle count ($Total_{\text{mode}}$) is calculated by multiplying the two terms, and the mode with the lower value is chosen as optimal.

Algorithm 1 Select Execution Mode in HALO

Input: Operand matrix dimensions (M, N, K), systolic array size (S_R, S_C)
Output: Selected execution mode: *half_mode* or *quad_mode*

$Cycle_{\text{half}} \leftarrow S_R + 2 \times (S_R + S_C + M - 2)$
$Tile_{\text{half}} \leftarrow \left\lceil \frac{2K}{S_R} \right\rceil \times \left\lceil \frac{N}{S_C} \right\rceil$
$Total_{\text{half}} \leftarrow Cycle_{\text{half}} \times Tile_{\text{half}}$
$Cycle_{\text{quad}} \leftarrow S_R + (S_R + S_C + M - 2)$
$Tile_{\text{quad}} \leftarrow \left\lceil \frac{2K}{S_R} \right\rceil \times \left\lceil \frac{2N}{S_C} \right\rceil$
$Total_{\text{quad}} \leftarrow Cycle_{\text{quad}} \times Tile_{\text{quad}}$
if $Total_{\text{quad}} < Total_{\text{half}}$ **then**
    $Mode \leftarrow quad\_mode$
else
    $Mode \leftarrow half\_mode$
end if
return *Mode*

Negation logic. To perform the operand negation for imaginary products, `HALO` places a 1-bit XOR gate along the data path at the midpoint of the systolic array (*i.e.,* at the row boundary between the upper and lower halves). Assuming a floating-point representation, the operand can be negated by flipping the sign bit using the XOR gate. Specifically, in `Half Mode`, the XOR gates are enabled during the first computation phase to negate the imaginary term ($I_I \times W_I$), and disabled in the second phase when the imaginary output is computed by direct addition. In `Quad Mode`, only the left half of the array requires negation for the real output computation, so the XOR gates are selectively activated in that region. This design enables in-place operand negation with marginal hardware overhead, as demonstrated in §5.5. By integrating this lightweight mechanism with mode-specific activation control, `HALO` efficiently supports both execution modes.

5. Evaluation

This section outlines our simulation methodology and presents the evaluation results for the proposed architecture. We comprehensively assess performance, memory usage, and energy efficiency to validate its effectiveness. Finally, we conduct a sensitivity study to analyze how the systolic array size impacts the performance of `HALO`.

TABLE 1: Baseline system parameters [27].

Component	Description
Systolic array	One 256×256 weight-stationary pod
On-chip SRAM buffer	6MB for input and weight, 4MB for output
Off-chip memory	HBM2 with a bandwidth of 614GB/s

5.1. Methodology

We evaluate our proposed design using a cycle-accurate systolic array simulator, SCALE-Sim v2 [50], which is modified to support CGEMM computations. The baseline system parameters are summarized in Table 1. To model the energy consumption, we configure the system to operate at a clock frequency of 1GHz and combine the simulation outputs from SCALE-Sim with energy modeling tools. Specifically, we use Accelergy [61] to estimate the dynamic and static energy consumption for MAC operations, SRAM buffer access, and off-chip memory access, and apply DeepScaleTool [51] to scale these values to the 7nm technology node, consistent with the fabrication process of TPUv4i [27]. Furthermore, we implement the introduced `Negation Logic` in Verilog and synthesize it using Synopsys Design Compiler with the FreePDK library [40] to obtain area estimates.

We run six CVNN models: `AlexNet`, `VGG11`, `VGG16`, `ResNet18`, `ResNet34`, and `Transformer` [65, 67]. These models have the same layer configurations as their RVNN counterparts [22, 34, 52, 58], except all matrix multiplications are implemented as CGEMMs. In our evaluations, we compare four configurations: (1) `Baseline`, (2) `Half Mode`, (3) `Quad Mode`, and (4) `HALO`. `Baseline` is a conventional systolic array architecture using the four-phase CGEMM execution (§3.1). `Half Mode` and `Quad Mode` partition the array into two and four logical sub-arrays, respectively, enabling CGEMM execution by fusing multiple sub-operations. `HALO` is the proposed architecture which switches between `Half` and `Quad Mode` for each layer, aiming to maximize performance across different layers in CVNNs.

