# WinAcc: Window-based Acceleration of Neural Networks Using Block Floating Point

Xin Ju, Jun He, Mei Wen✉, Jing Feng, Yasong Cao, Junzhong Shen, Zhaoyun Chen, and Yang Shi
*School of Computer, National University of Defense Technology*
*Key Laboratory of Advanced Microprocessor Chips and Systems*
Changsha, 410073, China
{jx, hejun19, meiwen, fengjing22, caoyasong, shenjunzhong, chenzhaoyun, shiyang14}@nudt.edu.cn

***Abstract*—Deep Neural Networks (DNNs) impose significant computational demands, necessitating optimizations for computational and energy efficiencies. Per-vector scaling, which applies a scaling factor to blocks of elements using narrow integer types, effectively reduces storage and computational overhead. However, the frequent occurrence of floating-point accumulations between vectors limits further improvements in energy efficiency. State-of-the-art accelerators address this challenge by grouping and summing vector products based on their exponent differences, thereby reducing the overhead associated with intra-group shifting and accumulation. Nevertheless, this approach increases the complexity of register usage and grouping logic, leading to limited energy benefits and hardware efficiency.**

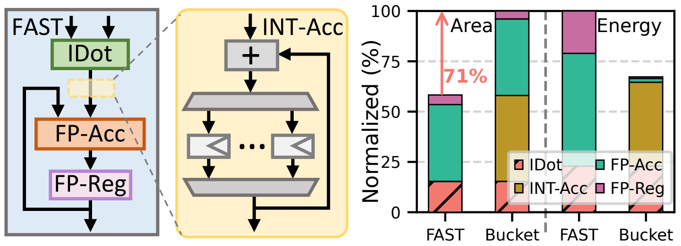
**In this context, we introduce *WinAcc*, a novel algorithm and architecture co-designed solution that utilizes a low-cost accumulator to handle the majority of data in DNNs, offering low area overhead and high energy efficiency gains. Our key insight is that the data of DNNs follows a Laplace-like distribution, which enables the use of a customized data format with a narrow dynamic range to encode most of the data. This allows for the design of a low-cost accumulator with narrow shifters and adders, significantly reducing reliance on floating-point accumulator and consequently improving energy efficiency. Compared with state-of-the-art architecture Bucket, *WinAcc* achieves 33.95% energy reduction across seven representative DNNs and reduces area by 9.5% while maintaining superior model performance.**

***Index Terms*—Floating-point accumulation, block floating point, data distribution, Deep Neural Networks**

## I. Introduction

Deep Neural Networks (DNNs) have achieved remarkable advancements, but these improvements come with significant storage and computational demands. To address these challenges, various model compression techniques have been proposed [1], [2], with per-vector scaling [3] standing out as a promising solution. Per-vector scaling converts blocks of conventional floating-point (FP) data into sign-magnitude integers with a scaling factor, following the block floating-point (BFP) format. This conversion replaces power-intensive FP operations, including shift alignment and normalization, with more efficient integer (INT) operations, while also reducing storage requirements. As a result, per-vector scaling has gained considerable attention in both academia [4], [5] and industry [6]–[8] for its potential to optimize performance while minimizing resource usage.

Several works have introduced customized data formats based on BFP [6], [9], [10]. For instance, [9] quantizes each vector as a superposition of multiple subword vectors, each



(a) Bucket: FAST + INT-Acc (b) Area and energy

Fig. 1: BFP-PEs Architecture and Resource Analysis.

with its own scaling factor. [10] offers configurable bit-widths for scaling factors and mantissas, while [6] uses a fixed 8-bit scaling factor with statically configurable mantissa sizes of 2, 3, or 4 bits. Although all of these approaches convert FP operations into INT operations within blocks, [6] is the simplest and most efficient. However, while INT operations are used within blocks, inter-block accumulation still relies on FP accumulator (FP-Acc) [4], [9], limiting potential reductions in energy and area overhead, e.g., FAST [4]. Bucket [5] introduces an INT accumulator (INT-Acc) before FP-Acc, grouping operands based on their exponents and combining those with close exponents. This approach reduces the overhead associated with shift alignment and addition within groups. Nevertheless, it incurs increased hardware area due to the added complexity of grouping logic and register requirements. Additionally, FP-Acc remains necessary for inter-group accumulation, which limits the overall efficiency improvements, as illustrated in Fig. 1. Consequently, the expensive FP-Acc remains a bottleneck in the energy and area efficiencies of state-of-the-art architectures.

To address these challenges, we propose a low-cost accumulator (LC-Acc) that employs simplified shifters and adders with a narrow accumulation range, eliminating the need for additional grouping logic and registers. Positioned before FP-Acc, LC-Acc handles a significant portion of data, reducing the reliance on FP-Acc. While this design decreases hardware area, it increases the activation frequency of FP-Acc, leading to higher energy overhead compared to Bucket. To mitigate this, we leverage the observation that DNN data follows a Laplace-like distribution [11], and introduce an AutoWin algorithm, which dynamically selects a window to capture the majority of data across tensors. We then design a customized data format, WinFloat, to efficiently encode data both inside and outside the window. Additionally, we develop a hardware-efficient WinFloat decoder and enhance the LC-Acc to handle data from windows of varying sizes and positions. This not only significantly reduces overall power consumption but also enables the FP-Acc to be shared across multiple processing elements (PEs), thereby alleviating area overhead.

