FLUTE: A LUT-Based Floating-Point Engine for Efficient LLM Acceleration Exploiting Core-Tail Hierarchical Alignment

Tielong Liu[1,2,*], Zeyu Zhu[1,2,*], Gang Li[1,†], Zitao Mo[1], Chen Bai[3], Junwen Si[1]
Peihuan Ni[1], Minnan Pei[1], Peisong Wang[1], Zhuoran Song[4], Xiaoyao Liang[4], Jian Cheng[1,†]
[1]C[2]DL, Institute of Automation, Chinese Academy of Sciences, Beijing, China
[2]School of Future Technology, University of Chinese Academy of Sciences, Beijing, China
[3]Fudan University, Shanghai, China
[4]Shanghai Jiao Tong University, Shanghai, China
{liutielong2022, gang.li, mozitao2017, nipeihuan2024}@ia.ac.cn; junwensi@smail.nju.edu.cn
{zhuzeyu211, peiminnan19}@mails.ucas.ac.cn; peisong.wang@nlpr.ia.ac.cn
cbai@fudan.edu.cn; {songzhuoran, liang-xy}@sjtu.edu.cn; jcheng@nlpr.ia.ac.cn

Abstract*—Weight-only floating-point (FP) quantization is becoming increasingly attractive for Large Language Models (LLMs). However, efficient hardware support for FP-quantized models remains challenging. Lookup-Table (LUT)-based computing can reduce much of the arithmetic cost, yet it is not naturally compatible with FP data: naive exponent alignment causes severe bit-plane explosion, largely undermining the efficiency benefits of LUT-based computing. To address this challenge, we propose FLUTE, an algorithm–hardware co-design accelerator for efficient inference on low-bit FP-quantized LLMs. Our key insight is that FP-quantized LLM weights exhibit a pronounced exponent clustering effect: most weights concentrate in a narrow core exponent range, while only a small fraction falls into a sparse tail. Guided by this insight, on the algorithm side, FLUTE introduces Core-Tail Hierarchical Exponent Alignment (CT-HEA), which partitions weights into core and tail groups and aligns them hierarchically to form a compact LUT-friendly representation. To efficiently support CT-HEA, on the hardware side, FLUTE further introduces the Bipolar Expansion Unit (BEU) to decode the ternary bit-planes induced by CT-HEA into parallel binary lookups for compact LUT realization, and further improves efficiency through PE-centric microarchitectural co-optimizations and energy-driven multi-granularity exploration of dataflow and LUT fan-out parallelism. Extensive evaluations demonstrate that FLUTE delivers average improvements of 1.85–3.13× in performance, 1.62–6.32× in area efficiency, and 2.37–6.02× in energy efficiency over the FP baseline, FIGNA, FIGLUT, and AxCore, while maintaining model accuracy.

***Index Terms*—LLM, Weight-only Quantization, Floating-Point, LUT, Accelerator**

I. Introduction

Large Language Models (LLMs) have achieved remarkable success across a wide range of domains [1]–[5] and continue to scale, driven by scaling laws [6]. As model sizes grow, efficient deployment becomes increasingly challenging. On the one hand, massive model parameters impose substantial storage and memory traffic pressure. On the other hand, the dominant workload in LLM inference, matrix multiplication, incurs substantial computation cost.

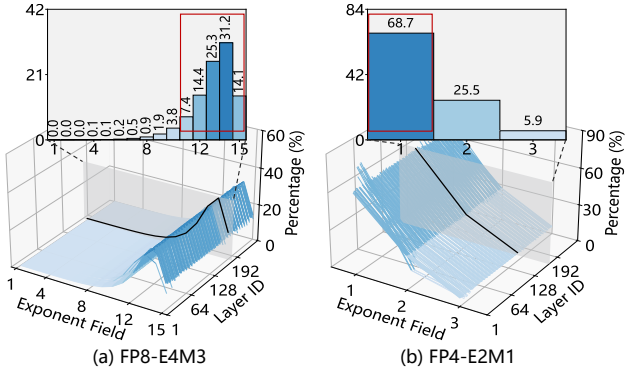
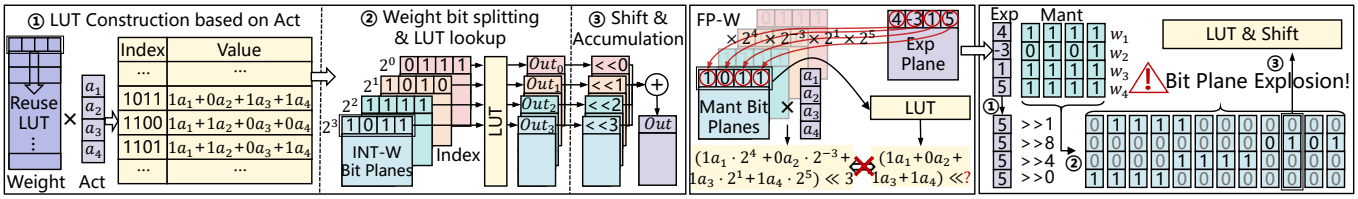


Fig. 1. Representative exponent distributions of FP8-E4M3- and FP4-E2M1-quantized weights across different layers of Qwen3-8B, showing a pronounced exponent clustering effect.

To mitigate these storage and computation burdens, quantization has become the de facto standard for efficient LLM inference [7]–[11]. In particular, weight-only quantization has emerged as the prevailing paradigm because it substantially reduces weight storage and bandwidth demand without disturbing activations, especially activation outliers [7], [9]. Among recent advances, low-bit FP quantization and its variants have drawn increasing attention [12]–[16], as explicit exponents provide a wider numeric range and usually achieve better post-quantization accuracy than INT formats [12], [16], [17]. However, this accuracy advantage comes at a cost: once quantized weights move from INT to FP, efficient computation can no longer rely on simple integer-style computation, but must instead introduce more complex FP arithmetic. As a result, enabling hardware-efficient computation on FP-quantized weights has become increasingly urgent.

Lookup-Table (LUT)-based computing is a promising approach to improve the area and energy efficiency of low-bit computation by replacing on-the-fly arithmetic with table

*Equal contribution.
†Corresponding author.



(a) Workflow of bit-serial LUT-based mpGEMM. (b) Mismatch for FP Quantization. (c) Naïve Solution.

Fig. 2. Standard LUT execution workflow, structural mismatch in FP representation, and the naive solution.

lookup and reuse [18]–[21]. Recent LUT-based accelerators have demonstrated substantial efficiency gains for INT-centric quantized inference [22]–[24], where activation forms a LUT that is queried by bit-serial weights. However, this advantage does not directly extend to FP data due to a fundamental semantic mismatch between LUT-based execution and FP representation: LUT-based bit-serial execution relies on shared bit-plane semantics, whereas FP representation makes each bit's contribution exponent-dependent.

As illustrated in Fig. 2(a), LUT-based INT execution works because bits at the same position share the same power-of-two semantics. This allows the bits from multiple INT values in one bit plane to be packed into a compact LUT index. After lookup, the retrieved partial sum can be scaled by a simple shift. However, FP data breaks this regularity. As depicted in Fig. 2(b), an FP value is jointly determined by its mantissa and exponent, so the contribution of each mantissa bit depends on both its bit position and its associated exponent. Therefore, even if mantissa bits come from the same bit plane, the looked-up result no longer corresponds to a shared scaling factor and cannot be recovered by a simple shift. A naive workaround is to first align all mantissas to a shared exponent, typically chosen as the largest exponent in that group [25], before lookup. However, as Fig. 2(c) shows, the wide exponent range of FP formats can significantly widen the mantissas, causing severe bit-plane explosion. For example, under FP8-E4M3, a mantissa may expand to as many as 19 bits after alignment. This substantially increases bit-plane processing overhead and weakens the energy efficiency of LUT-based computing.

However, this mismatch does not imply that LUT-based FP execution is infeasible; rather, it indicates the need for a structure-aware design beyond existing INT-oriented approaches. Our key observation, as illustrated in Fig. 1, is that FP-quantized LLM weights exhibit a strong clustering effect in the exponent domain: dominant core weights concentrate within a narrow exponent range, while only a small fraction falls into a sparse tail. This structure makes the practical cost of FP alignment much lower than it appears, and suggests that efficient FP-LUT execution should treat the dominant core and the sparse tail differently, rather than forcing all weights to follow a single alignment rule.