5.2. Speedup and PE Utilization

Figure 10 presents the speedup and PE utilization results, with speedup normalized to the performance of the `Baseline`. For these two metrics, we also evaluate an additional configuration, Fold Set-Wise Scheduling (`FSWS`), which is one of the CVNN-aware scheduling techniques proposed in prior work [36]. Whereas the baseline scheduler executes each sub-operation one by one (*i.e.,* completing all tile operations for a sub-operation before moving to the next), `FSWS` prefills a weight tile and then sequentially loads the corresponding real and imaginary input tiles, thereby enabling the prefilled weight tile to be reused across two sub-operations. That is, `FSWS` groups the four tile operations required for a single complex-valued output tile (*i.e.,* its real and imaginary components) into a *fold set*, which is then scheduled sequentially to eliminate duplicated weight fills. Our evaluation shows that `FSWS` yields an 8.3% speedup over the `Baseline`, but its PE utilization is identical to that of the `Baseline`. This is because `FSWS` only addresses redundant

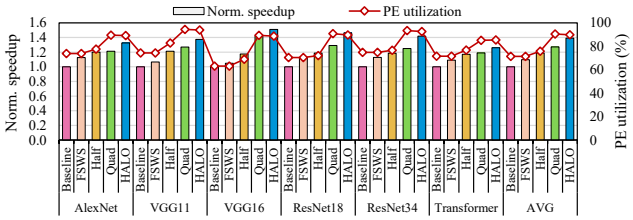


Figure 10: Normalized speedup and PE utilization.

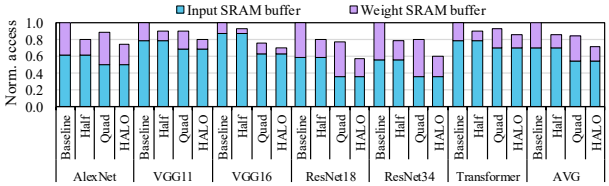


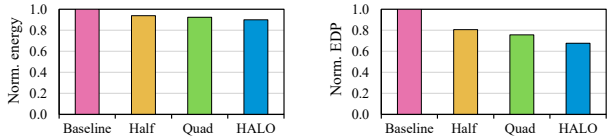
Figure 11: Normalized accesses to input and weight buffers.

weight loads and does not overcome the second challenge of executing CGEMM on systolic arrays—PE underutilization.

The main goal of our proposed architecture is to utilize PE resources more effectively such that these valuable resources are not left idle during CGEMM computations, ultimately accelerating CVNNs. As shown in the figure, `Half Mode` achieves an average speedup of 18.7%. This is attributed to two key benefits: (1) PE utilization increases by 4.5 PP over the `Baseline` through the logical partitioning scheme (a gain unattainable through `FSWS`); and (2) weights are reused across two computation phases, reducing the prefill time otherwise spent loading redundant values into the array. `Quad Mode` delivers a higher average speedup of 31.7%, further enhancing PE utilization by 19.1 PP due to the finer-grained array partitioning. Although `Quad Mode` outperforms `Half Mode` on average, there are layers where it offers little to no gain, while `Half Mode` provides significant performance improvements for them (§4.1). Therefore, by selecting the optimal mode on a per-layer basis, `HALO` achieves an overall speedup of 44.3% over the `Baseline`. It is worthwhile to note the PE utilization in `HALO` drops slightly (by 0.6 PP) compared to `Quad Mode`. This minor reduction occurs because some layers benefit more from reduced data fetches under `Half Mode` than from the higher PE utilization provided by `Quad Mode`. As such, `HALO` opts for `Half Mode` in those cases, leading to a modest decrease in overall utilization. Nonetheless, `HALO` still achieves 89.9% PE utilization, demonstrating that around 90% of PE resources are efficiently utilized throughout CGEMM computations. Consequently, the proposed hybrid systolic array architecture effectively enhances CVNN performance via a novel CGEMM execution strategy based on logical partitioning, requiring only minimal modifications to existing systolic array designs.

5.3. On-Chip Memory Usage

To assess the efficacy of `HALO` in reducing on-chip memory usage, we measure the number of accesses to input



(a) Normalized energy (b) Normalized EDP

Figure 12: Normalized energy and EDP.