Our key contributions are as follows:

- We propose a window search algorithm, AutoWin, that efficiently identifies a minimal window based on tensor distribution to capture the maximum amount of data.
- We introduce a window-aware floating-point data format, WinFloat, where both the exponent bit-width and bias are configurable based on the window's characteristics.
- We design a window-based architecture, *WinAcc*, that integrates a configurable LC-Acc for efficiently processing data within the window, and a shared FP-Acc for handling outliers to significantly reduce the energy and area overhead.
- *WinAcc* achieves a 68.87% and 33.95% reduction in energy, and a 22.9% and 9.5% reduction in area, compared to the state-of-the-art accelerators FAST and Bucket, respectively.

## II. Background and Motivation

### A. Per-vector Scaling

The IEEE-754 standard single-precision floating-point (FP32) format [12], as shown in Fig. 2(a), consists of a 1-bit sign, an 8-bit exponent, and a 23-bit mantissa. This format offers a wide dynamic range ($2^8$) and high precision ($\frac{1}{2^{23}}$), but at the cost of significant computational overhead, e.g., 256-bit shifters and 279-bit adders [13]. Per-vector scaling addresses this issue by quantizing FP vectors into integers with shared scaling factors, thereby amortizing the storage of exponents across multiple mantissas and enabling INT operations within vectors. MSFP [6] is a notable example of this approach, using an 8-bit shared scaling factor that is a power of two, which simplifies hardware design. Fig. 2(b) illustrates the MSFP-12 data format, where the mantissa is represented as a 2-bit two's complement value.

Matrix multiplication is the primary computational pattern in DNNs [14], as described in Eqs. (1–3). In this process, $\mathbf{a}_{m\frac{k}{g}}, \mathbf{w}_{\frac{k}{g}n}, c_{mn}$ represent elements of the activation matrix $A_{M\times K}$, the weight matrix $W_{K\times N}$, and the output matrix $C_{M\times N}$, respectively. These matrices are grouped by common dimensions according to the grouping factor $g$, and $p_{m\frac{k}{g}n}$ represents the element-wise product. To maintain model accuracy, the grouping factor must remain relatively small, typically set to 16 [4], [7], [9]. The FP-Acc is required for inter-group accumulation (Eq. 3), while intra-group operations consist of simple INT dot products (Eq. 2). However, the frequent inter-group accumulations using FP32, occurring $\frac{MN(K-1)}{g}$ times, result in significant energy overhead, making it a critical issue that must be addressed.

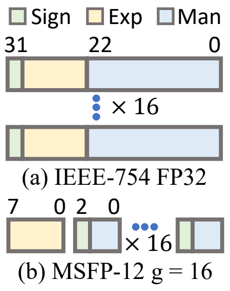


Fig. 2: Data formats.

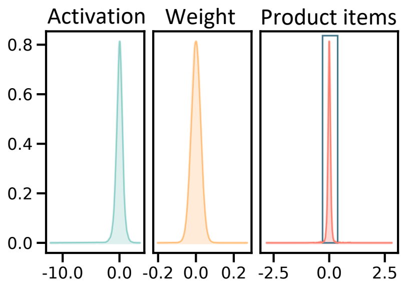


Fig. 3: Data distribution in DNNs.

$$C = A \times W \tag{1}$$

$$p_{m\frac{k}{g}n} = \mathbf{a}_{m\frac{k}{g}} \times \mathbf{w}_{\frac{k}{g}n} \tag{2}$$

$$c_{mn} = \sum p_{m\frac{k}{g}n} \tag{3}$$

### B. Motivation

Several works have shown that the weight and activation matrices in DNNs follow a Laplace-like distribution [11], [15], [16]. In this work, we specifically focus on the characteristics of the element-wise product, which also exhibits a similar Laplace-like distribution with a wide dynamic range and long-tail outliers, as depicted in Fig. 3. By leveraging the unique properties of the element-wise product, we identify a new opportunity to optimize accumulator architectures for improved energy efficiency.

However, previous works have primarily focused on hardware-level optimizations, there remains a need for a more efficient architecture that better aligns with the underlying data distribution. In response to this demand, we propose *WinAcc*, which employs an LC-Acc with a reduced dynamic range to process the majority of densely distributed values, while reserving an FP-Acc for handling long-tail outliers. This approach minimizes the use of the FP-Acc, significantly improving energy efficiency. Furthermore, the FP-Acc is shared across multiple PEs, reducing area overhead. To the best of our knowledge, this is the first work to leverage data distribution characteristics in BFP-based architectures, which advances the state-of-the-art BFP-based accelerators by significantly improving both energy and area efficiencies.

## III. Window-based data representation

### A. AutoWin Algorithm

In Section II-B, we observe that the element-wise product in BFPs tends to concentrate within a major portion of the value range. This concentration allows us to use a low-cost accumulator with a narrow accumulation range to efficiently handle these values. The key challenge is identifying a window that can optimally capture this range. To address this, we propose the AutoWin algorithm, which determines the optimal window size and position, defining the exponent bit-width and bias for encoding. The input to AutoWin includes a weight matrix $W$, an activation matrix $A$, and a threshold $T$, as described in Algo. 1. The process involves three main steps: (1) Compute the exponent vector based on the element-wise product, $PExp$. (2) Select window: initialize the window parameters, $ExpW$ and $Bias$. The window interval is calculated as $[1-Bias, 2^{ExpW}-2-Bias]$. Then shift the window left and right to identify the position that covers the maximum amount of data. (3) Return exponent bit-width $ExpW$ and bias $Bias$ if the proportion of elements in the window is greater than the given threshold $T$, otherwise increase $ExpW$. Note that we only execute the AuToWin algorithm once per tensor, because the distribution of tensors remains roughly similar, which has been exploited by many previous works [11], [17], [18].

