Pushing the Limits of BFP on Narrow Precision LLM Inference

Hui Wang[1*], Yuan Cheng[2,3*†], Xiaomeng Han[1], Zhengpeng Zhao[4], Dawei Yang[2✉], Zhe Jiang[1✉],
[1]National Center of Technology Innovation for EDA, School of Integrated Circuits, Southeast University
[2]Houmo AI
[3]Nanjing University
[4]Huazhong University of Science and Technology
whmio0115@seu.edu.cn, yuancheng@smail.nju.edu.cn, mingzhihan7@gmail.com, u202114911@hust.edu.cn,
dawei.yang@houmo.ai, zhejiang.uk@gmail.com

Abstract

The substantial computational and memory demands of Large Language Models (LLMs) hinder their deployment. Block Floating Point (BFP) has proven effective in accelerating linear operations, a cornerstone of LLM workloads. However, as sequence lengths grow, nonlinear operations, such as Attention, increasingly become performance bottlenecks due to their quadratic computational complexity. These nonlinear operations are predominantly executed using inefficient floating-point formats, which renders the system challenging to optimize software efficiency and hardware overhead. In this paper, we delve into the limitations and potential of applying BFP to nonlinear operations. Given our findings, we introduce a hardware-software co-design framework (DB-Attn), including: (i) DBFP, an advanced BFP version, overcomes nonlinear operation challenges with a pivot-focus strategy for diverse data and an adaptive grouping strategy for flexible exponent sharing. (ii) DH-LUT, a novel lookup table algorithm dedicated to accelerating nonlinear operations with DBFP format. (iii) An RTL-level DBFP-based engine is implemented to support DB-Attn, applicable to FPGA and ASIC. Results show that DB-Attn provides significant performance improvements with negligible accuracy loss, achieving 74% GPU speedup on Softmax of LLaMA and 10x low-overhead performance improvement over SOTA designs.

Introduction

The noteworthy success of Large Language Models (LLMs) has revolutionized various fields of artificial intelligence. The emerging LLMs like LLaMA families (Meta 2024) and Mistral families (Jiang et al. 2023) continue to push the boundaries of what these models can achieve, promising even greater capabilities in natural language understanding, generation, and problem-solving across diverse domains. While LLMs have exhibited remarkable performance across a range of tasks, their inference process demands substantial computing power and memory bandwidth, severely hindering their application and implementation.

Various approaches have been explored for efficient large model deployment. While quantization (Lin et al. 2024; Ma et al. 2024) and pruning (Frantar and Alistarh 2023; Xia et al. 2023) reduce model size and complexity, they often suffer from accuracy degradation and complex post-training processes. Alternative numerical formats like BF16 and TF32 (Burgess et al. 2019; Choquette et al. 2021) improve efficiency but remain costly for large-scale inference. Block Floating-Point (BFP) (Drumond et al. 2018; Darvish Rouhani et al. 2020) offers a promising solution by sharing exponents within data blocks, providing the dynamic range for DNN inference with minimal hardware overhead.

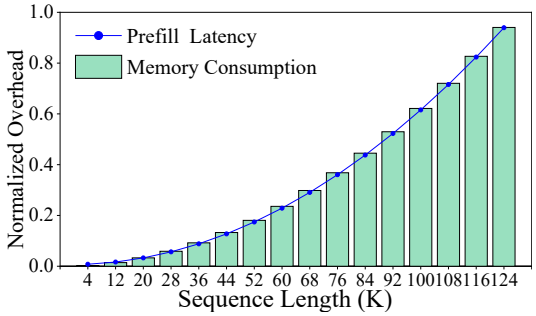


Figure 1: Memory overhead and latency of prefill stages for LLaMA3-8B scale superlinearly with sequence length.

While BFP research in deep learning has primarily targeted linear operations, such as convolution and fully connected layers, nonlinear operations like Softmax and GELU still depend on floating-point computations, emerging as performance bottlenecks. For instance, Softmax alone consumes over 30% of LLM inference time (Stevens et al. 2021). Analysis of LLaMA3-8B shows that longer sequences lead to super-linear growth in memory and latency (Fig. 1), due to the quadratic complexity of Attention layers.

In this paper, we pioneer the application of BFP to nonlinear operations and identify three key challenges: ❶ Outlier Sensitivity: accuracy degradation from exponent alignment with outliers. ❷ Representation Granularity: uniform alignment of exponent drops the representation granularity, which leads to less accurate results. ❸ Hardware Complexity: nonlinear operations involve complex logic (transcendental functions, division, etc.), complicating system optimization (detailed in Tab. 1). To this end, we propose: **Dynamic-BFP (DBFP)**, an advanced variant of BFP,

✉Corresponding authors.
*Equal contribution.
†This work was conducted during his internship at Houmo AI.

adeptly addresses the challenges of accuracy and efficiency in nonlinear operations. It incorporates two novel strategies: a pivot-focus strategy capable of conforming to various data distributions, and an adaptive grouping strategy that enables a more precise and flexible exponent sharing mechanism.

DB-Attn, a DBFP-based framework with software-hardware co-optimization for efficient Attention computation. It accelerates Softmax without floating-point operations and streamlines the dataflow between linear (BFP Matmul) and nonlinear (DBFP Softmax) operations by sharing exponents, eliminating explicit conversions.

- `Algorithm`. We present DH-LUT, a nonlinear operations dedicated lookup table (LUT) algorithm with DBFP format, achieving 74% speedup on the GPU for Softmax while maintaining comparable accuracy to floating-point.
- `Hardware`. We design and implement an RTL-level DBFP-based engine applicable to FPGA and ASIC, delivering 10x throughput over SOTA designs.

Related Work

Data Formats for LLMs

As LLMs grow in size and complexity, the standard 32-bit floating-point (FP32) format becomes less practical. Researchers develop low-bit formats such as BF16 (Jouppi et al. 2020) and TF32 (NVIDIA 2022) to address increasing memory and computational demands. Low-bit fixed-point data types convert operations to integer operations (e.g., INT4) by fixing the number of floating-point bits (Nagel et al. 2019; NVIDIA 2020; Mellempudi et al. 2017) , risking accuracy drop with large dynamic ranges. BFP formats (Drumond et al. 2018) share exponents within data blocks, enabling efficient GEMM operations through dense integer logic with reduced hardware complexity. While (Song, Liu, and Wang 2018) explores various BFP grouping methods, they are limited by spatial constraints and ignore data distribution characteristics. Although BSFP (Lo, Lee, and Liu 2023) improves upon BFP, its complex hardware design presents implementation challenges. The MXFP format (Rouhani et al. 2023) represents another step towards efficient floating-point computations in deep learning.