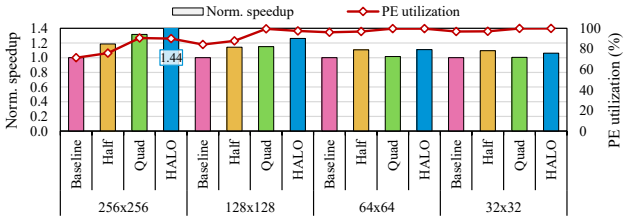


Figure 13: Normalized speedup and PE utilization of the proposed schemes varying systolic array configurations.

and weight SRAM buffers. Figure 11 shows the normalized accesses to both on-chip buffers. In Half Mode, weight buffer accesses are reduced by half (15.1 PP) compared to the Baseline through weight reuse, while input buffer accesses remain the same as in the Baseline. In contrast, Quad Mode does not reduce the weight buffer accesses but significantly decreases the input buffer accesses by 16.0 PP. The reduction in input buffer accesses stems from finer-grained tile mapping via smaller logical sub-arrays, which mitigates redundant accesses caused by coarse-grained tiling. Finally, HALO reduces both input and weight buffer accesses by 16.0 PP and 13.1 PP, respectively. Both benefits are achieved through the hybrid execution strategy, which selects the optimal mode for each CGEMM layer, balancing operand reuse and PE resource utilization.

5.4. Energy Efficiency

To evaluate the energy efficiency of the proposed architecture, we measure the energy consumption and the Energy-Delay Product (EDP). Figure 12 presents both results, normalized to the Baseline and averaged across all evaluated CVNNs. As shown in Figure 12a, Half Mode reduces energy consumption by 6.1%, primarily due to its ability to reuse weight matrices across two computation phases, thereby halving accesses to the power-hungry SRAM buffer. Quad Mode achieves a slightly higher reduction of 7.6%, which is attributed to improved utilization of PE resources and fewer accesses to the input SRAM buffer. By leveraging the benefits of both Half Mode and Quad Mode depending on the matrix dimensions of each layer, HALO achieves the best energy efficiency, decreasing energy consumption by 10.0% compared to the Baseline.

A similar trend is observed in the normalized EDP results, as shown in Figure 12b. Half Mode and Quad Mode reduce EDP by 19.3% and 24.3%, respectively, compared to the Baseline. The EDP reduction is even more pronounced in HALO, achieving an average reduction of 32.3% compared to the Baseline. In summary, HALO demonstrates the highest energy efficiency among all configurations, validating its performance as well as architectural effectiveness.

5.5. Hardware Overhead

To support the logical partitioning-based CGEMM computations, HALO integrates a Negation Logic that selectively flips the sign bit of intermediate products during accumulation. Although lightweight in design, this logic introduces hardware overhead with additional XOR gates for negation. Therefore, we evaluate the area and power consumption of the logic, assuming that HALO is integrated into TPUv4i [27] which contains four 128×128 pods. While the baseline includes a 256×256 pod similar to the latest TPU (TPUv6e [18]), its die size is not publicly disclosed. As such, we instead adopt TPUv4i for area and power measurements, as it is a representative systolic array architecture with publicly reported specifications.

Based on the synthesis results, the Negation Logic occupies $0.22mm^2$, accounting for just 0.05% of the $400mm^2$ TPUv4i die. For the power measurements, we scale the power consumption of a single XOR gate by the total number of gates across the arrays, amounting to 512 gates for four 128×128 pods. Given that a single XOR gate consumes 2.52μW [17], the total power consumption of the added logic is estimated at 1290μW. Compared to the 75W power budget of TPUv4i, this accounts for 2×10^{-6}% of the total chip power. In conclusion, the overhead from the added logic is negligible—especially when weighed against its substantial gains in performance and energy efficiency.

5.6. Sensitivity Study

Thus far, we have evaluated the proposed architecture using a single 256×256 systolic array. We now study how different systolic array configurations affect the PE utilization and performance of HALO. Figure 13 reports the speedup and PE utilization results for four configurations: 256×256 (1 pod), 128×128 (4 pods), 64×64 (16 pods), and 32×32 (64 pods). All results are averaged over a set of CVNNs, and speedup values are normalized to the Baseline.