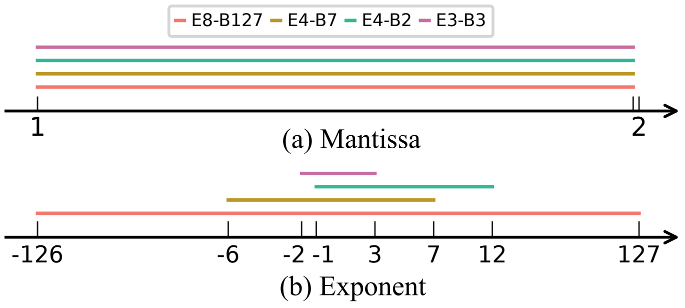
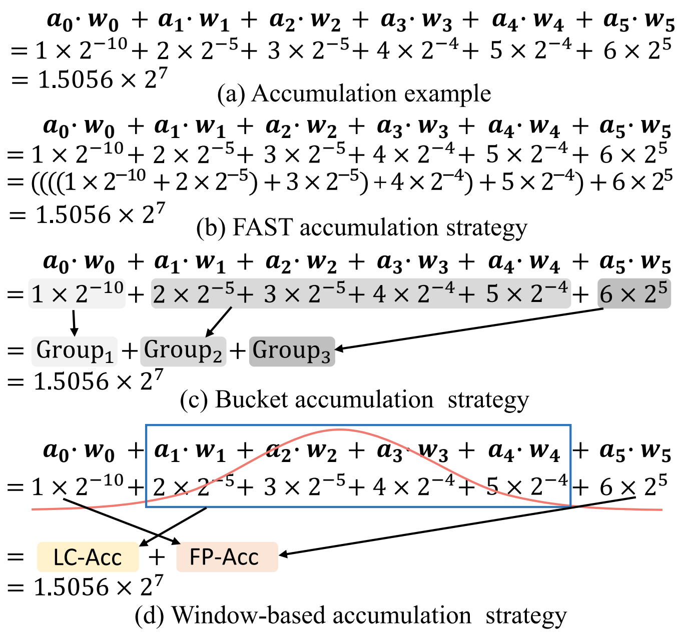


Fig. 4: Different WinFloat configuration.



$a_0 \cdot w_0 + a_1 \cdot w_1 + a_2 \cdot w_2 + a_3 \cdot w_3 + a_4 \cdot w_4 + a_5 \cdot w_5$
$= 1 \times 2^{-10} + 2 \times 2^{-5} + 3 \times 2^{-5} + 4 \times 2^{-4} + 5 \times 2^{-4} + 6 \times 2^{5}$
$= 1.5056 \times 2^{7}$
(a) Accumulation example
$a_0 \cdot w_0 + a_1 \cdot w_1 + a_2 \cdot w_2 + a_3 \cdot w_3 + a_4 \cdot w_4 + a_5 \cdot w_5$
$= 1 \times 2^{-10} + 2 \times 2^{-5} + 3 \times 2^{-5} + 4 \times 2^{-4} + 5 \times 2^{-4} + 6 \times 2^{5}$
$= ((((1 \times 2^{-10} + 2 \times 2^{-5}) + 3 \times 2^{-5}) + 4 \times 2^{-4}) + 5 \times 2^{-4}) + 6 \times 2^{5}$
$= 1.5056 \times 2^{7}$
(b) FAST accumulation strategy
$a_0 \cdot w_0 + a_1 \cdot w_1 + a_2 \cdot w_2 + a_3 \cdot w_3 + a_4 \cdot w_4 + a_5 \cdot w_5$
$= 1 \times 2^{-10} + 2 \times 2^{-5} + 3 \times 2^{-5} + 4 \times 2^{-4} + 5 \times 2^{-4} + 6 \times 2^{5}$
$= \text{Group}_1 + \text{Group}_2 + \text{Group}_3$
$= 1.5056 \times 2^{7}$
(c) Bucket accumulation strategy
$a_0 \cdot w_0 + a_1 \cdot w_1 + a_2 \cdot w_2 + a_3 \cdot w_3 + a_4 \cdot w_4 + a_5 \cdot w_5$
$= 1 \times 2^{-10} + 2 \times 2^{-5} + 3 \times 2^{-5} + 4 \times 2^{-4} + 5 \times 2^{-4} + 6 \times 2^{5}$
= LC-Acc + FP-Acc
$= 1.5056 \times 2^{7}$
(d) Window-based accumulation strategy

Fig. 5: Accumulate products using different strategies.