Guided by this observation, we propose **FLUTE**, an algorithm–hardware co-design accelerator for LUT-based inference on FP-quantized LLMs. On the algorithm side, FLUTE introduces **Core-Tail Hierarchical Exponent Alignment (CT-HEA)**, which partitions weights into *Core-Exponent Weights* (CEWs) and *Tail-Exponent Weights* (TEWs), and adopts hierarchical exponent alignment. CT-HEA aligns the dominant core and the sparse tail differently, thereby preserving LUT-friendly regularity while avoiding excessive information loss. On the hardware side, to address the efficiency challenges introduced by CT-HEA, we further apply coordinated optimizations at multiple levels. First, to substantially reduce the LUT size caused by CT-HEA-induced ternary queries, FLUTE introduces a **Bipolar Expansion Unit (BEU)** that decodes each ternary query into parallel binary lookups. Second, we co-design the PE microarchitecture with the FP-LUT computation flow to reduce the area and power overhead introduced by additional lookups and compensation operations. Finally, we perform energy-driven exploration of coarse-grained dataflow choices and fine-grained LUT fan-out trade-offs to identify the most energy-efficient design point. To the best of our knowledge, FLUTE is the first LUT-based accelerator to efficiently support FP-quantized LLM inference.

The main contributions of this paper are summarized as follows:

- We identify an exponent clustering effect in FP-quantized LLM weights and theoretically show that this property holds across FP quantization formats of different precisions.
- We propose CT-HEA, which enables LUT-based FP computation by mapping FP-quantized weights to a unified LUT-friendly representation through core-tail partitioning and hierarchical exponent alignment.
- We design the FLUTE architecture, which introduces BEU, PE-level optimizations, and multi-granularity exploration of dataflow and LUT fan-out parallelism to efficiently support the CT-HEA algorithm.
- We implement and evaluate FLUTE in a 28nm technology node across diverse precision settings. Compared with the FP baseline, FIGNA [26], FIGLUT [24], and AxCore [27], FLUTE achieves average improvements of 1.85–3.13× in performance, 1.62–6.32× in area efficiency, and 2.37–6.02× in energy efficiency while maintaining model accuracy.

II. Background and Related Work

A. GEMM and Weight-only Quantization for LLMs

Large Language Models (LLMs) are built from stacked decoder blocks containing linear layers, attention, and nonlinear units [28]–[30]. During inference, linear layers dominate the computation [31]; for example, they account for over 95% of the operations in OPT-175B at context lengths below 4k [27]. Thus, efficient LLM inference largely depends on

optimizing general matrix-matrix multiplications (GEMMs) in linear layers.

Quantization is widely used to reduce these costs. Since aggressively quantizing dynamic, outlier-prone activations often causes severe accuracy loss [32]–[35], and activations occupy much less memory than weights during decoding [36], weight-only quantization has become the dominant paradigm. Prior outlier-aware quantization methods exploit non-uniform LLM weight distributions to reduce quantization error, e.g., by protecting salient channels, outlier values, or Hessian-sensitive weights [7], [9], [37], [38]. These methods mainly identify outliers in the value domain. In contrast, CT-HEA distinguishes core and tail weights according to the exponent-domain clustering structure after FP quantization, rather than directly relying on weight magnitude.

This paradigm transforms linear operations into mixed-precision GEMMs (mpGEMMs). On commercial GPUs, low-bit weights are usually dequantized on the fly back to high precision [26], [39], which reduces off-chip traffic but incurs nontrivial on-chip overhead. To avoid this, prior accelerators directly support mixed-precision computation. For example, FIGNA [26] proposes an FP-INT MAC accelerator, while AxCore [27] approximates floating-point multiplication with integer addition. However, these designs are tightly coupled to fixed precisions and data formats, limiting their support for diverse quantization algorithms.

B. FP Quantization and Its Optimization

Within weight-only quantization, the data format is critical. Owing to its wide dynamic range and high post-quantization accuracy, floating-point (FP) is a promising choice. According to IEEE 754 [40], FP values are represented as $Value = (-1)^S \times 2^{E-Bias} \times (1+M)$, where S, E, and M denote the sign, exponent, and mantissa. While prior efforts mainly adopted hardware-friendly INT formats [32], [33], [41], [42], later FP8 studies showed better post-training accuracy [12], [13]. More recent work further pushes FP quantization to lower bit-widths, such as ZeroQuant(4+2) [17] and FP6-LLM [43]. However, sub-byte FP arithmetic still incurs substantial hardware overhead.

Block Floating-Point (BFP) [25] mitigates this cost by sharing one exponent across a block, enabling direct integer computation, but it suffers from inherent accuracy loss. Recovering this loss with BFP-aware training is costly [44]–[46]. PTQ-based BFP variants avoid retraining but introduce extra metadata overheads. For example, BBAL [15] adds a flag bit for bidirectional shifting, while BiE [14] adopts dual-exponent grouping; both require one extra bit per weight. Fundamentally, prior shared-exponent methods follow a *value-wise exponent handling* paradigm, where different exponent classes or outlier values must be explicitly distinguished using extra per-value metadata such as flags or selectors. FLUTE instead adopts a *distribution-aware exponent structuring* paradigm, exploiting the statistical concentration of exponents across FP-quantized weights. By extracting structure at the distribution level rather than handling exceptions per value, FLUTE greatly reduces metadata and lowers hardware overhead.

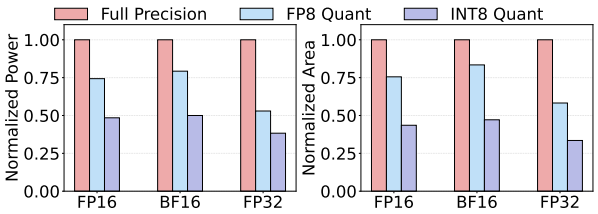


Fig. 3. Relative power and area of MAC units with FP16, BF16, and FP32 activations (x axis) under full-precision, FP8-quantized, and INT8-quantized weights.

C. LUT-Based mpGEMM

To avoid the high area and energy cost of direct mixed-precision MACs, Lookup Table (LUT)-based approaches have emerged as efficient alternatives [18]–[22]. A naive design that precomputes all MAC results for INT4 weights and FP16 activations would require a prohibitively large table of $2^4 \times 2^{16} = 2^{20}$ entries, which is unaffordable. Therefore, recent LUT-based LLM accelerators [23], [24] typically construct tables online.

As illustrated in Fig. 2(a), LUT-based computation replaces repeated MAC operations with three steps. First, LUT construction uses four activations a_0, a_1, a_2, a_3 to generate a $2^4 = 16$-entry LUT, where each 4-bit index selects a subset of activations; for example, "1011" retrieves $a_0 + a_2 + a_3$. Second, weight bit splitting forms one lookup index per bit plane using bits from four adjacent INT weights. Third, shift and accumulation scales each lookup result by its bit position and accumulates the results. Since all bits in the same INT bit plane share the same numerical significance 2^b, this process reconstructs the final dot product.

Such bit-serial execution is necessary to keep the table compact. Directly using four 8-bit weights as a joint index would require an infeasible 2^{32}-entry on-chip table. However, this bit-serial workflow is better suited to low-bit INT quantization. Since latency and table accesses scale linearly with the weight bit-width, moving from 4-bit to higher-precision substantially increases compute cycles and weakens the throughput and efficiency benefits of LUT computing.

III. Challenges and Opportunities

A. Exponent-Induced Challenges and Opportunities

Mismatch between LUT and FP. Although FP quantization preserves accuracy better than INT formats, its deployment efficiency remains challenging because FP arithmetic is inherently expensive in hardware. Fig. 3 compares different weight formats within each activation-precision group. Across all groups, FP8 weight quantization reduces area and power from the full-precision baseline, but remains more costly than INT8 quantization due to exponent handling and FP arithmetic logic. LUT-based computing is a promising approach to further improve area and energy efficiency by replacing on-the-fly arithmetic with table lookup and reuse, and has proven highly

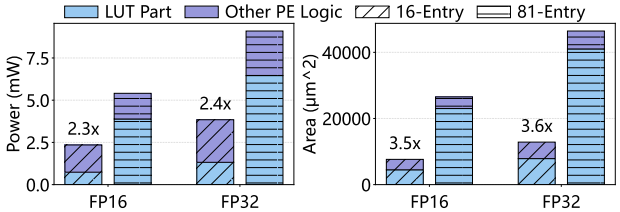


Fig. 4. Area and power comparison between 16-entry and 81-entry LUT-based PEs under FP16/FP32 activations.

effective for INT-centric low-bit inference [18]–[21]. Unfortunately, this advantage cannot be directly extended to FP data, because the bit-serial LUT execution and FP representation are fundamentally mismatched: the former relies on shared bit-plane semantics, whereas the latter is jointly determined by mantissas and exponents. As illustrated in Fig. 2(b), the contribution of an FP mantissa bit depends not only on its bit position, but also on the associated exponent. Consequently, bits from the same mantissa plane no longer share the same scaling factor, as in LUT-based INT computation, and the lookup-and-shift workflow used in INT-oriented LUT execution no longer applies.