Under the modest configuration with four 128×128 arrays, HALO shows performance improvement of 26.2% over the Baseline, though this gain is slightly lower than that achieved with a single 256×256 array. This is because the Baseline system with 128×128 arrays already achieves relatively high PE utilization (12.9 PP higher than with 256×256 arrays), thus leaving less room for improvement. Scaling down to 64×64 and 32×32 shows diminishing gains, as these configurations inherently mitigate underutilization, with Baseline already achieving 96% utilization. As a result, HALO achieves more modest speedups of 11.0% and 6.1% over each corresponding Baseline on the 64×64 and 32×32 configurations, respectively.

Meanwhile, we find that in these scale-out designs, Half Mode tends to outperform Quad Mode; while Half Mode consistently provides around 10.0% speedup, Quad Mode offers less than 2% improvement in these configurations.

The rationale is that, in scale-out designs with minimal underutilization, reusing weights in `Half Mode` offers more substantial benefits by mitigating on-chip memory pressure. In summary, `HALO` demonstrates its effectiveness in both scale-up and scale-out architectures, albeit with smaller gains in scale-out designs.

6. Related Work

Architecting CGEMM accelerators. Several prior studies have suggested accelerator designs to improve the computational efficiency of CGEMM. Ha *et al.* [20] introduced a Matrix Multiplication Unit (MXU) architecture supporting high-precision complex-valued matrix operations using low-precision MXUs, enhancing performance with modest extensions and improved power efficiency. Ahmad *et al.* [3] designed an FPGA-based accelerator tailored for CVNNs, particularly for image classification tasks using polar-form complex representations. Their design includes detailed implementations of complex-valued arithmetic units such as complex adders and multipliers. Gan *et al.* [16] presented an algorithm-hardware co-design for CVNNs in wireless communication, featuring a dedicated engine with configurable complex convolution and ReLU units. While these studies achieve efficient CGEMM acceleration primarily through *adder tree*-based architectures and dedicated hardware components, this work introduces a *systolic array*-based CGEMM accelerator specifically designed for CVNNs. Lee *et al.* [36] also developed a CVNN accelerator using systolic arrays, where rectangular-form complex numbers are computed through four separate arrays. While their focus lies in optimizing scheduling across multiple arrays, this work explores executing CGEMM within a single systolic array through a novel logical partitioning approach.

Optimizing GEMM execution on systolic arrays. Prior work has explored various architectural strategies to enhance the efficiency of GEMM execution on systolic arrays. Xu *et al.* [64] explored dataflow and buffering strategies to improve GEMM execution in compact CNNs. They proposed a dataflow scheme to exploit data reuse and designed a flexible on-chip buffer that adapts to array size scaling while minimizing memory traffic. Nayan *et al.* [42] implemented a novel in-array data orchestration mechanism that feeds data along the diagonal and enables bi-directional propagation, reducing prefill latency and thus improving performance. Qin *et al.* [45] presented a flexible GEMM accelerator architecture that addresses the inefficiencies of systolic arrays when processing irregular and unstructured sparse matrices in DNN training. Their design introduces a reconfigurable interconnect and tree-based reduction to boost compute utilization and reduce data movement under varying sparsity. Xu *et al.* [63] addressed low PE utilization in systolic arrays for small or depthwise convolutions by enabling multi-directional dataflows. While these studies focus on optimizing GEMM execution on systolic arrays, our work aims to accelerate CGEMM using a hybrid design that maximizes PE utilization.

7. Conclusion

Computing CGEMM operations on systolic arrays poses two key challenges: redundant data fetches and underutilization of PE resources. To overcome these limitations, this paper proposes `HALO`, a hybrid systolic array architecture that exploits logical partitioning to execute multiple CGEMM sub-operations concurrently by adjusting matrix layout. We present a simple yet effective mode selection algorithm to determine the optimal execution mode per layer and incorporate a lightweight negation logic to enable seamless accumulation of partial products across logical partitions. `HALO` switches between `Half Mode` and `Quad Mode` based on the configuration of each layer, thereby achieving optimal performance across diverse CGEMM layers in CVNNs. Compared to the baseline four-phase CGEMM execution on conventional systolic arrays, `HALO` delivers a 44.3% speedup with 10.0% and 32.3% reductions in energy consumption and EDP, respectively.