Nonlinear Operation Algorithms

Nonlinear operations present unique challenges in efficient temporal memory utilization and computation, requiring multiple passes over input values held in memory (Kim et al. 2023). Calculation methods for these operations typically fall into two categories: LUT-based method (Zhang et al. 2023; Du et al. 2019; Zhang et al. 2022) and approximation algorithms (Kim et al. 2021; Xia and Zhang 2023; Wang et al. 2018). While LUTs offer high accuracy, they demand substantial storage. Approximation, on the other hand, generally improves in accuracy with larger computational units.

Methodology

Dynamic Block Floating-Point

Here, we introduce the DBFP, including its mathematical model and the corresponding optimization.

BFP Formulation. Let $\mathbb{F}$ denotes the set of floating-point numbers, and $x \in \mathbb{F}$ is represented as: $x = (-1)^s \cdot 2^e \cdot m$ where $s \in \{0, 1\}$ is the sign bit, $e \in [e_{\text{min}}, e_{\text{max}}]$ is the exponent, and $m \in [1, 2)$ is the mantissa. BFP numbers are formally defined as a group: $Y = (y_1, y_2, \ldots, y_n)$, where each y_i shares an exponent e_s, such that $y_i = (-1)_i^s \cdot m_i \cdot 2^{e_s}$. BFP numbers can be partitioned into two fields: a shared field S and a private field P. Each $\hat{s_j} \in S$ represents a shared exponent for multiple elements in P. Each $p_i \in P$ is of the form $p_i = (-1)_i^s \cdot m_i$.

To convert floating-point numbers to BFP, a uniform exponential alignment is applied as Eq.1:

$$\begin{aligned} x_i = (-1)_i^s \cdot 2^{e_i} \cdot m_i &= (-1)_i^s \cdot 2^{e_i - d_{ij}} \cdot (m_i \cdot 2^{d_{ij}}) \\ &= (-1)_i^s \cdot 2^{\hat{s_j}} \cdot m_i' \end{aligned} \tag{1}$$

, where $d_{ij} = e_i - \hat{s_j}$ denotes the difference between the exponent of x_i and the jth shared exponent. This alignment introduces an error due to the finite precision of mantissa multiplication by $2^{d_{ij}}$ (shifting d_{ij} bits). The error depends on the distance d_{ij}, determined by the input exponent e_i. Therefore, selecting appropriate shared exponents $\hat{s_j}$, which are determined by many factors, is crucial to minimize d_{ij}.

Problem Formulation: To this end, we formulate the problem as finding a adaptive grouping function f that partitions a set X of floating-point values into k subsets based on their characteristics: $f : X \to \chi\{S_1, S_2, ..., S_k\}$ where each subset S_j is determined within a frame of discernment Ω.

Each subset S_j has a unique BFP representation B_j with shared and private fields. We then explore factors influencing BFP format accuracy, which define the function f.

Observations and Insights. Through experimental and theoretical analysis, we explore the limitations of vanilla BFP, providing new insights for improving accuracy.

Observation 1 Pivot-focus policy: Vanilla BFP formats typically align a group of x_i (e.g., every 16 elements) to the maximum exponent within that group, leading to significant accuracy drop. Our experimental findings in Softmax show that setting the alignment direction to the default maximum causes up to 9.6x greater loss than the median one.

Insight 1: This suggests that using maximum values for alignment is suboptimal. Instead, a more representative value (e.g., median) as an alignment pivot better preserves accuracy across the group.

Observation 2 Adaptive grouping strategy: Prior work on BFP relied on fixed bounding boxes, which are particularly vulnerable to outliers. Outliers can cause disproportionate shifts in data exponents and lead to substantial accuracy drop. See Tab. 1 for a more detailed analysis.

Insight 2: If elements with similar magnitude distributions can share exponents within a group, it can reduce the bit shifts (diminish the d_{ij}) for individual numbers.

Fig. 2 illustrates the difference between the mentioned formats: (a) represents floating-point data formats such as FP16 or TF32; (b) shows the vanilla BFP format; (c) highlights our DBFP, which employs automatic grouping based on the intervals in which the exponent falls.

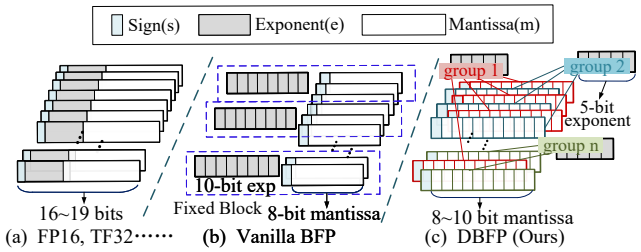


Figure 2: Number system comparison between floating-point numbers(a), BFP(b) and our DBFP(c).

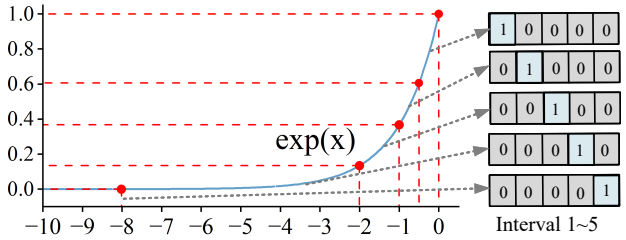


Figure 3: Non-uniform hierarchical LUT with five intervals.