Expensive naive solution. A natural workaround is to align each FP packing group to a shared exponent before bit-plane slicing and LUT lookup, so that the aligned mantissas share the same scaling factor. This exponent is naturally chosen as the largest one in the group to avoid losing significant bits. However, as shown in Fig. 2(c), such alignment causes severe bit-plane explosion: the mantissas of weights with smaller exponents must be right-shifted, and preserving the shifted-out information requires many additional bit planes due to exponent differences. These expanded bits also cannot be simply truncated, as in conventional BFP, without incurring substantial numerical error. Taking FP8-E4M3 as an example, naive alignment expands a single FP8 value to 19 bit planes, including 18 magnitude bit planes (1 implicit leading bit, 3 original mantissa bits, and up to 14 alignment-induced bit planes) plus 1 sign bit. This substantial increase in bit-plane count sharply raises bit-serial processing overhead, making naive shared-exponent alignment impractical for FP-LUT execution.

A more realistic implementation may avoid worst-case provisioning by using a fixed alignment bit budget. However, this still leads to an unfavorable accuracy–efficiency trade-off: small budgets lose tail information, while large budgets require many bit-plane iterations. As shown in Table I, naive alignment needs 10 bits/weight to approach the perplexity of our 6-bit final design. Moreover, if we use lossless per-group provisioning, where each group keeps enough bits to cover its maximum-to-minimum exponent span, the resulting average bit-width still reaches about 14–15 bits/weight in our profiling. These results indicate that simply tuning the alignment bit budget cannot fundamentally resolve the FP-LUT mismatch, motivating a more structure-aware alignment strategy.

Opportunities from the exponent distribution. The above cost arises directly from shared-exponent alignment: values in each group must be aligned to the largest exponent, while the resulting expanded bits cannot be naively truncated. This motivates us to analyze the exponent distribution of FP-quantized weights. Our analysis reveals that the exponents of FP-quantized weights are far from arbitrarily distributed. Instead, they exhibit a pronounced clustering effect: dominant core weights concentrate within a narrow exponent range, while only a small fraction falls into a sparse tail. Fig. 1 provides an example on Qwen3-8B: under FP8-E4M3 quantization, more than 92% of weights fall within the exponent range from 11 to 15, while under FP4-E2M1, more than 94% of weights concentrate between exponents 1 and 2. Moreover, Section IV-A further proves that this clustering behavior persists across FP quantization formats of different precisions. These observations indicate that the practical cost of FP alignment can be much lower than it appears, since most weights with a narrow exponent range require only small exponent shifts and only a sparse tail incurs large alignment overhead. This in turn suggests that efficient FP-LUT execution should treat the dominant core and the sparse tail differently, rather than forcing all weights to follow a single alignment rule. Based on this insight, we propose the CT-HEA algorithm to mitigate bit-plane explosion through hierarchical exponent alignment.

TABLE I
NAIVE ALIGNMENT BIT-WIDTH SWEEP ON LLAMA-2-7B. FP16 IS REPORTED AS THE FULL-PRECISION REFERENCE.

Setting (Bit)	FP16	3	4	5	6	7	8	9	10
Naive alignment	9.238	30.524	10.832	9.683	9.415	9.396	9.337	9.307	9.271
Ours	9.238	14.024	9.540	9.414	9.271	–	–	–	–

B. Hardware Challenges and Multi-Granularity Trade-offs

The exponent clustering effect enables CT-HEA to construct a more LUT-friendly representation for FP-quantized weights. However, this algorithmic opportunity does not directly yield an efficient hardware implementation. Once mapped onto a LUT datapath, the new representation introduces challenges at both the microarchitectural level and across multi-granularity design trade-offs.

Microarchitectural-level challenges. A standard LUT-based PE first precomputes the partial sums of all combinations of a small activation group into a compact LUT, and then queries it with packed bit-plane indices formed from adjacent weights. For a 4-activation group, if each position in the query only needs binary selection semantics, the LUT requires only $2^4 = 16$ entries. This binary indexing is incompatible with CT-HEA. Since each FP weight contains one sign bit and m mantissa bits, and the sign must be fused with each mantissa bit to determine its numerical meaning, each effective bit-plane becomes ternary, i.e., $\{-1, 0, +1\}$. As a result, the same 4-activation group now requires $3^4 = 81$ lookup combinations, creating a **ternary indexing crisis**. As shown in Fig. 4, an 81-entry LUT increases the area of a PE with 32 RAC units by 3.5–3.6× and its power by 2.3–2.4× relative to the 16-entry baseline, largely undermining the benefit of LUT-based computing. Beyond LUT expansion, ternary query realization also introduces extra PE-execution overheads, including

Algorithm 1 CT-HEA Encoding for One LUT-Shared Weight Block

Input: FP-quantized weights $\mathcal{W}$; core window width r.
Output: Aligned payloads $\mathcal{P}$; block exponent e_{unified}; tail flags $\mathcal{F}_{\text{tail}}$; tail metadata $\mathcal{M}_{\text{tail}}$.

Extract (s_i, e_i, m_i) for each $w_i \in \mathcal{W}$.
Find the width-r exponent window $[e_{\text{low}}, e_{\text{high}}]$ covering the most weights.
Select e_{unified} from $[e_{\text{low}}, e_{\text{high}}]$ and split $\mathcal{W}$ into quadruples G_j.
for each quadruple G_j **do**
    $\mathcal{T}_j \leftarrow w_i \in G_j \mid e_i \notin [e_{\text{low}}, e_{\text{high}}]$.
    if $|\mathcal{T}_j| = 0$ **then**
        Align G_j to e_{unified}; $\mathcal{F}_{\text{tail}}[j] \leftarrow 0$.
    else if $|\mathcal{T}_j| = 1$ **then**
        Align core weights to e_{unified} and truncate the single TEW; $\mathcal{F}_{\text{tail}}[j] \leftarrow 0$.
    else
        $e_{\text{tail}}[j] \leftarrow \max_{w_i \in G_j} e_i$; align G_j to $e_{\text{tail}}[j]$.
        Store $e_{\text{tail}}[j]$ in $\mathcal{M}_{\text{tail}}$; $\mathcal{F}_{\text{tail}}[j] \leftarrow 1$.
    end if
end for
return $\mathcal{P}$, e_{unified}, $\mathcal{F}_{\text{tail}}$, and $\mathcal{M}_{\text{tail}}$.

additional lookup, result reconstruction, and compensation operations, which increase the cost of the read-and-accumulate path. Therefore, microarchitectural optimization is required to reduce both the LUT overhead caused by ternary indexing and the PE overhead introduced by the resulting extra operations.

Multi-granularity trade-off challenges. Even when the microarchitectural overhead is controlled, overall efficiency still depends on trade-offs at multiple granularities. At the coarse granularity, different dataflows expose different reuse opportunities. In our design, LUT generation itself requires FP multiplication, making table construction expensive. A Weight-Stationary (WS) dataflow favors weight reuse, but rebuilds LUTs for every bit plane of each weight tile, incurring high LUT-construction overhead. By contrast, an Input-Stationary (IS) dataflow improves activation-side LUT reuse and amortizes table construction cost, but increases weight-buffer accesses and SRAM energy due to more frequent weight fetching. Thus, coarse-grained dataflow selection must balance weight reuse against LUT-construction reuse. At the fine granularity, efficiency further depends on the degree of intra-PE LUT parallelism, quantified by the K_{RAC}, i.e., the number of RAC units sharing the same LUT within a PE. Under a fixed throughput budget, increasing K_{RAC} reduces the total number of LUTs in the array, but raises LUT fan-out and lookup-path overhead, thereby increasing both static leakage and dynamic lookup energy. Therefore, design space exploration is necessary to jointly optimize coarse-grained dataflow selection and fine-grained LUT fan-out parallelism, so as to maximize LUT-construction amortization and LUT sharing while controlling the associated memory and circuit overhead.

IV. Core-Tail Hierarchical Exponent Alignment

A. Exponent Distribution Analysis

To formalize the exponent clustering effect in Section III-A, we analyze the exponent distribution under symmetric group-wise quantization. Consistent with recent work [47], we model the linear layer weights as $W \sim \mathcal{N}(0, \sigma^2)$. Let the quantization scaling factor be $s = \max_i |w_i| / y_{\max}$, where $y_{\max}$ denotes the maximum representable value of the target quantized FP format. Consequently, the normalized magnitude y_i for any weight w_i is given by:

$$y_i = \frac{|w_i|}{s} = \frac{|w_i|}{\max_j |w_j| / y_{\max}} = \frac{y_{\max} \cdot |w_i|}{\max_j |w_j|}. \tag{1}$$

Since $y_{\max}$ is constant for a given FP format, the final quantized exponent distribution is determined by the relative magnitude ratio, defined as:

$$R_i = |w_i| / \max_j |w_j| \in [0, 1]. \tag{2}$$

Without loss of generality, we use the FP8-E4M3 format as an example. In this format, an exponent ≥ 11 is reached when $y_i \geq 2^{11-7} = 16$ (where 7 is the exponent bias). Given that the maximum representable value is $y_{\max} = 480$, this threshold corresponds to:$R \geq 16/480 = 1/30 \approx 3.33\%$.