**Algorithm 1:** AutoWin algorithm

```
input  : Weight: $W$; Activation: $A$; Threshold: $T$
output: Exponent bit-width: $ExpW$; Bias: $Bias$
1  def Window($W$, $A$, $T$):
2      $PExp$ = GetAWEleExp($W$, $A$)
3      $Bias = 2^{(ExpW-1)} - 1 -$ GetMosFreVal($PExp$)
4      for each $ExpW$ in [3, 4] do
5          $WinEle =$ CountWin($PExp, Win$)
6          for each dir in Left, Right do
7              for each Shift in $1 \to 6$ do
8                  if dir == right/Left then
9                      $BiasT = Bias \pm Shift$
10                 $WinEleT =$ CountWin($PExp, Win$)
11                 if $WinEle < WinEleT$ then
12                     $WinEle = WinEleT$
                       $Bias = BiasT$
13         if $WinEle/TenEle > T$ then
14             return $ExpW, Bias$
```

## B. WinFloat Data Format

After obtaining the window, the original values are now split into two parts, to represent values inside and outside different windows, we propose a customized data format, WinFloat. WinFloat is computed as shown in Eq. 4, maintaining the mantissa bit-width of 24, identical to that of FP32, thereby preserving the same level of precision. The exponent bit-width and bias are configurable parameters, as depicted in Fig. 4. In this context, the notation Ex-By refers to an exponent bit-width of $x$ and a bias value of $y$. For example, E8-B127 corresponds to FP32, which provides a wide dynamic range of [-126, 127]. Similarly, E4-B7 and E3-B3 follow the IEEE-754 standard but with narrower dynamic ranges of [-6, 7] and [-2, 3], respectively, while E4-B2 offers a range of [-1, 12]. This configurability allows WinFloat to adjust its data representation range through modifications to exponent bit-width and bias while maintaining full compatibility with FP32 format.

$$Sign \times 2^{Exponent-Bias} \times 1.Mantissa \tag{4}$$

We compare FAST and Bucket accumulation strategies with our proposed window-based approach to highlight its advantages. In Fig. 5, we illustrate the accumulation of six product terms, $\sum_{i=0}^{5} \mathbf{a}_i \mathbf{w}_i$. The FAST method performs element-wise additions, requiring five FP-Accs for accumulation. In contrast, the Bucket strategy categorizes products into three groups based on their exponent, utilizing three registers to store intermediate results and two FP-Accs for combining results between groups. Our window-based approach improves this process by dividing data into two regions: a minimal window, determined by the data distribution, and the remaining data outside the window. Within the window, we employ three LC-Accs and only one register, while data outside the window requires two FP-Accs. Compared to FAST, our approach reduces FP-Acc usage, and relative to Bucket, it requires fewer intermediate registers, thereby increasing energy efficiency.

## IV. *WinAcc* Architecture

### A. Overview

The overall architecture of *WinAcc* is depicted in Fig. 6. It is structured as a two-dimensional spatial array of PE groups, allowing weights and activations to be spatially shared while partial sums (Psum) are accumulated within each PE group. A PE group comprises multiple PEs that share a single FP-Acc, as illustrated on the right side of Fig. 6. Each PE includes an IDot product unit, three WinFloat decoders, an LC-Acc, and a shared FP-Acc.

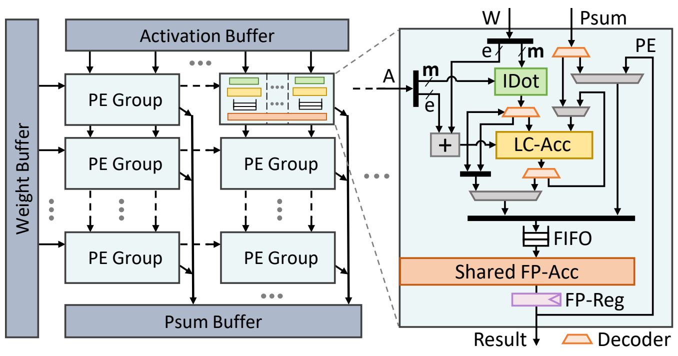
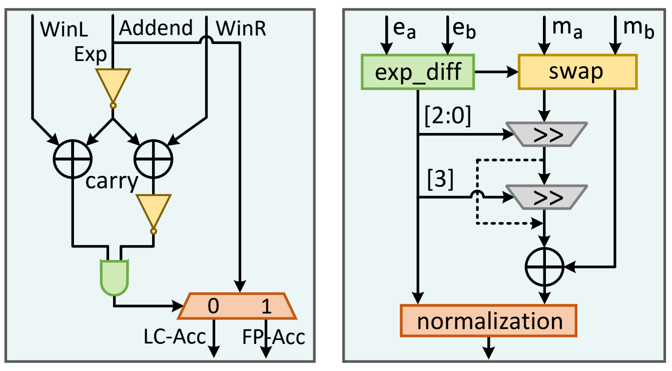


Fig. 6: Overview of *WinAcc* architecture.



(a) WinFloat decoder (b) Low-cost accumulator

Fig. 7: WinAcc components.

The IDot unit is a tree structure designed to perform the dot product of two MSFP mantissa vectors. The three **WinFloat decoders** are responsible for decoding the WinFloat data format for the IDot, LC-Acc, and Psum, respectively. The **LC-Acc** accumulates data inside the window, while the FP-Acc processes data outside the window. Since the proportion of data outside the window is minimal, a single FP-Acc can be shared across multiple PEs, significantly reducing area overhead without performance degradation.