Acknowledgements

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-00553645, NRF-2022R1C1C1011021, and NRF-2021R1C1C1012172). Yunho Oh and Myung Kuk Yoon are co-corresponding authors.

References

[1] A. Abdelfattah, S. Tomov, and J. Dongarra, "Towards half-precision computation for complex matrices: A case study for mixed precision solvers on gpus," in *2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, 2019, pp. 17–24.

[2] A. J. Abdelmaksoud, S. Agwa, and T. Prodromakis, "Dip: A scalable, energy-efficient systolic array for matrix multiplication acceleration," *arXiv preprint arXiv:2412.09709*, 2024.

[3] M. Ahmad, L. Zhang, and M. E. Chowdhury, "Fpga implementation of complex-valued neural network for polar-represented image classification," *Sensors*, vol. 24, no. 3, p. 897, 2024.

[4] A. Anderson, A. Vasudevan, C. Keane, and D. Gregg, "High-performance low-memory lowering: Gemm-based algorithms for dnn convolution," in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 99–106.

[5] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *International conference on machine learning*. PMLR, 2016, pp. 1120–1128.

[6] J. A. Barrachina, C. Ren, G. Vieillard, C. Morisseau, and J.-P. Ovarlez, "About the equivalence between complex-valued and real-valued fully connected neural networks - application to polinsar images," in *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, 2021, pp. 1–6.

[7] J. A. Barrachina, C. Ren, C. Morisseau, G. Vieillard, and J.-P. Ovarlez, "Complex-valued vs. real-valued neural networks for classification perspectives: An example on non-circular data," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 2990–2994.

[8] S. Barrachina, M. F. Dolz, P. San Juan, and E. S. Quintana-Ortí, "Efficient and portable gemm-based convolution operators for deep neural network training on multicore processors," *Journal of Parallel and Distributed Computing*, vol. 167, pp. 240–254, 2022.

[9] J. Bassey, L. Qian, and X. Li, "A survey of complex-valued neural networks," 2021. [Online]. Available: https://arxiv.org/abs/2101.12249

[10] L. Böttcher and M. A. Porter, "Complex networks with complex weights," *Physical Review E*, vol. 109, no. 2, p. 024314, 2024.

[11] S. Chatterjee, P. Tummala, O. Speck, and A. Nürnberger, "Complex network for complex problems: A comparative study of cnn and complex-valued cnn," in *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)*, vol. Five, 2022, pp. 1–5.

[12] J. Choi, Y. Ha, J. Lee, S. Lee, J. Lee, H. Jang, and Y. Kim, "Enabling fine-grained spatial multitasking on systolic-array npus using dataflow mirroring," *IEEE Transactions on Computers*, vol. 72, no. 12, pp. 3383–3398, 2023.

[13] S. Choi, S. Park, J. Park, J. Kim, G. Koo, S. Hong, M. K. Yoon, and Y. Oh, "Savector: Vectored systolic arrays," *IEEE Access*, 2024.

[14] E. K. Cole, J. Y. Cheng, J. M. Pauly, and S. S. Vasanawala, "Analysis of deep complex-valued convolutional neural networks for mri reconstruction," 2020. [Online]. Available: https://arxiv.org/abs/2004.01738

[15] J. Feng, M. Wen, X. Ju, J. Shen, and Y. Guo, "Enhancing the pe utilization for multi-precision systolic array via optimizing computation latency," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2024, pp. 1–5.

[16] J. Gan, Q. Li, H. Shao, Z. Wen, T. Yang, Y. Pan, and G. Sun, "A zynq-based platform with conditional-reconfigurable complex-valued neural network for specific emitter identification," *IEEE Transactions on Instrumentation and Measurement*, 2024.

[17] A. Garg, D. Agrawal, S. Rajput, S. Singhal, and A. Mehra, "A novel power efficient xor gate based on single inverted input," in *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2018, pp. 1179–1182.