This threshold is very small. To mathematically contextualize this within our bell-shaped distribution model, we note that the maximum absolute value within a typical quantization block is tightly bounded around 3σ (following the empirical three-sigma rule [48], [49]). Consequently, the 3.33% relative threshold equates to an absolute weight magnitude of merely $0.0333 \times 3\sigma \approx 0.1\sigma$. Thus, any weight $|w_i| \geq 0.1\sigma$ directly maps to an exponent ≥ 11. The theoretical proportion of weights surpassing this magnitude is:

$$P(|w| \geq 0.1\sigma) = 2 \times (1 - \Phi(0.1)) \approx 92.04\%. \tag{3}$$

This theoretical result closely matches our empirical measurements, which demonstrated that over 92% of E4M3 weights cluster in the aforementioned narrow exponent band. More importantly, the above derivation shows that exponent clustering is mainly governed by the relative-magnitude threshold after group-wise scaling, rather than by the exact weight distribution form. The Gaussian model is used only to demonstrate that the analytical trend is consistent with empirical observations. FLUTE instead relies on the empirically measurable core-tail exponent structure, which appears consistently across diverse LLMs and FP formats.

B. CT-HEA for a Unified Datapath

Shared Dominant Exponent and Quadruple Formation. Based on the exponent clustering effect and analysis in Section IV-A, we divide the exponent space into a dominant core region and a sparse tail region. We refer to weights in the core region as Core-Exponent Weights (CEWs) and those in the tail region as Tail-Exponent Weights (TEWs). To support FP-LUT computation efficiently while preserving model accuracy, we aim to organize both types of weights into a unified LUT datapath. As discussed in Section II-C, each LUT is generated from 4 consecutive activations. We adopt group-wise quantization along the column dimension with group size G. For $W \in \mathbb{R}^{M \times K}$, the column dimension corresponds to the M-dimension; thus each column is partitioned into

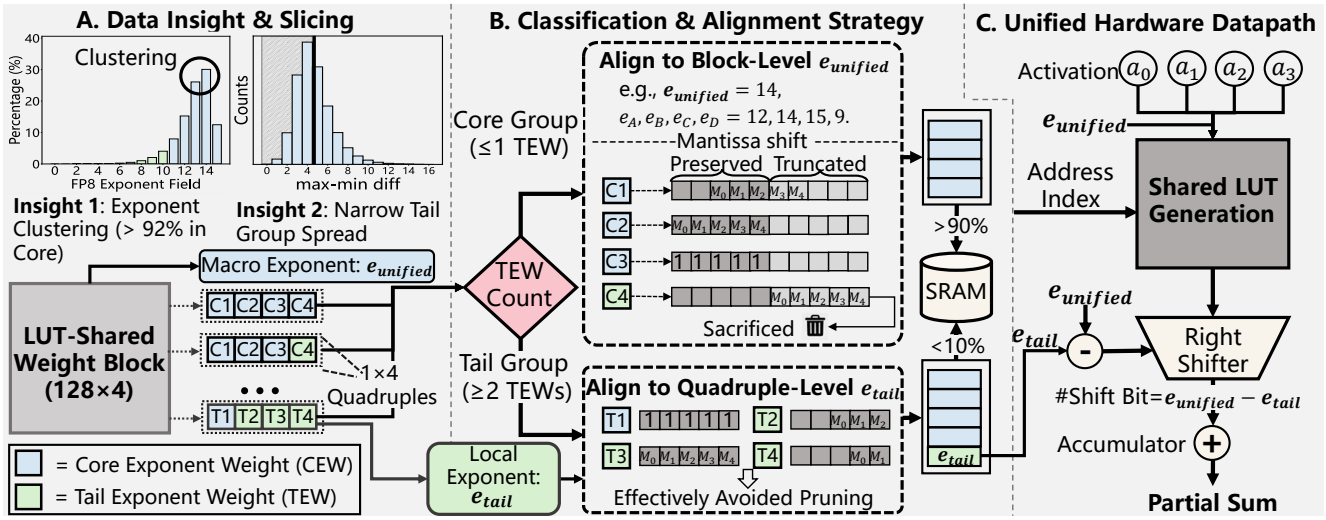


Fig. 5. Overview of CT-HEA. (a) Weights are sliced into quadruples based on distribution insights. (b) Quadruples are dynamically classified and aligned using hierarchical exponents. (c) Weights are processed through a unified LUT datapath.

M/G groups. This aligns naturally with the four-column LUT-sharing structure. Accordingly, we partition the weights into LUT-shared blocks, each of size $G \times 4$. For each block, we choose a dominant exponent as the shared exponent $e_{unified}$. All weights in the block are aligned to $e_{unified}$ and then sliced into bit planes, while the four horizontally adjacent weights in the same row jointly form the lookup index. We define these four weights as a **quadruple** and use it as the basic processing granularity, as shown in Fig. 5(a). For the dominant CEWs, their exponents are already close to $e_{unified}$, so only a few mantissa bits are needed for alignment while preserving accuracy well. In the FP8 case, this shared alignment can be represented with 1 sign bit and 5 mantissa bits, and mapped to a unified LUT computation flow.

TEW-Induced Dilemma and Insight. However, when a quadruple contains TEWs, this unified processing scheme becomes problematic. If such a quadruple is still forced to align to $e_{unified}$, a dilemma arises: preserving TEW information requires much higher precision for the heavily shifted results, reintroducing severe bit-width explosion; otherwise, if the retained mantissa bit-width remains limited, these TEWs are directly pruned to zero, causing significant accuracy degradation. Our profiling shows that this effect prunes nearly 8% of the network weights on top of quantization.

A direct alternative is to separate these TEWs from the main LUT path and process them using a dedicated FP MAC datapath. However, this dual-path design is impractical. For a 64×64 weight tile multiplied by a 64×1 activation vector, the LUT path requires 6 cycles for a 6-bit CEW representation. Given the empirical TEW ratio of about 8%, the tile contains approximately 327 TEWs, so avoiding stalls would require the FP path to sustain 55 $(327/6)$ FP MACs per cycle. Such an auxiliary FP path is highly expensive relative to the lightweight LUT path.

Since the above challenges mainly stem from the exponent characteristics of TEWs, we further analyze the intra-group exponent distribution of quadruples containing TEWs. This leads to our second key insight: although these quadruples contain tail weights, their intra-group exponent spread is usually still narrow. Therefore, applying alignment within each tail-heavy quadruple allows most tail information to be preserved without introducing a second FP path. More specifically, a tail weight is pruned only when the maximum intra-group exponent gap exceeds the mantissa width retained in hardware. Our profiling shows that this condition is often avoided in practice. For FP8, retaining a tail weight requires the maximum intra-group exponent gap to be smaller than 5, and 52.9% of Tail Groups satisfy this condition. Similarly, for FP6-E3M2, the required maximum gap is smaller than 4, and the satisfaction probability reaches 66.0%. This indicates that more localized exponent alignment can effectively preserve most tail information in practice.

CT-HEA for Unified LUT Execution. Based on the above analysis, we propose Core-Tail Hierarchical Exponent Alignment (CT-HEA). Fig. 5 illustrates the overall organization and hardware mapping, while Algorithm 1 summarizes the corresponding offline encoding flow for one LUT-shared weight block. Specifically, CT-HEA first identifies the densest core exponent window and selects the block-level alignment exponent $e_{unified}$, as shown in lines 2–3 of Algorithm 1. It then operates at the fine-grained quadruple level. For each quadruple, we classify it into one of three cases, as depicted in Fig. 5(b): **(1) Pure Core Group**: If all four weights are CEWs, the quadruple is aligned to the block-level exponent $e_{unified}$, corresponding to lines 6–7 of Algorithm 1. **(2) Pseudo Core Group**: If the quadruple contains three CEWs and only one TEW, it is still aligned to $e_{unified}$, while the TEW with the larger exponent gap is algorithmically "sacrificed", allowing it to be truncated so that the dominant core pattern remains fully LUT-friendly, as shown in lines 8–9. **(3) Tail Group**: If the quadruple contains two or more TEWs, we classify it as a Tail Group and assign it a new, dedicated quadruple-

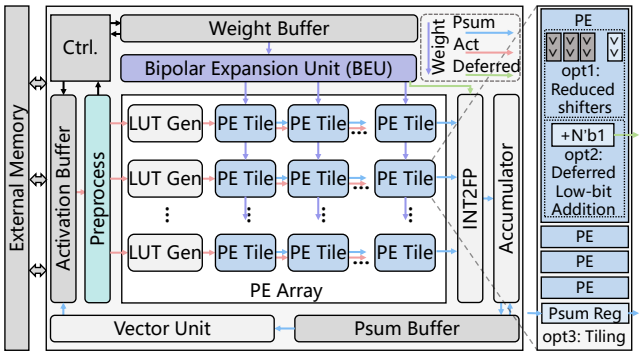


Fig. 6. Top-level architecture of FLUTE.