## *B. WinFloat Decoder*

To efficiently decode WinFloat inputs and allocate them between the LC-Acc and FP-Acc, we design a hardware-friendly WinFloat decoder tailored for the AutoWin algorithm. The input to the decoder includes left interval and right interval of the window, denoted as $WinL$ and $WinR$, which are computed by $1 - Bias$ and $2^{ExpW} - 1 - Bias$, respectively, along with the addend. If the exponent of the addend lies within the window, i.e., $WinL \leq Exp < WinR$, the LC-Acc is selected; otherwise, the FP-Acc is used. Thus, the key task is to determine whether $Exp$ falls within the range $[WinL, WinR)$. According to the derivation in Eq. 8, this process is reduced to check whether the sum of $WinL + \widetilde{Exp}$ and $WinR + \widetilde{Exp}$ result in carries of $10_2$, which simplifies the decoder's hardware implementation. The WinFloat decoder requires only basic logic elements such as NOT gates, AND gates, multiplexers, and adders, as illustrated in Fig. 7(a), making it highly efficient and hardware-friendly.

$$WinL \leq Exp < WinR \tag{5}$$
$$WinL - Exp \leq 0 \quad and \quad WinR - Exp > 0 \tag{6}$$
$$WinL + \widetilde{Exp} + 1 \leq 0 \quad and \quad WinR + \widetilde{Exp} + 1 > 0 \tag{7}$$
$$WinL + \widetilde{Exp} \leq -1 \quad and \quad WinR + \widetilde{Exp} \geq 0 \tag{8}$$

## *C. Low-Cost Accumulator*

To accumulate data within different windows (encoded as WinFloat), we design a configurable low-cost accumulator, which retains the core structure of standard FP32 accumulators but utilizes narrow shifters and adders, as shown in Fig. 7(b). To support WinFloat's varying dynamic ranges, the alignment shifter in LC-Acc is designed with two stages, allowing the second stage to be bypassed when the operands are encoded with a smaller dynamic range (as shown by the dotted line in Fig. 7(b)). The accumulation process begins by calculating the exponent difference between operands, followed by adjusting mantissa based on the exponent difference. The smaller mantissa is aligned with the greater one using a shift operation. After alignment, the smaller mantissa is added to the greater one. Finally, the result undergoes normalization to ensure proper precision, generating the final output. This design ensures computational efficiency while maintaining flexibility when handling various configurations of WinFloat.

TABLE I: Evaluated Model and Dataset.

| Type | CNN-based | | | Transformer-based | | |
|---|---|---|---|---|---|---|
| Model | Res34, 50 [19] | VGG16 [20] | Mob-v2 [21] | ViT [22] | BERT [23] | Llama2 [24] |
| Dataset | ImageNet [25] | | | | SQuAD_v1.1 [26] | MMLU [27] |

## V. Evaluations

## *A. Experimental Settings*

*1) **Baselines:*** We evaluate two state-of-the-art designs, FAST [4] and Bucket [5], in comparison to *WinAcc*. For consistency in evaluation, we simplify FAST by excluding variable precision and stochastic rounding techniques. Bucket adapts INT-Acc preceding FP-Acc. To further assess the benefits of *WinAcc*, we also include seven hypothetical designs based on *WinAcc*: WA-3, WA-4, WA-3-NoA, WA-4-NoA, WA-3-NoS, WA-4-NoS, and WA-NoS. The labels "-3" and "-4" indicate that the exponent bit-width of the LC-Acc is fixed at 3 or 4, respectively. "-NoA" refers to designs that do not utilize the AutoWin algorithm, while "-NoS" refers to those that do not share the FP-Acc.

*2) **Hardware Modeling:*** We implement all architectures in Verilog and synthesize them with 12-nm CMOS process at 1GHz clock frequency. Static timing analysis tools are employed to assess module-level power consumption, utilizing the synthesized netlist and activity files with real BERT test patterns. Additionally, we develop a cycle-accurate simulator by extending the PyTorch framework. Specifically, we modify the *Conv2D* and *Linear* operators in the PyTorch to track activation patterns across different computational components. These patterns are then combined with module-level power analysis, enabling us to derive precise energy estimates.

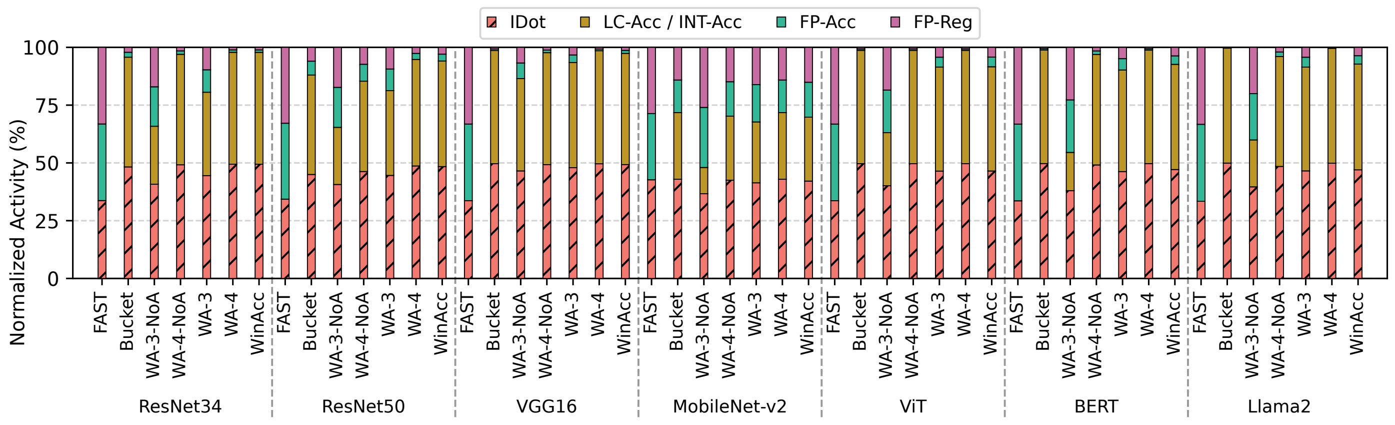


Fig. 8: Activity breakdown on primary compute components.