Theoretical Analysis. For the pivot-focus policy, we introduce directional vectors to characterize the process. Let $\vec{s_j}$ denote the vector representing the direction and magnitude from the origin to the point $\hat{s_j}$ in the vector space. The shift vector $v_{ij}^{\rightarrow} = (|\vec{s_j}| - e_i)\hat{v}$ determines the shift for each x_i relative to the sharing exponent. $\vec{m_i'}$ and $\vec{m_i}$ denote the mantissa's projection before and after conversion, with their Euclidean distance as $d_{ij}^2 = \left\|\vec{m_i}' - \vec{m_i}\right\|^2$. μ_{ij} represents the confidence that x_i falls within the interval defined by $\vec{s_j}$. It's aimed to minimize the following objective function:

$$\begin{aligned}\arg\min_{\mu_{ij},\vec{s_j}} \mathcal{J}_{\mathrm{DBFP}} &\triangleq \sum_{i=1}^{n}\sum_{S_j\subseteq\Omega}\mu_{ij}^{\beta}D_{ij}^2 \\ &= \sum_{i=1}^{n}\sum_{\{j/S_j\neq\emptyset,S_j\subseteq\Omega\}}\mu_{ij}^{\beta}d_{ij}^2 + \sum_{i=1}^{n}\mu_{i\emptyset}^{\beta}\delta^2 \\ s.t. \quad \sum_{\{j/S_j\subseteq\Omega S_j\neq\emptyset\}} &\mu_{ij} + \mu_{i\emptyset} = 1, \forall i = 1, n,\end{aligned} \tag{2}$$

In Eq. 2, hyperparameter β (default 2) regulates μ_{ij}'s importance. An empty set mitigates outlier impact on $\vec{s_j}$ selection, with $\mu_{i\emptyset}$ as its confidence. The distance between outliers and $\hat{s_\emptyset}$ is δ^2, related to the distances of all S_j.

$$D_{ij}^2 = \begin{cases}\delta^2, S_j = \emptyset \\ d_{ij}^2, |S_j| = 1\end{cases} \tag{3}$$

To minimize $\mathcal{J}_{\mathrm{DBFP}}$, we alternately fix one of the variables μ_{ij} or $\vec{s_j}$ and solve the constrained minimization problem for the other. By introducing n Lagrange multipliers λ_i, the Lagrangian is expressed as:

$$\mathcal{L}(U, S, \lambda_1, \cdots, \lambda_n) = \mathcal{J}_{\mathrm{DBFP}}(U, S) - \sum_{i=1}^{n}\sum_{S_j\subseteq\Omega}\lambda_i\mu_{ij}^{\beta} \tag{4}$$

By differentiating the Lagrangian and setting each element of gradient $\nabla\mathcal{L}$ to zero, specifically $\frac{\partial\mathcal{L}}{\partial\mu_{ij}}, \frac{\partial\mathcal{L}}{\partial\mu_{i\emptyset}}, \frac{\partial\mathcal{L}}{\partial\lambda_i}$, we obtain μ_{ij}, the necessary conditions for optimality:

$$\mu_{ij} = \frac{d_{ij}^{-2/(\beta-1)}}{\sum_{S_k\neq\emptyset} d_{ik}^{-2/(\beta-1)} + \delta^{-2/(\beta-1)}} \tag{5}$$

Similarly, we can further calculate the $\vec{s_j}$

$$\vec{s_j} = \frac{\sum_{i=1}^{n}\sum_{S_j\neq\emptyset}\mu_{ij}^{\beta}\vec{e_i}}{\sum_{i=1}^{n}\sum_{S_j\neq\emptyset}\mu_{ij}^{\beta}} \tag{6}$$

This system can be solved using a standard linear system solver. Ultimately, this process yields an optimal solution for the set S, i.e., the configuration that complies with the adaptive grouping strategy and pivot-focus policy.

DB-Attn Algorithm Design

In this section, we introduce the algorithmic core of DB-Attn. Unlike the vanilla BFP format, which struggles with nonlinear operations, our proposed DH-LUT, for the first time, completes nonlinear operations in a BFP-like format. We also apply DBFP to linear operations, optimizing Matmul efficiency through cascade operations.

Optimization for Softmax. The Softmax operation is among the most computationally intensive nonlinear components in Transformers, significantly impacting computational efficiency. Our optimization methodologies using DBFP offer a general approach to efficiently computing nonlinear functions, readily extensible to other nonlinear operations. The Softmax function converts Attention scores into a probability distribution. It can be represented as follows:

$$\mathrm{Softmax}\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=0}^{i} e^{x_j}} = \frac{e^{x_i - x_{\max}}}{\sum_{j=0}^{i} e^{x_j - x_{\max}}} \tag{7}$$

, which is transformed by the down-scaling exponentiation to smooth data distribution and prevent overflow.

Softmax's bottleneck stems from the low throughput of exponential functions. Leveraging DBFP's shared exponents, we propose Dynamic Hierarchical LUT (DH-LUT), a lightweight solution. DH-LUT uses a two-dimensional structure of sub-LUTs, dynamically loaded based on DBFP shared exponents and high k bits of mantissa. For Softmax's $e^{x-x_{\max}}$ operation, only the $[-\infty, 0]$ range needs fitting, showing non-uniform distribution. DH-LUT adapts to this through pivot-focus policy and non-uniform fitting. We optimize accuracy and memory with a non-uniform hierarchical allocation method (Algo.1), adaptively partitioning to focus resources on nonlinear regions shown in Fig. 3.

For the Softmax based on DH-LUT, its computational error stems from the DBFP format and the mapping error of DH-LUT. Eq. 8 introduces the error analysis of vanilla BFP, which is applicable to DBFP. For a BFP block, using the

rounding-to-nearest scheme, its quantization error is zero-mean with variance σ^2, defined as:

$$\sigma^2 = \frac{2^{-2L_m}}{12} \sum_{i=1}^{N_\gamma} p_{\gamma_i} 2^{2\gamma_i} \tag{8}$$

, where L_m denote the bit length of the block mantissa, and p_{γ_i} represent the probability mass function (PMF) of the block exponent. $N_\gamma = 2^{L_E}$ is the number of available block exponential levels. Increasing the bit length of the block mantissa and reducing the shared probability of larger exponents can effectively reduce input value errors. While aligning to the maximum exponent (vanilla BFP) causes significant errors as inputs with tiny exponents are crucial for Softmax shown in Fig.3, and minimum exponent alignment preserves accuracy at the cost of redundant mantissa bit, as the table width of DH-LUT is fixed at the Pareto frontiers, detailed in Fig.5, our approach aligns with the median exponent, balancing accuracy and efficiency.