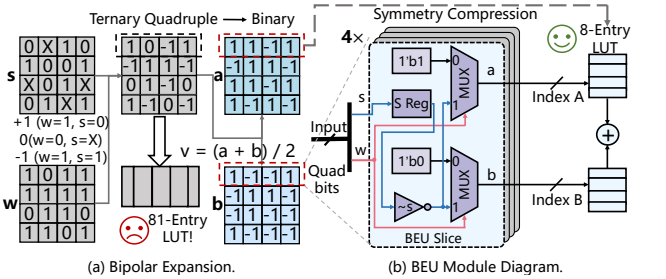


Fig. 7. Bipolar Expansion Unit (BEU). (a) Algorithmic decomposition of ternary quadruples. (b) BEU implementation with four slices, each processing one position of a quadruple.

level exponent e_{tail} equal to the maximum exponent within the Tail Group, as shown in lines 10–12. Despite this additional exponent, Tail Groups still use the same LUT datapath. During hardware execution, the LUT is constructed from the four adjacent activations and the block-level exponent $e_{unified}$. When a Tail Group queries the LUT, the table contents remain unchanged, and the hardware only right-shifts the LUT output by $e_{unified} - e_{tail}$ bits for compensation, as illustrated in Fig. 5(c). In this way, CT-HEA combines the block-level shared exponent $e_{unified}$ with the quadruple-level local exponent e_{tail}, enabling a unified LUT datapath while preserving tail information. Notably, since Tail Groups account for less than 10% of all quadruples, storing one extra 4-bit e_{tail} per Tail Group adds less than 0.1 bit per weight on average, corresponding to only about 1.7% relative storage overhead for a 6-bit representation.

V. The FLUTE Accelerator Architecture

A. Architecture Overview

Fig. 6 illustrates the overall FLUTE accelerator, built around a unified LUT-centric datapath. We adopt the IS dataflow, introduce the BEU alongside a fused preprocessing module, and implement a series of microarchitectural optimizations within the PE design. During execution, input activations are first fetched from the Activation Buffer and preprocessed before entering the LUT Generator. The generated LUTs are then broadcast across the PE array and stored row-wise in the PE tiles, with each PE row corresponding to one activation column. During computation, compressed bit-serial weights are fetched from the Weight Buffer and decoded by BEU into LUT indices, which are propagated downwards along the PE columns to trigger table lookups. As these lookup operations proceed, integer partial sums are accumulated horizontally across the PE Array, converted back to floating-point by the INT2FP unit, and then accumulated across tiles before being written to the Psum Buffer. Meanwhile, a lightweight Vector Processing Unit (VPU) handles non-GEMM operations during LLM inference.

B. Bipolar Expansion Unit (BEU)

CT-HEA represents FP8 weights in a 6-bit format, with 1 sign bit and 5 mantissa bits. However, during table lookup, the sign bit cannot index the table independently and must be fused with the mantissa. As a result, each bit-plane becomes ternary, taking values in {-1, 0, +1}. For a 4-activation group, ternary indexing would require a $3^4 = 81$-entry LUT, making the table prohibitively expensive.

As discussed in Section III-B, directly instantiating such an 81-entry LUT would introduce prohibitive hardware overhead and undermine the efficiency of LUT-based computing. To avoid this table expansion, we propose the Bipolar Expansion Unit (BEU), which adopts an index-decomposition strategy. Our core mathematical insight is that any ternary value $v \in \{-1, 0, +1\}$ can be equivalently decomposed into the arithmetic mean of two bipolar values $a, b \in \{-1, +1\}$. Specifically, the transformation rules are defined as:

$$\begin{aligned} +1 &= ((+1) + (+1))/2, \\ -1 &= ((-1) + (-1))/2, \\ 0 &= ((+1) + (-1))/2. \end{aligned} \tag{4}$$

Therefore, any ternary quadruple index can be decomposed into two distinct binary quadruples with an associated shift, as presented in Fig. 7(a).

At the hardware level, the BEU implements this expansion with negligible logic overhead. Let w denote the mantissa bit and s the sign bit. Since one sign bit is shared across multiple mantissa bits of a weight, s is fetched once from SRAM and cached locally in the S Reg. The BEU then maps the ternary state into binary signs, a and b (where logic 1 represents '+1' and logic 0 represents '-1'), as depicted in Fig. 7(b). This lightweight hardware transformation avoids the need for an 81-entry LUT by splitting the query into two standard binary lookups; the two fetched values are then added, while the division-by-2 is absorbed during LUT construction. We further combine BEU with previous symmetry-based compression [23], [24]: since bipolar inner products are anti-symmetric ($(a_1 + a_2 + a_3 + a_4) = -(-a_1 - a_2 - a_3 - a_4)$), only half of the combinations need to be stored, while the MSB of the 4-bit index is used to determine sign inversion. Consequently, the BEU reduces the LUT size to only 8 entries, minimizing area and power overheads.

C. Efficient Microarchitectural Optimizations

While the BEU eliminates the prohibitive LUT expansion caused by ternary indexing, it also complicates each Read-and-Accumulate (RAC) operation. Within each PE, every

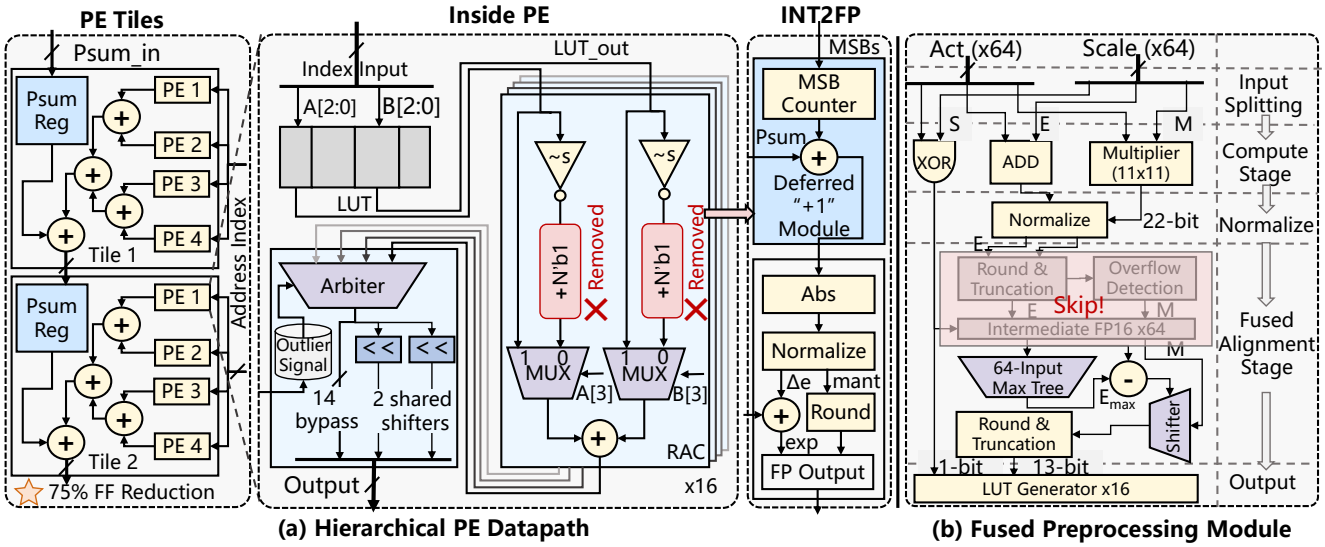


Fig. 8. Efficient microarchitectural optimizations. (a) Hierarchical PE datapath with shared shifters, deferred addition, and spatial accumulation. (b) Fused preprocessing module for integer conversion before LUT construction.

decomposed quadruple index triggers two parallel reads from 8-entry LUTs. The retrieved values are then sign-adjusted according to the MSBs of the decomposed indices and summed to reconstruct the ternary lookup result. Afterward, the result may require an additional right shift for tail-group compensation before being accumulated with the incoming systolic partial sum. Compared with a conventional single-read-and-accumulate path, these extra stages introduce non-trivial area and power overheads in the PE datapath. To mitigate them, FLUTE applies four microarchitectural optimizations: sparsity-aware shared shifters, deferred two's-complement addition, intra-tile spatial accumulation, and fused preprocessing/alignment.