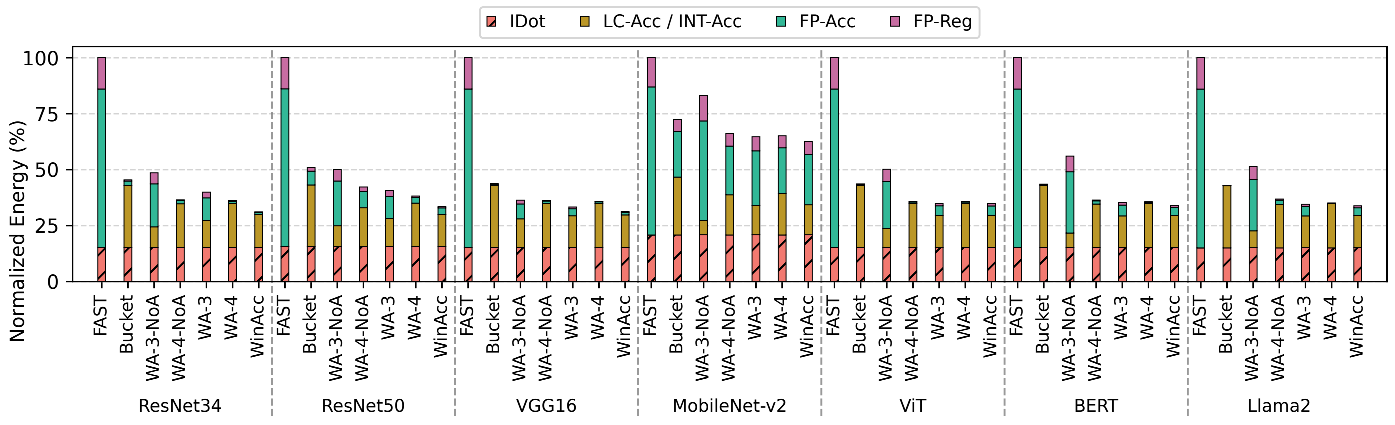


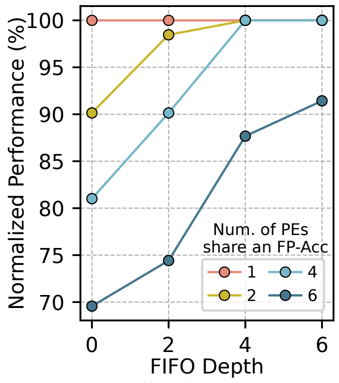
Fig. 9: Energy comparison between different accelerators on seven representative DNNs.

*3) **DNN Models and Datasets:*** Our experiments focus on seven representative DNNs, including ResNet34 (Res34), ResNet50 (Res50) [19], VGG16 [20], MobileNet-v2 (Mob-v2) [21], Vision Transformer (ViT) [22], BERT-Large [23], and Llama2-7B [24]. These models encompass a range of workloads, covering traditional convolutional neural network classifiers (CNN-based), lightweight models, transformer-based classifiers, and transformer-based text generation architectures. The specific applications and datasets associated with each model are detailed in Table I.

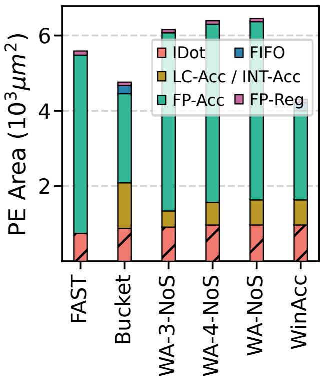
## *B. Experimental Results*

*1) **Module-wise Activity Breakdown:*** We track the activity ratios of key computing components, including IDot, LC-Acc (referred to as INT-Acc in Bucket), FP-Acc, and FP-Reg across FAST, Bucket, *WinAcc*, WA-3-NoA, WA-4-NoA, WA-3, and WA-4. As shown in Fig. 8, *WinAcc* demonstrates significantly lower FP-Acc usage compared to FAST across all networks, as it shifts a major portion of FP-Acc activity to the more efficient LC-Acc. Moreover, *WinAcc* outperforms Bucket in reducing FP-Acc activity on networks with larger dynamic accumulation ranges, such as ResNet34 and ResNet50. For other networks, *WinAcc* maintains minimal FP-Acc usage, largely attributed to the superior performance of the AutoWin algorithm. For instance, with the LC-Acc configured to an exponent bit-width of 3, designs incorporating AutoWin (WA-3) achieve a 45% to 82% reduction in FP-Acc activity compared to those without AutoWin (WA-3-NoA) across all tested networks. Notably, the effectiveness of the AutoWin algorithm improves as the dynamic accumulation range increases. These results emphasize *WinAcc*'s capability to optimize resource utilization and significantly lower energy overhead.