The data stored in DH-LUT is all DBFP, and its characteristic of shared exponents allows calculations not only in the floating-point domain but also on computing resources-limited devices, supporting integer-only computation:

$$\mathrm{Softmax}(x_i) = \frac{\mathrm{e}^{\mathrm{x_i}-\mathrm{x_{max}}}_{\text{DH-LUT}}}{\sum_{\mathrm{j}}^{\mathrm{d}} \mathrm{e}^{\mathrm{x_j}-\mathrm{x_{max}}}_{\text{DH-LUT}}} = \frac{2^{\mathrm{s}} \cdot \mathrm{e}^{\mathrm{int}}_{\mathrm{i}}}{\sum_{\mathrm{j}}^{\mathrm{d}} 2^{\mathrm{s}} \cdot \mathrm{e}^{\mathrm{int}}_{\mathrm{j}}} = \frac{\mathrm{e}^{\mathrm{int}}_{\mathrm{i}}}{\sum_{\mathrm{j}}^{\mathrm{d}} \mathrm{e}^{\mathrm{int}}_{\mathrm{j}}} \tag{9}$$

DB-Attn leverages the DH-LUT with minimal storage overhead to compute the exponential function, enhancing the throughput of nonlinear operations. Its shared exponent feature facilitates the migration of the Softmax algorithm to devices with limited computing resources, achieving significant speedups on both floating-point platforms and edge devices with integer-only capabilities.

Optimization of Matrices. From Eq. 1, a vector in DBFP format can be viewed as the product of a shared coefficient and an integer vector, expressed as: $\overrightarrow{A_D} = 2^{e_A} \cdot \overrightarrow{A_I}$ where $\overrightarrow{A_I}$ represents an integer vector. e_A is the exponent shared by this vector. Applying DBFP to matrices in Attention, the dot product operation of a single vector $\overrightarrow{Q_{\mathrm{D}}}$ and $\overrightarrow{K_{\mathrm{D}}^T}$ within the Matmul of Query and Key can be described as :

$$\begin{aligned} \overrightarrow{Q_{\mathrm{D}}} \cdot \overrightarrow{K_{\mathrm{D}}^T} &= (2^{e_Q} \cdot \overrightarrow{Q_I}) \cdot (2^{e_K} \cdot \overrightarrow{K_I^T}) \\ &= 2^{e_Q + e_K} (\overrightarrow{Q_I} \cdot \overrightarrow{K_I^T}) \end{aligned} \tag{10}$$

It can be discerned that vector dot products and Matmul in the DBFP format can be achieved using integer Matmul units and integer adders, bypassing complex floating-point operations and enhancing computational efficiency. Extending this operation to the entire matrix reveals the trait of cascaded DBFP Matmul. The result of multiplying two DBFP matrices remains a DBFP matrix, capable of being seamlessly chained for subsequent Matmul without additional handling. For DBFP matrices with finer-grained blocks, realignment can be introduced during cascading operations, efficiently handled via hardware shift operations, enabling even more efficient computation.

Algorithm 1: DH-LUT hierarchical algorithm

Input: Target Function F, lut table size m, all possible values under the FP16 format V.
Output: Optimal Partition Points of the LUT OPP .

function UPDATE_NEXT(OPP, i, m, V)
  $interval = (len(x) - OPP[i])//(m - 1 - i)$
  for j = 1 to m - 1 - i **do**
    $OPP[i + j] = OPP[i] + j * interval$
  end for
end function

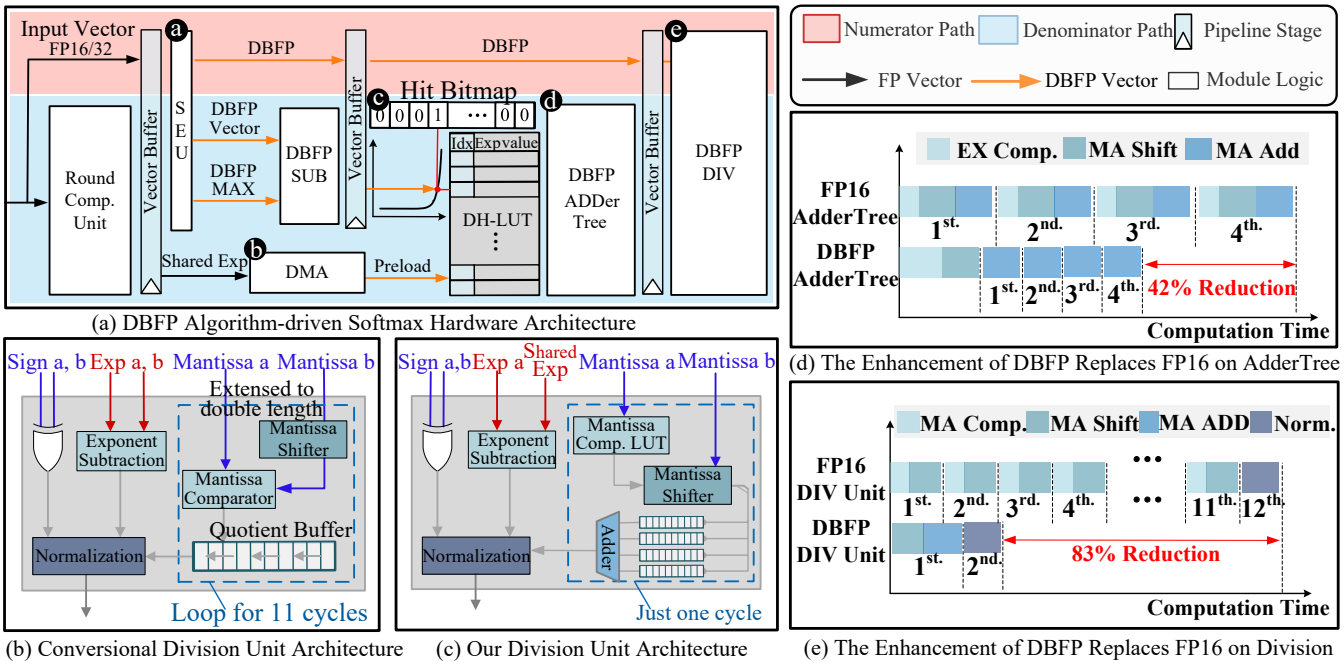
function SELECT_BEST_OPP(F, V, m)
  $interval = len(x)//(m - 1)$
  Initialize OPP with $interval$
  for i = 1 to m **do**
    $D_{max} = \infty$
    $pre = OPP[i - 1]$
    $next = OPP[i + 1]$
    $out_F = F(V)$
    for j = pre + 1 to next **do**
      calculate Interpolation out_{right}
      calculate MAE D_{right}
      calculate Interpolation out_{left}
      calculate MAE D_{left}
      $D = D_{left} + D_{right}$
      Update $OPP[i], D_{max}$ if $D \leq D_{max}$
      UPDATE_NEXT(OPP, i, m, V)
    end for
  end for
  return OPP
end function