Sparsity-Aware Shared Shifters. As discussed in Section IV-B, tail-group computations require an additional right shift on the LUT readout to compensate for the exponent gap between $e_{unified}$ and e_{tail}. A straightforward implementation would assign one shifter to each RAC, which is prohibitively inefficient in area and power. Our profiling shows that only about 10% of the localized computations require this shift. Given 16 RACs per PE, this corresponds to an expected simultaneous shift demand of only about 1.6, suggesting that a small shared pool is sufficient in practice. We therefore provision 2 shared shifters per PE, with a lightweight arbitration mechanism routing the few shift-demanding operands to the shared pool, as illustrated in Fig. 8(a). Experimental results show that this design preserves throughput with negligible pipeline stalls while noticeably reducing area overhead.

Deferred Two's Complement Addition. As discussed in Section V-B, FLUTE combines BEU with symmetry-based LUT compression, where the MSB of each decomposed index indicates whether the retrieved LUT value should be negated. A straightforward implementation would perform this negation locally in every PE as two's-complement conversion, i.e., $-x = \sim x + 1$. However, this requires a dedicated high-precision adder (20 bits for FP16 activation) in each PE solely for the "+1" operation, incurring non-trivial overhead. To avoid this cost, we exploit the distributive property of accumulation, i.e., $\sum(\sim x+1) = \sum(\sim x) + \sum 1$, and decouple the addition from the compensation term. Each PE therefore performs only the lightweight bitwise inversion locally and accumulates the inverted LUT outputs, while a lightweight 6-bit MSB counter tracks the number of active sign-inversion events across the accumulation dimension. The corresponding total "+1" compensation is then deferred to the INT2FP unit at the end of the PE row, where it is added once to the final accumulated sum. This rearrangement eliminates redundant high-precision adders from the dense PE array without affecting correctness.

Intra-Tile Spatial Accumulation. In the baseline PE, each RAC maintains a high-bitwidth partial-sum (psum) register, which incurs energy and area overhead. To reduce this cost, FLUTE groups four horizontal PEs into one tile and first combines their outputs through a lightweight combinational adder tree. The incoming psum from the left tile is buffered for one cycle in a shared psum register, where it is added to the aggregated output of the four local PEs. The resulting psum is then propagated to the right tile, forming a systolic accumulation pattern across tiles in the same row. By replacing per-PE high-bitwidth psum registers with one shared tile-level register, this design substantially reduces register overhead and improves both area and energy efficiency. We choose a tile size of four as a timing-area trade-off, since larger tiles would require deeper adder trees and a longer critical path.

Fused Preprocessing and Alignment. In FLUTE, LUT construction first multiplies each FP activation by its column-wise FP16 scaling factor to form an intermediate product, and then generates LUT entries by adding or subtracting four consecutive products. However, for efficient accumulation in the subsequent PE pipeline, these LUT entries should be represented in integer form rather than standard floating-point form. Therefore, all LUT entries must be aligned to the largest exponent and converted into integer form by right-shifting

the mantissas accordingly. A straightforward implementation would first compute all products and LUT entries in FP and then perform exponent alignment, but this would incur substantial overhead due to full FP multiplication and addition. To avoid this cost, we propose a fused preprocessing module, as shown in Fig. 8(b). After mantissa multiplication, exponent addition, and normalization for Act and Scale, the module skips the remaining FP post-processing. Instead, it extracts the maximum from the summed exponents and right-shifts the corresponding mantissa products accordingly, and directly obtains the aligned integers. These integers are then fed to the LUT generator, where integer additions produce the final LUT entries. Through this fused design, we eliminate unnecessary intermediate steps and reduce downstream hardware overhead while preserving numerical correctness.

D. Multi-Granularity Design Space Exploration

As discussed in Section III-B, final energy efficiency is significantly affected by multi-granularity trade-offs. We therefore perform design space exploration to jointly optimize coarse-grained dataflow selection and fine-grained LUT fan-out parallelism. Specifically, we compare different dataflows and K_{RAC} values under a fixed peak throughput to identify the most energy-efficient design point.

The throughput budget is parameterized as $m_{tile} \times k_{tile} \times$ (1bit $\times$ FP16). In our evaluation, we set $m_{tile} = k_{tile} = 64$. For a given LUT fan-out K_{RAC}, the array contains m_{tile}/K_{RAC} rows of PE tiles and $k_{tile}/16$ columns, where each PE tile contains four horizontal PEs and each PE maintains one LUT built from 4 consecutive activations.

We consider static leakage, SRAM read/write, pre-PE energy (LUT construction), LUT lookup energy in PE, other PE-operation energy (Addition, Shifting), and post-PE energy (INT2FP, Accumulation). Among these terms, static leakage, other PE-operation energy, and post-PE energy are obtained for each design through post-synthesis power characterization. Meanwhile, the energy of SRAM read/write, pre-PE operations, and LUT lookup in PEs depends on their operation counts. We therefore derive an analytical model for a GEMM workload with weight matrix $W \in \mathbb{R}^{M\times K}$ and activation matrix $A \in \mathbb{R}^{K\times N}$, parameterized by the dataflow, K_{RAC}, tiling parameters (m_{tile}, k_{tile}), and quantization group size G. Output SRAM writes are omitted below because they are identical across different dataflows and K_{RAC} values.

Under WS, an $m_{tile} \times k_{tile}$ binary weight tile is first preloaded into the RACs of the PE array, while LUT values are streamed through the array. Since the LUT values depend on the current activations, every bit plane of every weight tile incurs one round of activation reads and LUT construction. Moreover, because each LUT is constructed from 4 activation values, both LUT-build and LUT-lookup counts include a division factor of 4. The resulting operational counts are as follows:

$$\begin{aligned}
&W_{SRAM}\ \text{Read} = M \cdot K \cdot w_{bits},\\
&A_{SRAM}\ \text{Read} = K \cdot N \cdot (M \cdot w_{bits}/m_{tile}) \cdot a_{bits},\\
&LUT_{build}\ \text{Count} = K \cdot N \cdot (M \cdot w_{bits}/m_{tile})/4,\\
&LUT_{lookup}\ \text{Count} = M \cdot N \cdot K \cdot w_{bits}/4.
\end{aligned} \tag{5}$$

In contrast, under IS, the LUTs generated from activation sub-vectors are kept stationary and reused across multiple weight queries. Each PE row holds the LUTs generated from one activation sub-vector, with different PE rows corresponding to different columns of activations, so the PE array can hold up to m_{tile}/K_{RAC} such sub-vectors at a time. During computation, $k_{tile} \times K_{RAC}$ weight-bit indices are injected into a PE row per cycle to query these stationary LUTs, while the indices propagate vertically across PE rows and partial sums propagate horizontally. Consequently, one LUT construction can be amortized over at most G weight rows within the same quantization group, because the LUT is constructed by the activation values and the group-specific scaling factors. This reduces both activation reads and LUT builds. The weight-SRAM-read term has two cases. If $N \le m_{tile}/K_{RAC}$, all activation sub-vectors fit in the array at once, and each weight bit is fetched only once. Otherwise, the workload must be partitioned into tiles along the N dimension, so the same weight tile is fetched repeatedly $(N \cdot K_{RAC}/m_{tile})$ times. The resulting operational counts are:

$$\begin{aligned}
&W_{SRAM}\ \text{Read} = \begin{cases} M \cdot K \cdot w_{bits}, & \text{if } N \le m_{tile}/K_{RAC},\\ M \cdot K \cdot (N \cdot K_{RAC}/m_{tile}) \cdot w_{bits}, & \text{else.}\end{cases}\\
&A_{SRAM}\ \text{Read} = K \cdot N \cdot M/G \cdot a_{bits},\\
&LUT_{build}\ \text{Count} = K \cdot N \cdot M/G/4,\\
&LUT_{lookup}\ \text{Count} = M \cdot N \cdot K \cdot w_{bits}/4.
\end{aligned} \tag{6}$$

This analysis highlights both the coarse-grained and fine-grained trade-offs. At the coarse-grained level, IS may increase weight SRAM accesses, but it significantly reduces activation fetches and, more importantly, LUT construction frequency. At the fine-grained level, increasing K_{RAC} improves LUT sharing but also raises hardware and SRAM overhead. As illustrated in Fig. 9, combining the energy from post-synthesis and the analytical model, the overall energy of IS is consistently lower than WS across all evaluated batch sizes, and the best energy efficiency is achieved at $K_{RAC} = 16$, which we select as the final hardware configuration.

VI. Evaluation

A. Experimental Setup

Baselines. We compare FLUTE with representative baselines on both the algorithm and hardware sides. Algorithmic baselines include standard FP, INT, and BFP quantization, as well as two SOTA BFP methods, BBAL [15] and BiE [14]. Hardware baselines include three MAC-based accelerators, namely FPE (full-precision FP MAC) [26], FIGNA [26], and AxCore [27], and one LUT-based accelerator, FIGLUT [24]. These comparisons are conducted across nine weight-activation combinations (4/6/8-bit weights $\times$ FP16/BF16/FP32 activations).