*2) **Energy Comparisons:*** Fig. 9 provides a detailed breakdown of energy across different designs. FAST demonstrates the highest energy overhead, primarily due to its frequent reliance on the FP-Acc. While Bucket reduces FP-Acc usage, the power consumption of INT-Acc remains elevated, leading to overall high energy overhead. *WinAcc* demonstrates the lowest energy across all evaluated designs. Specifically, *WinAcc* achieves energy savings ranging from 37.42% to 68.87% compared to FAST, and from 13.58% to 33.95% compared to Bucket. These reductions are primarily attributed to the synergistic effects of the AutoWin algorithm and the use of the LC-Acc. AutoWin proves especially effective, with designs utilizing it (WA-3, WA-4) showing energy savings of 8.4% to 36.84% compared to those without it (WA-3-NoA, WA-4-NoA). Additionally, the exponent bit-width of the LC-Acc is a key factor in determining energy efficiency. Configurations with a wider exponent bit-width (e.g., WA-4 compared to WA-3) effectively reduce FP-Acc activity, but this reduction comes at



(a) Normalized performance



(b) PE area

Fig. 10: Normalized performance of different FP-Acc share schemes, and PE area with the optimal FP-Acc share scheme.

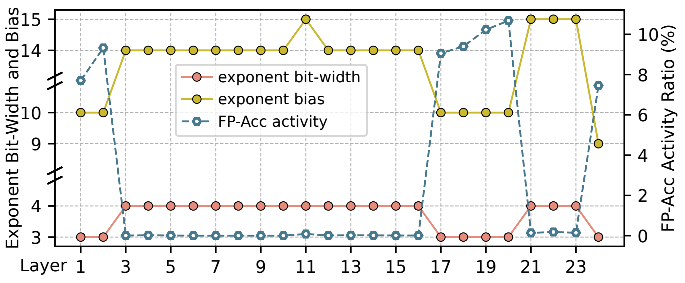


Fig. 11: WinFloat configuration with corresponding FP-Acc activity ratio in BERT.

the cost of higher hardware resource demands. *WinAcc* achieves an optimal trade-off between lowering FP-Acc utilization and managing hardware overhead, underscoring its superior energy efficiency.

*3) **Evaluations on Area:*** When the number of PEs sharing an FP-Acc is two, only minimal queue overhead is incurred, resulting in negligible performance degradation, as shown in Fig. 10(a). Based on this observation, we adopt a two-PE sharing approach in our final design. Fig. 10(b) presents a comparison of the area breakdown across different designs. Compared to FAST, designs that do not share the FP-Acc, such as WA-3-NoS, WA-4-NoS, and WA-NoS, experience an increase in area due to the integration of the LC-Acc. Specifically, the LC-Acc in WA-3-NoS has an exponent bit-width of 3, while WA-4-NoS accommodates a wider exponent range. WA-NoS further enhances flexibility by supporting a configurable exponent bit-width and incorporating multiplexers and bypass logic. These additional features contribute to a 15.5% increase in area for WA-NoS compared to FAST. However, by enabling FP-Acc sharing, area consumption is significantly reduced. As a result, *WinAcc* achieves a total area reduction of 22.9% and 9.5% relative to FAST and Bucket, respectively.

*4) **WinFloat Configuration:*** To evaluate the impact of the AutoWin algorithm on reducing FP-Acc activity, we conduct experiments to measure the layer-wise configuration of WinFloat and the corresponding FP-Acc activity across the models listed in Table I. Due to space limitations, we only present results for BERT in Fig. 11, with other models exhibiting comparable trends. The results indicate that different layers adopt varying WinFloat configurations, optimizing energy by tailoring the data format to the specific computational characteristics of each tensor.

## VI. Related Works

This section presents related works on customized data formats for DNNs, and BFP-based Accelerators.

The customized data formats are generally categorized into FP and BFP formats. For FP, BrainFloat [28] adopts a 1s8e7m format (1 sign bit, 8 exponent bits, and 7 mantissa bits), which preserves the dynamic range of FP32 while reducing mantissa bit-width to minimize multiplier overhead. TensorFloat [29] extends the mantissa to 10 bits, striking a balance between the dynamic range of FP32 and the precision of FP16. ANT [11] offers a flexible encoding scheme that exponent and mantissa bit-width can be adjusted to accommodate different tensors. In the realm of BFP, FlexPoint [10] introduces a “flexN+M” format, where a tensor of M-bit two’s complement integers share an N-bit exponent. MSFP-N, on the other hand, employs a 1-bit sign, an 8-bit exponent, and a group of (N-9)-bit mantissa that shares the maximum exponent within the group, making it highly efficient for hardware implementation.

Several BFP-based architectures have also been developed to accelerate DNNs. FAST [4] incorporates variable precision training with stochastic rounding. VSQ [8] applies quantization at the vector level, using 4 or 8 bits for the tensor’s representation. MX [7] introduces a configurable two-stage scaling factor to enhance model accuracy. Bucket [5] leverages an INT-Acc with a narrow dynamic range to process part of the data. However, these approaches exhibit limitations in terms of energy and area efficiencies.

## VII. Conclusion

In this work, we propose a window-based architecture, *WinAcc*, which features a low-cost accumulator with a narrow range alongside an FP32 accumulator. *WinAcc* is further complemented by a customized data format, allowing the architecture to efficiently handle BFP computations. *WinAcc* achieves both high energy efficiency and low area overhead. The key insight is that the element-wise products in DNNs follow a Laplace-like distribution, allowing the majority of data to be encoded using a narrow dynamic range format, while a shared FP-Acc processes long-tail data. Our evaluation demonstrates that *WinAcc* significantly outperforms similar schemes in energy efficiency and area reduction while maintaining superior model performance.