Algorithm-driven Hardware Architecture

We design and implement an RTL-level engine for Softmax with DBFP and evaluate the hardware resources and throughput advantage. Benefiting from the DBFP, the proposed accelerator offers competitive performance with very light design complexity, making it easily adaptable to other general-purpose GPUs and NPUs. Below, We'll first detail the proposed architecture and implementation.

Overall Architecture. With the alignment to the algorithms presented above, we split the architecture of the accelerator into four pipeline stages: **`Max`** finds the maximum value within the input vector; **`SE`** performs Shared Exponent calculation and maximum value subtraction; **`Exp`** computes exponents and sum using an adder tree; **`Div`** executes division operations to obtain the result.

The `SE` stage segments and aligns the exponent of the input vector with the `MAX` stage's maximum value, followed by subtraction (Fig. 4 ⓐ). Simultaneously, it checks if the DH-LUT's current data exponent matches the required to-be-shared exponent, signalling DMA (Direct Memory Access, a module allowing hardware subsystems to access memory for efficient data transfer between memory and devices) to preload new data required for the following computations (Fig. 4 ⓑ). These initial stages prepare for computation. The subsequent three phases complete the exponential function (Fig. 4 ⓒ), denominator summation (Fig. 4 ⓓ), and division (Fig. 4 ⓔ) – the main operations in Softmax.



(a) DBFP Algorithm-driven Softmax Hardware Architecture

(d) The Enhancement of DBFP Replaces FP16 on AdderTree

(b) Conversional Division Unit Architecture

(c) Our Division Unit Architecture

(e) The Enhancement of DBFP Replaces FP16 on Division

Figure 4: DB-Attn algorithm-driven Softmax hardware architecture and enhancement compared with FP16 design.

Low-bit Storage Implementation of DBFP. Our dynamic hierarchical non-uniform LUT strategy introduced in the previous section enables a compact yet flexible hardware implementation. By extracting n bits from vector elements, we create a 2^n-entry table that balances accuracy and size. This compact design is suitable for Softmax computation, which requires global input information. Our approach utilizes two tables: a value table storing exp values for approximation and a hit bitmap table recording mantissa occurrences (Fig. 4 ⓒ). Input vectors perform lookups in parallel, with each exponent index hitting a DH-LUT interval, setting a corresponding bitmap bit. After recording the input vector, we multiply and sum values from both tables using an adder tree structure. This achieves parallel lookup results without extra hardware resources, leveraging the compact table size.

Efficient DBFP Computing. DBFP's ability to convert floating-point operations into integer operations is a key advantage. By sharing exponents, most floating-point computations simplify basic exponent operations combined with integer mantissa operations, streamlining arithmetic calculations. Vector addition in neural networks exemplifies this efficiency. Traditional floating-point addition requires aligning exponents for each number pair. DB-Attn pre-aligns exponents within groups, reducing the operation to exponential multiplication with mantissa addition. For cascading structures like adder trees (Fig. 4 ⓓ), a single initial exponent alignment allows direct mantissa calculations, yielding substantial benefits. Fig. 4(d) demonstrates an experimentally proven 42% latency reduction in a 4-level adder tree.

Division operations in Softmax and Layernorm often limit parallelism. DBFP's shared exponents enable efficient parallel integer division approximation (Fig.4 ⓔ), reducing latency of these complex operations. While traditional FP16 dividers require 11 cycles for 10-bit mantissa calculation, consuming 90% of area and power, our DBFP Divider uses LUTs and shift-addition (Jha et al. 2020) to complete division in a single cycle. In Softmax, with fixed divisor and identical DBFP exponents, we need only one exponent subtraction and lookup per 64 divisions, achieving 83% latency reduction (Fig.4(e)). This approach enables FP16 operations using 10-bit integer operations, particularly benefiting bit-width sensitive operations.

Evaluation

Baselines. As described, DB-Attn involves both software and hardware implementation. Hence, we examined DB-Attn against the SOTA work in both worlds. For the software, we compare DB-Attn with FP32, FP16, vanilla BFP, and FP8, focusing on the tradeoffs between accuracy and precision. To measure the improvement DBFP brings to the hardware, we compare its hardware metrics with SOTA Softmax acceleration architectures: Hyft (Xia and Zhang 2023), TCAS-I'22 (Zhang et al. 2023), ISCAS'23 (Koca, Do, and Chang 2023) and TCAS-II'22 (S 2021).

Models and Datasets. We evaluate DB-Attn on both LLM and Vision models. For LLMs, we test on LLaMA-7B, LLaMA2-7B, and LLaMA3-8B (Touvron et al. 2023a,b; Meta 2024) , using WikiText2 and C4 datasets for perplexity comparison. Zero-shot accuracy is evaluated on PIQA, ARC, BoolQ, HellaSwag, and Winogrande tasks. For Vision tasks, we test image classification using ViT and Swin (Dosovitskiy et al. 2020; Liu et al. 2021) on ImageNet, and object detection using Detr on COCO dataset.