TABLE II
GEOMETRIC AVERAGE PERPLEXITY (PPL) ON GENERATIVE TASKS. THE EVALUATION SPANS 3-BIT, 4-BIT, AND 6-BIT TARGET PRECISIONS.

Models	FP16	3-bit						4-bit						6-bit					
		FP4	INT3	BFP3	BBAL	BiE	**Ours**	FP6	INT4	BFP4	BBAL	BiE	**Ours**	FP8	INT6	BFP6	BBAL	BiE	**Ours**
OPT-6.7B	11.88	12.76	1146	2068	8102	8815	**12.85**	11.94	12.71	13.60	14.98	174.6	**12.09**	11.89	11.90	11.91	11.92	11.93	**11.88**
OPT-13B	11.16	12.00	531.1	706.2	8528	3458	**12.34**	11.20	11.72	12.77	13.68	20.92	**11.29**	11.17	11.19	11.20	11.20	11.21	**11.16**
OPT-30B	10.57	11.15	760.8	1071	6988	3587	**11.28**	10.60	11.02	12.05	12.66	35.39	**10.67**	10.57	**10.57**	10.63	10.59	10.61	**10.57**
LLaMA-2-7B	9.24	11.14	22.59	25.21	nan	9542	**14.02**	9.39	10.32	10.21	10.78	17.08	**9.54**	9.23	9.31	9.39	9.31	9.29	**9.27**
LLaMA-2-13B	9.68	10.63	13.35	14.79	2398	7840	**11.23**	9.80	10.00	10.11	11.04	10.93	**9.70**	9.69	9.69	9.73	9.71	9.69	**9.67**
LLaMA-2-70B	6.58	6.87	8.09	10.96	215.1	2281	**7.54**	6.59	6.79	6.99	7.65	7.75	**6.67**	6.58	6.58	6.62	6.60	6.61	**6.57**
Qwen3-8B	12.93	14.44	31.00	96.38	nan	4211	**16.71**	13.18	13.74	14.86	15.59	20.57	**13.55**	12.91	12.92	12.92	12.97	13.03	**12.82**
Qwen3-30B-A3B	12.02	13.70	141.7	592.6	nan	5516	**17.59**	12.15	12.98	18.51	28.74	176.2	**12.55**	12.10	12.06	12.07	12.11	12.08	**11.93**

TABLE III
ZERO-SHOT ACCURACY ON DOWNSTREAM REASONING TASKS ACROSS DIVERSE LLMS.

Bit	Tasks	Qwen3-8B							OPT-13B							OPT-30B						
		FP16	FP	INT	BFP	BBAL	BiE	**Ours**	FP16	FP	INT	BFP	BBAL	BiE	**Ours**	FP16	FP	INT	BFP	BBAL	BiE	**Ours**
3-bit	Avg	0.723	0.702	0.501	0.371	0.317	0.322	**0.623**	0.629	0.606	0.349	0.340	0.307	0.307	**0.621**	0.658	0.640	0.377	0.368	0.313	0.305	**0.641**
4-bit	ARC-C	0.565	0.573	0.538	0.509	0.514	0.433	**0.562**	0.354	0.349	0.354	0.340	0.335	0.307	**0.355**	0.385	0.387	0.370	0.346	0.333	0.321	**0.373**
	ARC-E	0.811	0.813	0.768	0.719	0.717	0.679	**0.784**	0.612	0.609	0.610	0.600	0.586	0.529	**0.618**	0.653	0.651	0.636	0.619	0.604	0.578	**0.648**
	BoolQ	0.867	0.865	0.846	0.830	0.831	0.741	**0.858**	0.666	0.671	0.662	0.666	0.664	0.660	**0.691**	0.716	0.709	0.673	0.681	0.691	0.650	**0.729**
	Hella	0.750	0.747	0.730	0.701	0.691	0.636	**0.738**	0.699	0.698	0.683	0.664	0.637	0.588	**0.695**	0.722	0.720	0.710	0.700	0.680	0.573	**0.710**
	Lambada	0.611	0.612	0.556	0.530	0.531	0.386	**0.587**	0.635	0.630	0.610	0.587	0.576	0.462	**0.635**	0.650	0.645	0.614	0.613	0.609	0.408	**0.614**
	PIQA	0.777	0.780	0.760	0.762	0.755	0.733	**0.767**	0.767	0.769	0.761	0.757	0.748	0.733	**0.761**	0.787	0.790	0.782	0.772	0.767	0.748	**0.782**
	Wino	0.680	0.683	0.664	0.639	0.650	0.591	**0.665**	0.668	0.665	0.665	**0.669**	**0.671**	0.638	0.657	0.697	0.693	0.676	0.639	0.652	0.634	**0.676**
	Avg	0.723	0.725	0.694	0.670	0.670	0.600	**0.709**	0.629	0.627	0.621	0.612	0.603	0.560	**0.630**	0.658	0.656	0.637	0.624	0.619	0.559	**0.637**
6-bit	Avg	0.723	0.722	0.718	0.721	0.719	0.715	**0.722**	0.629	0.629	0.628	0.627	0.628	0.629	**0.630**	0.658	0.658	0.658	0.656	0.657	0.655	**0.659**

Bit	Tasks	LLaMA-2-7B							LLaMA-2-13B							LLaMA-2-70B						
		FP16	FP	INT	BFP	BBAL	BiE	**Ours**	FP16	FP	INT	BFP	BBAL	BiE	**Ours**	FP16	FP	INT	BFP	BBAL	BiE	**Ours**
3-bit	Avg	0.701	0.693	0.613	0.567	0.327	0.316	**0.642**	0.729	0.720	0.666	0.624	0.320	0.315	**0.694**	0.777	0.764	0.740	0.680	0.379	0.309	**0.758**
4-bit	ARC-C	0.450	0.451	0.448	**0.454**	0.442	0.422	0.448	0.489	0.485	0.482	**0.486**	0.469	0.449	0.477	0.573	0.566	0.550	**0.561**	0.550	0.521	0.549
	ARC-E	0.739	0.742	0.725	0.712	0.714	0.677	**0.733**	0.765	0.763	0.758	0.761	0.751	0.719	**0.766**	0.796	0.796	0.785	0.795	0.792	0.780	**0.801**
	BoolQ	0.794	0.788	0.776	0.756	0.745	0.742	**0.783**	0.821	0.823	0.818	0.814	0.819	0.793	**0.821**	0.852	0.848	0.840	0.823	0.817	0.811	**0.851**
	Hella	0.762	0.758	0.755	0.749	0.736	0.722	**0.760**	0.796	0.795	**0.789**	0.784	0.766	0.765	0.783	0.844	0.841	**0.835**	0.824	0.809	0.795	0.831
	Lambada	0.688	0.687	0.672	0.647	0.633	0.588	**0.678**	0.706	0.713	0.692	0.691	0.687	0.663	**0.693**	0.748	0.750	0.738	0.738	0.737	0.701	**0.761**
	PIQA	0.788	0.786	0.782	0.780	0.784	0.770	**0.789**	0.802	0.808	0.803	0.796	0.797	0.785	**0.804**	0.826	0.823	0.820	0.819	0.817	0.815	**0.824**
	Wino	0.694	0.695	0.692	0.690	0.679	0.680	**0.684**	0.723	0.718	0.716	0.715	0.708	0.703	**0.734**	0.804	0.811	0.792	0.779	0.779	0.769	**0.800**
	Avg	0.702	0.701	0.693	0.684	0.676	0.657	**0.697**	0.729	0.729	0.723	0.721	0.714	0.697	**0.725**	0.778	0.776	0.765	0.763	0.757	0.742	**0.774**
6-bit	Avg	0.702	0.700	0.700	0.701	0.700	0.698	**0.701**	0.729	0.729	0.729	0.728	0.728	0.729	**0.731**	0.778	0.777	0.776	0.774	0.775	0.776	**0.777**

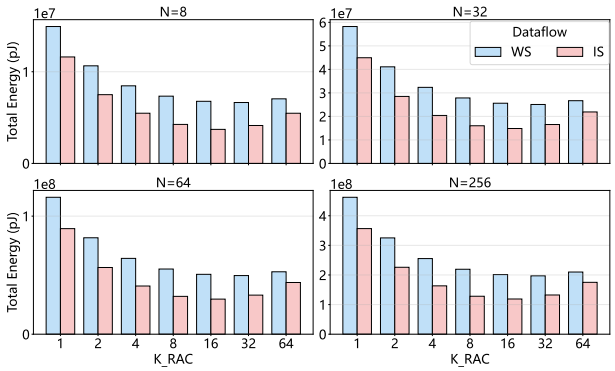


Fig. 9. Energy comparison across dataflows and K_{RAC}.