## Acknowledgment

This work was supported by National Natural Science Foundation of China (Grant No. U23A20301, 62102433), Natural Science Foundation of Hunan Province, China (No. 2023JJ40679), and NUDT Foundation (No. ZK2023-16, 2022-KJWPDL-08). Mei Wen is the corresponding author.

## REFERENCES

[1] T. Choudhary, V. K. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artif. Intell. Rev.*, vol. 53, no. 7, pp. 5113–5155, 2020.

[2] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[3] S. Dai, R. Venkatesan, M. Ren, B. Zimmer, W. J. Dally, and B. Khailany, "Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference," in *Proceedings of the Fourth Conference on Machine Learning and Systems, MLSys 2021.* mlsys.org, 2021.

[4] S. Qian Zhang, B. McDanel, and H. T. Kung, "Fast: Dnn training under variable precision block floating point with stochastic rounding," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 846–860.

[5] Y. Lo and R. Liu, "Bucket getter: A bucket-based processing engine for low-bit block floating point (BFP) dnns," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2023.* ACM, 2023, pp. 1002–1015.

[6] B. D. Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner, A. Forin, H. Zhu, T. Na, P. Patel, S. Che, L. C. Koppaka, X. Song, S. Som, K. Das, S. Tiwary, S. K. Reinhardt, S. Lanka, E. S. Chung, and D. Burger, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

[7] B. D. Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar, L. Shao, G. Kolhe, D. Melts, J. Klar, R. L'Heureux, M. Perry, D. Burger, E. S. Chung, Z. S. Deng, S. Naghshineh, J. Park, and M. Naumov, "With shared microexponents, A little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA 2023.* ACM, 2023, pp. 83:1–83:13.

[8] B. Keller, R. Venkatesan, S. Dai, S. G. Tell, B. Zimmer, C. Sakr, W. J. Dally, C. T. Gray, and B. Khailany, "A 95.6-tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization in 5 nm," *IEEE J. Solid State Circuits*, vol. 58, no. 4, pp. 1129–1141, 2023.

[9] Y. Lo, T. Lee, and R. Liu, "Block and subword-scaling floating-point (BSFP) : An efficient non-uniform quantization for low precision inference," in *The Eleventh International Conference on Learning Representations, ICLR 2023.* OpenReview.net, 2023.

[10] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Hall, L. Hornof, A. Khosrowshahi, C. Kloss, R. J. Pai, and N. Rao, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 2017, pp. 1742–1752.

[11] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "ANT: exploiting adaptive numerical data type for low-bit deep neural network quantization," in *55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022.* IEEE, 2022, pp. 1414–1433.

[12] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[13] Y. Kim, J. Jang, J. Lee, J. Park, J. Kim, B. Kim, B. Park, S. J. Kwon, D. Lee, and J. Kim, "Winning both the accuracy of floating point activation and the simplicity of integer arithmetic," in *The Eleventh International Conference on Learning Representations, ICLR 2023.* OpenReview.net, 2023.

[14] J. Zhuang, J. Lau, H. Ye, Z. Yang, Y. Du, J. Lo, K. Denolf, S. Neuendorffer, A. K. Jones, J. Hu, D. Chen, J. Cong, and P. Zhou, "CHARM: composing heterogeneous accelerators for matrix multiply on versal ACAP architecture," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2023*, P. Ienne and Z. Zhang, Eds. ACM, 2023, pp. 153–164.

[15] F. Liu, N. Yang, H. Li, Z. Wang, Z. Song, S. Pei, and L. Jiang, "SPARK: scalable and precision-aware acceleration of neural networks via efficient encoding," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2024.* IEEE, 2024, pp. 1029–1042.

[16] L. Liu, Z. Xu, Y. He, Y. Wang, H. Li, X. Li, and Y. Han, "Drift: Leveraging distribution-based dynamic precision quantization for efficient deep neural network acceleration," in *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC 2024.* ACM, 2024, pp. 140:1–140:6.

[17] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, J. E. Gonzalez, and I. Stoica, "Ansor: Generating high-performance tensor programs for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020.* USENIX Association, 2020, pp. 863–879.

[18] J. Lin, J. Tang, H. Tang, S. Yang, W. Chen, W. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "AWQ: activation-aware weight quantization for on-device LLM compression and acceleration," in *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024.* mlsys.org, 2024.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016.* IEEE Computer Society, 2016, pp. 770–778.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015*, Y. Bengio and Y. LeCun, Eds., 2015.

[21] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018.* Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4510–4520.

[22] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *9th International Conference on Learning Representations, ICLR 2021.* OpenReview.net, 2021.

[23] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186.

[24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *CoRR*, vol. abs/2307.09288, 2023.

[25] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA.* IEEE Computer Society, 2009, pp. 248–255.

[26] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, J. Su, X. Carreras, and K. Duh, Eds. The Association for Computational Linguistics, 2016, pp. 2383–2392.

[27] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021.* OpenReview.net, 2021.

[28] D. D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of BFLOAT16 for deep learning training," *CoRR*, vol. abs/1905.12322, 2019.

[29] P. Kharya, "Tensorfloat-32 in the a100 gpu accelerates ai training, hpc up to 20x," *NVIDIA Corporation, Tech. Rep*, 2020.