Implementations. We implement DB-Attn on NVIDIA

Model	Method	Nonlinear Op	Zero-Shot							Perplexity	
			PIQA(↑)	ARC-e(↑)	ARC-c(↑)	BoolQ(↑)	HellaSwag(↑)	Winogrande(↑)	avg.(↑)	WikiText2(↓)	C4(↓)
LLaMA-7b	FP16	FP32	77.37	52.27	41.21	73.27	72.87	67.32	64.05	5.68	7.08
	BFP	BFP	53.65	30.05	25.34	61.87	32.06	49.41	42.06	67.31	67.13
	FP8 e4m3	FP32	49.51	25.08	22.69	37.82	25.04	49.57	34.95	NaN	NaN
	FP8 e4m3-S	FP32	49.46	25.17	22.78	37.82	25.04	49.57	34.97	NaN	NaN
	FP8 e5m2	FP32	77.31	**51.98**	41.04	72.14	72.24	65.66	63.40	5.80	7.16
	DBFP	**DH-LUT**	**77.64**	51.85	**41.47**	**73.30**	**72.87**	**67.01**	**64.02**	**5.68**	**7.08**
LLaMA2-7b	FP16	FP32	76.88	53.62	40.61	71.07	72.94	67.09	63.70	5.47	6.97
	BFP	BFP	52.72	28.24	25.26	61.90	33.64	49.41	41.86	32.72	41.29
	FP8 e4m3	FP32	49.51	25.08	22.69	37.82	25.04	49.57	34.95	NaN	NaN
	FP8 e4m3-S	FP32	49.56	25.08	22.69	37.82	25.04	49.57	34.96	NaN	NaN
	FP8 e5m2	FP32	**76.98**	52.35	40.69	70.55	72.09	65.43	63.02	5.61	7.09
	DBFP	**DH-LUT**	76.77	**53.20**	**41.47**	**71.01**	**72.60**	**67.32**	**63.73**	**5.48**	**6.98**
LLaMA3-8b	FP16	FP32	80.79	77.74	53.33	81.35	79.17	72.61	74.17	6.14	8.88
	BFP	BFP	54.30	31.06	22.78	49.17	35.67	49.72	40.45	69.65	79.49
	FP8 e4m3	FP32	49.51	25.08	22.70	37.82	25.04	49.56	34.95	NaN	NaN
	FP8 e4m3-S	FP32	49.73	25.17	22.61	37.82	25.04	49.56	34.99	NaN	NaN
	FP8 e5m2	FP32	79.59	78.32	52.55	79.41	78.05	71.50	73.24	6.42	9.24
	DBFP	**DH-LUT**	**80.36**	**78.66**	**53.16**	**81.01**	**78.86**	**73.48**	**74.26**	**6.14**	**8.89**

Table 1: Accuracy performance of DB-Attn in different LLM tasks

Model	Method	Nonlinear Op	Map	Model	Method	Nonlinear Op	Top-1 acc.	Model	Method	Nonlinear Op	Top-1 acc.
DETR	FP32	FP32	41.9	ViT-base	FP32	FP32	84.536	Swin-tiny	FP32	FP32	81.378
	BFP	BFP	26.8		BFP	BFP	39.132		BFP	BFP	72.052
	FP8 e4m3	FP32	NaN		FP8 e4m3	FP32	84.482		FP8 e4m3	FP32	81.312
	FP8 e4m3-S	FP32	NaN		FP8 e4m3-S	FP32	84.442		FP8 e4m3-S	FP32	81.400
	FP8 e5m2	FP32	28.4		FP8 e5m2	FP32	84.246		FP8 e5m2	FP32	81.268
	DBFP	**DH-LUT**	**41.8**		**DBFP**	**DH-LUT**	**84.522**		**DBFP**	**DH-LUT**	**81.384**

Table 2: Accuracy performance of DB-Attn in different Vision tasks

A6000 GPU using Pytorch and huggingface, replacing only the Attention layer with DB-Attn. We use 128-element blocks along matrix rows, with 8-bit mantissa and 5-bit shared exponent for both DBFP and BFP formats. DBFP is applied to Softmax operations and matrix multiplications in Attention Modules, using 7-bit DH-LUT for Softmax.

We implement DBFP in RTL using Chisel , verify with Verilator, and deploy on AMD Alveo™ U280 FPGA using Vivado. We compare our design with other FPGA-based Softmax accelerators. For accurate area and power analysis, we synthesize the accelerator using Synopsys Design Compiler at 2.0 GHz on 28nm TSMC process.

Accuracy Results of DB-Attn

LLM Tasks. We examine the accuracy of DB-Attn on the language generation task and six zero-shot tasks on LLaMA LLMs, comparing it against vanilla BFP and FP8 format. Tab. 1 presents the perplexity and zero-shot accuracy of LLMs. Wherein FP8 e4m3-S denotes the use of scaling factor (the maximum value that FP8 e4m3 can represent) to rescale the value within the representable range of FP8 e4m3. It can be seen in Tab. 1 that the direct application of vanilla BFP format to Softmax operations leads to substantial accuracy drop, with LLaMA3's average accuracy on zero-shot tasks decreasing by 33.81%. Similarly, FP8 e4m3 is inadequate for Attention calculations due to its inability to represent infinity. DB-Attn outperforms both vanilla BFP and FP8 formats across all evaluated tasks, maintaining nearly the same accuracy as floating-point. These results unequivocally demonstrate the comprehensive excellence of DB-Attn in maintaining model performance while potentially reducing computational overhead.

Vision Tasks. We assess tasks of object detection and image classification (Tab. 2). It is seen that DB-Attn's performance is similar to results on LLMs. DB-Attn can be losslessly integrated into existing Transformer models, showing its generalization and versatility across various distributions.

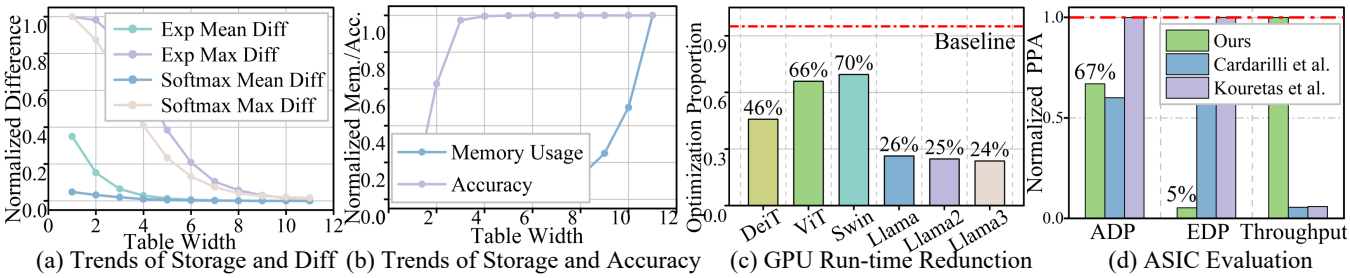
Precision-to-Accuracy Pareto Frontier. We test the computational error, LUT memory usage, and actual model accuracy of Softmax in DB-Attn under different LUT bit widths. To find the Pareto frontier of the LUT bit width configuration, we visualize some results in Fig. 5(a) and (b) – when the LUT bit width is 5-7, a balance is achieved among computational error, memory usage, and accuracy.