[18] Google Cloud, "Tpu v6e," 2025, [Online]. Accessed: 2025-08-30. [Online]. Available: https://cloud.google.com/tpu/docs/v6e

[19] C. Guo, Y. Zhou, J. Leng, Y. Zhu, Z. Du, Q. Chen, C. Li, B. Yao, and M. Guo, "Balancing efficiency and flexibility for dnn acceleration via temporal gpu-systolic array integration," 2020. [Online]. Available: https://arxiv.org/abs/2002.08326

[20] D. Ha, Y. Zhang, C.-C. Kao, C. J. Hughes, W. W. Ro, and H.-W. Tseng, "M3xu: Achieving high-precision and complex matrix multiplication with low-precision mxus," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1–16.

[21] M. M. Hammad, "Comprehensive survey of complex-valued neural networks: Insights into backpropagation and activation functions," 2024. [Online]. Available: https://arxiv.org/abs/2407.19258

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] J. Hennessy and D. Patterson, "A new golden age for computer architecture: domain-specific hardware/software co-design, enhanced," in *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[24] A. Hirose, "Complex-valued neural networks: The merits and their origins," in *2009 International Joint Conference on Neural Networks*, 2009, pp. 1237–1244.

[25] A. Hirose and S. Yoshida, "Generalization characteristics of complex-valued feedforward neural networks in relation to signal coherence," *IEEE Transactions on Neural Networks and learning systems*, vol. 23, no. 4, pp. 541–551, 2012.

[26] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. Patterson, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," 2023. [Online]. Available: https://arxiv.org/abs/2304.01433

[27] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.

[28] N. P. Jouppi, D. H. Yoon, G. Kurian, S. Li, N. Patil, J. Laudon, C. Young, and D. Patterson, "A domain-specific supercomputer for training deep neural networks," *Communications of the ACM*, vol. 63, no. 7, pp. 67–78, 2020.

[29] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Communications of the ACM*, vol. 61, no. 9, pp. 50–59, 2018.

[30] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[31] L. Kljucaric and A. D. George, "Deep learning inferencing with high-performance hardware accelerators," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 4, pp. 1–25, 2023.

[32] T. Kouya, "Acceleration of complex matrix multiplication using arbitrary precision floating-point arithmetic," in *2023 International Conference on Engineering and Emerging Technologies (ICEET)*, 2023, pp. 1–6.

[33] A. Krishnakumar, U. Ogras, R. Marculescu, M. Kishinevsky, and T. Mudge, "Domain-specific architectures: Research problems and promising approaches," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 2, pp. 1–26, 2023.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[35] C. Lee, H. Hasegawa, and S. Gao, "Complex-valued neural networks: A comprehensive survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 8, pp. 1406–1426, 2022.

[36] H. Lee, H. Jang, S. Kim, S. Kim, W. Cho, and W. W. Ro, "Exploiting inherent properties of complex numbers for accelerating complex valued neural networks," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1121–1134.

[37] P. Lin, H. Zhang, L. Li, and Y. Li, "Systolic array architecture for multitasking neural processing unit," in *2024 29th International Conference on Automation and Computing (ICAC)*, 2024, pp. 1–6.

[38] B. Liu, X. Chen, Y. Wang, Y. Han, J. Li, H. Xu, and X. Li, "Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 733–738.

[39] X. Liu, "Neural networks with complex-valued weights have no spurious local minima," in *2025 59th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2025, pp. 1–6.

[40] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 171–178.

[41] H. Michel and A. Awwal, "Enhanced artificial neural networks using complex numbers," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, vol. 1, 1999, pp. 456–461 vol.1.

[42] M. M. R. Nayan, R. Raj, G. B. Shaik, T. Krishna, and A. J. Naeemi, "Axon: A novel systolic array architecture for improved run time and energy efficient gemm and conv operation with on-chip im2col," 2025. [Online]. Available: https://arxiv.org/abs/2501.06043

[43] S. Neuman, B. Plancher, and V. Janapa Reddi, "The magnificent seven challenges and opportunities in domain-specific accelerator design for autonomous systems," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–4.

[44] K. Nguyen, C. Fookes, S. Sridharan, and A. Ross, "Complex-valued iris recognition network," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 182–196, 2023.

[45] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.