Hardware Implement Evaluation

As BFP has been validated on linear operations in previous work (Zou et al. 2024; Yeh et al. 2022) we mainly focus on the performance of hardware deployments for the yet to be well optimized nonlinear operation Softmax.

DBFP GPU Run-time Analysis. We implement a custom CUDA Softmax operator to emulate DBFP formats on NVIDIA A800 GPU. This operator replaces the Softmax in various models (e.g., LLaMA and ViT) and performs inference. Results in Fig. 5(c) demonstrate that DBFP-based Softmax consistently achieves speed improvements of at least 30% across diverse model architectures. Notably, on the LLaMA series, we reduced latency by 74% on average.

DBFP Hardware Implement on FPGA. We evaluate designs against SOTA based on Softmax processing latency,

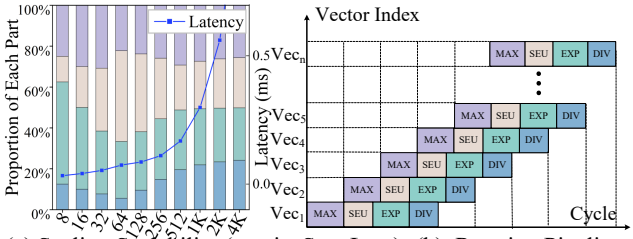


(a) Trends of Storage and Diff (b) Trends of Storage and Accuracy (c) GPU Run-time Reduction (d) ASIC Evaluation

Figure 5: Pipeline's balanced proportion under input sequences length growth.

Methods	NUM	Format	LUT	FF	Fmax (MHz)	Latency (ns)	FOM
Xilinx FP	8	FP32	13254	18664	435	232.3	3.488
Hyft16	8	FP16	1072	824	625	12.4	42.194
Hyft32	8	FP32	2399	1528	526	19	34.290
TCAS-I'22	10	Fixed 16	1476	698	500	NA	36.798
ISCAS'23	8	FP16	909	333	476	10.5	49.056
TCAS-II'22	1	FP16	128	97	588	22.1	41.813
Ours	**1024**	**DBFP**	**10872**	**3743**	**455**	**73**	**509.563**

Table 3: SOTA Softmax accelerators comparison on FPGAs



(a) Scaling Scalability (*x*-axis: Seq. Len.) (b) Running Pipeline

Figure 6: Pipeline's balance under sequences scaling up.

FPGA resource utilization (LUT and FF), maximum operating frequency, and FOM (a comprehensive metric).

$$\text{FOM} = \frac{F_{\max} \times N \times W}{\text{LUT} + \text{FF}} \tag{11}$$

, where W and N denote the precision and numbers of the inputs. A higher FOM value indicates better performance.

Tab. 3 shows our comparison with multiple SOTA designs, including Xilinx FP IP (Koca, Do, and Chang 2023) baseline. While existing Softmax accelerators only support input bandwidths under 16, our design uniquely accommodates the larger bandwidths needed for modern LLMs. Testing with 1024-length sequences, our implementation achieves 54.21% less resource usage while operating at higher frequencies, reduces processing latency by 62.5%, and delivers 128x higher computational bandwidth. Notably, it shows a 10x FOM improvement over ISCAS'23 SOTA, demonstrating the significant potential of our approach for nonlinear operation hardware units.

Design Evaluation on ASIC. ASIC implementations can be supplemented with more accurate data on power consumption, maximum clock frequency, and scalability for high-volume applications. We evaluate the hardware design on ASIC using four key metrics: Area-Delay-Product (ADP, Area $\times$ Latency), Energy-Delay-Product (EDP, Energy $\times$ Latency), and Throughput (Freq $\times$ Bandwidth).

For scenarios with a uniform input sequence length of 1024, we normalized the experimental results to 28nm. Fig. 5 (d) compares our design's normalized PPA (Power, Performance, and Area) metrics with SOTA designs (Cardarilli et al. 2021; Kouretas and Paliouras 2020). Our design is capable of handling large bandwidth and long input sequences. Hence, there is an average 10% increase in area compared to the SOTA design. However, this increase is offset by a significant enhancement in both energy consumption and throughput, exceeding 10x. In this context, the additional area requirement is considered a tolerable trade-off for substantial improvements in performance and efficiency.

Hardware Scalability. The design keeps scalability in mind. To demonstrate this feature, we conducted tuning tests with input sizes ranging from 8 to 4096 elements. Fig. 6 shows the total latency and each part of the processing pipeline. The total computation time grows exponentially with input size due to the quadratic relationship between input length and the size of the processed matrix. The scaled histogram in Fig. 6 shows the balanced growth in the proportion of time consumed by each pipeline level as the input size increases. No single component shows disproportionate latency growth; instead, pipeline allocation becomes more balanced with longer sequences. These observations evidenced the parallelism and scalability of the design.

Conclusion

We present DBFP, an enhanced BFP variant optimizing nonlinear operations. Our DB-Attn framework, enables efficient Attention, advancing narrow-precision LLM inference.

Lessons learned. Different from the conventional solutions that attempted to optimize nonlinear computation solely through hardware design or software techniques, this work shows that using an algorithm/hardware co-designed approach (DB-Attn), computation latency and memory accesses can be largely improved with light overhead. The construction of DBFP and the algorithm-driven hardware provide key insights and effective means that foster a collaborative environment of hardware and the algorithm for LLMs research that neither discipline could achieve independently.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful and helpful feedback. This work is supported by the National Key Research and Development Program (Grant No.2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086), the Basic Research Program of Jiangsu (Grants No. BK20243042 and NO. BG2024010), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005).

References

Burgess, N.; Milanovic, J.; Stephens, N.; Monachopoulos, K.; and Mansell, D. 2019. Bfloat16 processing for neural networks. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, 88–91. IEEE.