[46] T. Raja, "Systolic array data flows for efficient matrix multiplication in deep neural networks," 2024. [Online]. Available: https://arxiv.org/abs/2410.22595

[47] P. S. Rao, G. SaradhiVarma, and R. Mutukuri, "Effective and high computing algorithms for convolution neural networks," *International Journal of Engineering & Technology*, vol. 7, no. 3.31, pp. 66–71, 2018.

[48] M. Rothmann and M. Porrmann, "A survey of domain-specific architectures for reinforcement learning," *IEEE Access*, vol. 10, pp. 13 753–13 767, 2022.

[49] E. Russo, M. Palesi, S. Monteleone, D. Patti, A. Mineo, G. Ascia, and V. Catania, "Dnn model compression for iot domain-specific hardware accelerators," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6650–6662, 2021.

[50] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 58–68.

[51] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[53] J. W. Smith, "Complex-valued neural networks for data-driven signal processing and signal understanding," *arXiv preprint arXiv:2309.07948*, 2023.

[54] H. Sun, J. Shen, C. Zhang, and H. Liu, "A hybrid scale-up and scale-out approach for performance and energy efficiency optimization in systolic array accelerators," *Micromachines*, vol. 16, no. 3, p. 336, 2025.

[55] G. Tiwari, S. Nakhate, A. Pathak, A. Jain, and S. Penurkar, "Hardware accelerators for deep learning applications," in *2025 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*. IEEE, 2025, pp. 1–10.

[56] F. G. Van Zee, "Implementing high-performance complex matrix multiplication via the 1m method," *SIAM Journal on Scientific Computing*, vol. 42, no. 5, pp. C221–C244, 2020.

[57] F. G. Van Zee and T. M. Smith, "Implementing high-performance complex matrix multiplication via the 3m and 4m methods," *ACM Transactions on Mathematical Software (TOMS)*, vol. 44, no. 1, pp. 1–36, 2017.

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[59] B. Wang, S. Ma, G. Zhu, X. Yi, and R. Xu, "A novel systolic array processor with dynamic dataflows," *Integration*, vol. 85, pp. 42–47, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926022000359

[60] S. Wu, Y. Zhai, H. Dai, H. Zhao, Y. Zhu, H. Hu, and Z. Chen, "Turbofno: High-performance fourier neural operator with fused fft-gemm-ifft on gpu," 2025. [Online]. Available: https://arxiv.org/abs/2504.11681

[61] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[62] R. Xu, S. Ma, Y. Guo, and D. Li, "A survey of design and optimization for systolic array-based dnn accelerators," *ACM Comput. Surv.*, vol. 56, no. 1, Aug. 2023. [Online]. Available: https://doi.org/10.1145/3604802

[63] R. Xu, S. Ma, Y. Wang, X. Chen, and Y. Guo, "Configurable multi-directional systolic array architecture for convolutional neural networks," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 4, pp. 1–24, 2021.

[64] R. Xu, S. Ma, Y. Wang, and Y. Guo, "Hesa: Heterogeneous systolic array architecture for compact cnns hardware accelerators," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 657–662.

[65] S. Yadav and K. R. Jerripothula, "Fccns: Fully complex-valued convolutional networks using complex-valued color model and loss function," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 10 655–10 664.

[66] E. Yago, P. Castelló, S. Petit, M. E. Gómez, and J. Sahuquillo, "Impact of the array shape and memory bandwidth on the execution time of cnn systolic arrays," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2020, pp. 510–517.

[67] M. Yang, M. Q. Ma, D. Li, Y.-H. H. Tsai, and R. Salakhutdinov, "Complex transformer: A framework for modeling complex-valued sequence," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 4232–4236.

[68] Q. Yang and H. Li, "Bitsystolic: A 26.7 tops/w 2b 8b npu with configurable data flows for edge devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1134–1145, 2021.

[69] A. C. Yüzügüler, C. Sönmez, M. Drumond, Y. Oh, B. Falsafi, and P. Frossard, "Scale-out systolic arrays," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 2, pp. 1–25, 2023.

[70] H. Zhang, M. Gu, X. Jiang, J. Thompson, H. Cai, S. Paesani, R. Santagati, A. Laing, Y. Zhang, M.-H. Yung *et al.*, "An optical neural chip for implementing complex-valued neural network," *Nature communications*, vol. 12, no. 1, p. 457, 2021.