Cardarilli, G. C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Nannarelli, A.; Re, M.; and Spanò, S. 2021. A pseudo-softmax function for hardware-based high speed image classification. *Scientific reports*, 11(1): 15307.

Choquette, J.; Lee, E.; Krashinsky, R.; Balan, V.; and Khailany, B. 2021. 3.2 the a100 datacenter gpu and ampere architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, 48–50. IEEE.

Darvish Rouhani, B.; Lo, D.; Zhao, R.; Liu, M.; Fowers, J.; Ovtcharov, K.; Vinogradsky, A.; Massengill, S.; Yang, L.; Bittner, R.; et al. 2020. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. *Advances in neural information processing systems*, 33: 10271–10281.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Drumond, M.; Lin, T.; Jaggi, M.; and Falsafi, B. 2018. Training dnns with hybrid block floating point. *Advances in Neural Information Processing Systems*, 31.

Du, G.; Tian, C.; Li, Z.; Zhang, D.; Yin, Y.; and Ouyang, Y. 2019. Efficient softmax hardware architecture for deep neural networks. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 75–80.

Frantar, E.; and Alistarh, D. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, 10323–10337. PMLR.

Jha, C. K.; Prasad, K.; Srivastava, V. K.; and Mekie, J. 2020. FPAD: a multistage approximation methodology for designing floating point approximate dividers. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. IEEE.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; Casas, D. d. l.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Jouppi, N. P.; Yoon, D. H.; Kurian, G.; Li, S.; Patil, N.; Laudon, J.; Young, C.; and Patterson, D. 2020. A domain-specific supercomputer for training deep neural networks. *Communications of the ACM*, 63(7): 67–78.

Kim, S.; Gholami, A.; Yao, Z.; Mahoney, M. W.; and Keutzer, K. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*, 5506–5518. PMLR.

Kim, S.; Hooper, C.; Wattanawong, T.; Kang, M.; Yan, R.; Genc, H.; Dinh, G.; Huang, Q.; Keutzer, K.; Mahoney, M. W.; Shao, Y. S.; and Gholami, A. 2023. Full Stack Optimization of Transformer Inference: a Survey. arXiv:2302.14017.

Koca, N. A.; Do, A. T.; and Chang, C.-H. 2023. Hardware-efficient Softmax Approximation for Self-Attention Networks. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. IEEE.

Kouretas, I.; and Paliouras, V. 2020. Hardware implementation of a softmax-like function for deep learning. *Technologies*, 8(3): 46.

Lin, Y.; Tang, H.; Yang, S.; Zhang, Z.; Xiao, G.; Gan, C.; and Han, S. 2024. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, 10012–10022.

Lo, Y.-C.; Lee, T.-K.; and Liu, R.-S. 2023. Block and subword-scaling floating-point (BSFP): An efficient non-uniform quantization for low precision inference. In *The Eleventh International Conference on Learning Representations*.

Ma, Y.; Li, H.; Zheng, X.; Ling, F.; Xiao, X.; Wang, R.; Wen, S.; Chao, F.; and Ji, R. 2024. Affinequant: Affine transformation quantization for large language models. *arXiv preprint arXiv:2403.12544*.

Mellempudi, N.; Kundu, A.; Das, D.; Mudigere, D.; and Kaul, B. 2017. Mixed low-precision deep learning inference using dynamic fixed point. *arXiv preprint arXiv:1701.08978*.

Meta. 2024. Meta Llama 3: Advancing Generative AI Responsibly. https://ai.meta.com/blog/meta-llama-3/. Accessed: 2024-08-01.

Nagel, M.; Baalen, M. v.; Blankevoort, T.; and Welling, M. 2019. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 1325–1334.

NVIDIA. 2020. Int4 precision for AI inference. https://developer.nvidia.com/blog/int4-for-ai-inference/. Accessed: 2024-08-01.

NVIDIA. 2022. TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x. https://blogs.nvidia.com/blog/tensorfloat-32-precision-format/. Accessed: 2024-08-01.

Rouhani, B. D.; Zhao, R.; More, A.; Hall, M.; Khodamoradi, A.; Deng, S.; Choudhary, D.; Cornea, M.; Dellinger, E.; Denolf, K.; et al. 2023. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*.

S, R. 2021. Reduced Softmax Unit for Deep Neural Network Accelerators. arXiv:2201.04562.

Song, Z.; Liu, Z.; and Wang, D. 2018. Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Stevens, J. R.; Venkatesan, R.; Dai, S.; Khailany, B.; and Raghunathan, A. 2021. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 469–474. IEEE.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Wang, M.; Lu, S.; Zhu, D.; Lin, J.; and Wang, Z. 2018. A high-speed and low-complexity architecture for softmax function in deep learning. In *2018 IEEE asia pacific conference on circuits and systems (APCCAS)*, 223–226. IEEE.

Xia, H.; Zheng, Z.; Li, Y.; Zhuang, D.; Zhou, Z.; Qiu, X.; Li, Y.; Lin, W.; and Song, S. L. 2023. Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity. arXiv:2309.10285.

Xia, T.; and Zhang, S. Q. 2023. Softmax Acceleration with Adaptive Numeric Format for both Training and Inference. *arXiv preprint arXiv:2311.13290*.

Yeh, T.; Sterner, M.; Lai, Z.; Chuang, B.; and Ihler, A. 2022. Be Like Water: Adaptive Floating Point for Machine Learning. In *International Conference on Machine Learning*, 25490–25500. PMLR.

Zhang, Y.; Peng, L.; Quan, L.; Zhang, Y.; Zheng, S.; and Chen, H. 2023. High-precision method and architecture for base-2 softmax function in dnn training. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(8): 3268–3279.

Zhang, Y.; Zhang, Y.; Peng, L.; Quan, L.; Zheng, S.; Lu, Z.; and Chen, H. 2022. Base-2 softmax function: Suitability for training and efficient hardware implementation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(9): 3605–3618.

Zou, L.; Zhao, W.; Yin, S.; Bai, C.; Sun, Q.; and Yu, B. 2024. BiE: Bi-Exponent Block Floating-Point for Large Language Models Quantization. In *Forty-first International Conference on Machine Learning*.