Accuracy Evaluation Setup. We evaluate algorithmic accuracy on OPT (6.7B, 13B, 30B), LLaMA-2 (7B, 13B, 70B), and Qwen3 (8B, 30B-A3B) [50]–[52]. We measure perplexity (PPL) on WikiText-2 [53], PTB [54], and C4 [55], and evaluate zero-shot reasoning on ARC-Easy, ARC-Challenge [56], BoolQ [57], HellaSwag [58], LAMBADA [59], PIQA [60], and WinoGrande [61] via the lm-evaluation-harness [62] framework. We convert the original FP4, FP6, and FP8 quantized formats into 3-, 4-, and 6-bit representations with 1 sign bit and 2, 3, and 5 mantissa bits, respectively. We maintain a uniform quantization group size of 128.

Hardware Evaluation Setup. We implement FLUTE and all hardware baselines in Verilog HDL and synthesize them using Synopsys Design Compiler in a 28nm TSMC process at 1GHz. On-chip SRAM is modeled using the CACTI simulator [63]. For off-chip memory, we use HBM2 DRAM with a bandwidth of 256 GB/s, and obtain the DRAM access energy following FIGNA [26]. End-to-end performance is obtained using a custom cycle-accurate simulator validated against the HDL design. To ensure fairness, all architectures are normalized to the same peak throughput (TOPs/s). For 4-bit weights, the FIGLUT and FLUTE arrays are configured as $2 \times 16 \times 4$ and $4 \times 16 \times 4$, respectively, while the MAC-based baselines use a 64×64 systolic array. In our final FLUTE configuration, $4 \times 16 \times 4$ denotes four PE arrays; each array contains a 4×4 tile mesh, with four PEs per tile and 16 RAC units sharing one LUT in each PE. For 6-bit and 8-bit settings, FIGLUT and FLUTE arrays remain unchanged due to their support for bit-serial execution, while MAC-based baselines are scaled accordingly. All hardware performance and energy results are measured in the decode stage, with batch size 64, prefill length 512, and decode length 128.

B. Accuracy Evaluation

Perplexity on Generative Tasks. Table II summarizes the geometric average perplexity (PPL) across diverse LLMs, including both dense models and MoE model. Our low-bit formats are derived from the original FP8, FP6, and FP4 formats (FP8→6-bit, FP6→4-bit, FP4→3-bit). First, our method maintains accuracy comparable to pre-conversion formats; for instance, our 3-bit OPT-6.7B yields a 12.85 PPL, closely tracking its FP4 baseline (12.76). Second, under identical

TABLE IV
GEOMEAN PPL UNDER 4-BIT WEIGHT QUANTIZATION.

Model	FP16	INT4	AWQ	MXFP4	NVFP4	FLUTE
LLaMA-2-7B	9.238	10.323	9.969	10.671	10.283	**9.540**
LLaMA-2-13B	9.682	10.003	9.976	11.569	10.178	**9.697**
OPT-13B	10.572	11.024	11.314	11.993	11.602	**10.676**

TABLE V
ACCURACY LOSS DECOMPOSITION ON LLAMA-2-7B. DATA SEP. DENOTES SEPARATION ONLY, CT-HEA INCLUDES PSEUDO-CORE SACRIFICE, AND FLUTE EXEC. DENOTES HARDWARE EXECUTION.

Setting	FP16	FP6	FP4	Data Sep.	CT-HEA	FLUTE Exec.
Avg. Acc.	0.702	0.699	0.694	0.698	0.697	0.697
Δ vs. prev.	–	-0.003	-0.005	+0.004	-0.001	0.000

constraints, we significantly outperform baselines (INT, BFP, BBAL, BiE), especially at 3-bit, where coarse-grained quantization causes severe PPL degradation. Fundamentally, BFP's power-of-2 scaling degrades accuracy. Furthermore, advanced variants (BBAL, BiE) necessitate an extra flag bit, which shrinks the representational space under strict bit-widths. By overcoming these bottlenecks, our method substantially improves low-bit accuracy.

Comparison with PTQ and Emerging FP4 Formats. We then compare FLUTE with AWQ and recent FP4 representations under the 4-bit weight-only setting on LLaMA-2-7B. Table IV reports the perplexity. FLUTE achieves the lowest perplexity among all methods, showing that CT-HEA remains accurate compared with both PTQ-based INT quantization and emerging FP4 formats.

Accuracy Ablation Study. We decompose the accuracy loss on LLaMA-2-7B over downstream tasks, as shown in Table V. The results indicate that most accuracy loss comes from low-bit FP quantization itself. In contrast, CT-HEA introduces only 0.001 additional loss after core/tail separation, mainly from sacrificing sparse Pseudo-Core TEWs. FLUTE execution introduces no further loss.

Zero-Shot Accuracy on Downstream Tasks. Table III extends our evaluation to zero-shot reasoning, providing a detailed 4-bit task breakdown, while reporting only the 3-bit and 6-bit averages due to space constraints. The "FP" column denotes the corresponding pre-conversion formats: FP4, FP6, and FP8 for the 3-bit, 4-bit, and 6-bit settings, respectively. Downstream accuracy follows the same trend as PPL. At 3-bit, conventional baselines suffer severe degradation. On LLaMA-2-7B and OPT-13B, BFP variants (BBAL, BiE) drop to near-random averages, whereas our method maintains accuracies of 0.642 and 0.621. Our 4-bit breakdown further shows this advantage. On challenging tasks like LAMBADA, highly compressed baselines such as BiE drop to 0.386 (Qwen3-8B) and 0.462 (OPT-13B), while our 4-bit configuration delivers 0.587 and 0.635. Overall, across all evaluations, our method surpasses standard INT/BFP schemes and achieves performance comparable to pre-conversion formats. This provides the algorithmic basis for the hardware benefits that follow.

TABLE VI
AREA BREAKDOWN OF FLUTE (IN MM^2).

Act	Pre	LUT-G	BEU	PEs	INT2FP	Acc	Total
FP16	0.025	0.008	0.007	1.203	0.053	0.104	1.400
BF16	0.023	0.006	0.007	1.015	0.051	0.104	1.207
FP32	0.064	0.015	0.007	1.941	0.075	0.104	2.207

C. Performance Comparison

Fig. 11 compares the normalized performance relative to the FPE baseline under FP4, FP6, and FP8 settings. We report both GEMM-level performance and end-to-end decode performance. The GEMM-level workload uses a (4096×4096) weight matrix, while the end-to-end evaluation uses LLaMA-2-7B, including linear layers, attention, and non-linear operations. Under the iso-peak-throughput constraint, MAC-based accelerators, including FPE, FIGNA, and AxCore, compute directly using their native precisions, and their minor performance differences mainly come from internal routing and module-level pipeline design.

At the GEMM level, FIGLUT suffers from floating-point exponent alignment and must process 5, 10, and 19 bit planes for FP4, FP6, and FP8 weights, resulting in only $0.81\times$, $0.61\times$, and $0.43\times$ normalized performance relative to FPE. In contrast, FLUTE maps these weights into compact 3-bit, 4-bit, and 6-bit CT-HEA representations, requiring only 2, 3, and 5 bit-plane iterations and achieving $2.01\times$, $2.02\times$, and $1.61\times$ GEMM-level speedup. In the end-to-end evaluation, FLUTE still delivers $1.84\times$, $1.85\times$, and $1.53\times$ end-to-end speedup, showing that the acceleration of weight-only linear layers effectively translates into system-level benefits.

D. Area Efficiency

For area efficiency, we analyze throughput per unit area (Performance/Area) of the GEMM datapath across the LLaMA-2 family, with results shown in Fig. 10. FPE is normalized to 1.0. The results indicate that FLUTE maintains its advantage across different model sizes, demonstrating its applicability to different GEMM matrix dimensions.

Among the MAC-based baselines, FPE has the lowest efficiency because of its large FP units. FIGNA removes FP multipliers through integer-based computation, but its high-precision accumulators and deep alignment FIFOs still limit area efficiency. AxCore improves upon FPE by approximating FP multiplication with integer additions, but its conversion logic and lookup tables still incur a substantial hardware footprint. Among LUT-based architectures, FIGLUT achieves reasonable efficiency at low precision, but suffers from severe bit-plane explosion at higher precisions. In contrast, FLUTE removes this bottleneck by mapping FP weights to compact low-bit formats and applying coordinated optimizations at multiple levels in hardware. Overall, FLUTE outperforms all baselines across the evaluated precision settings, achieving average improvements in area efficiency of $6.32\times$, $2.97\times$, $2.66\times$, and $1.62\times$ over FPE, FIGNA, FIGLUT, and AxCore, respectively.

Furthermore, Table VI details the GEMM datapath area breakdown of FLUTE across diverse activation formats.

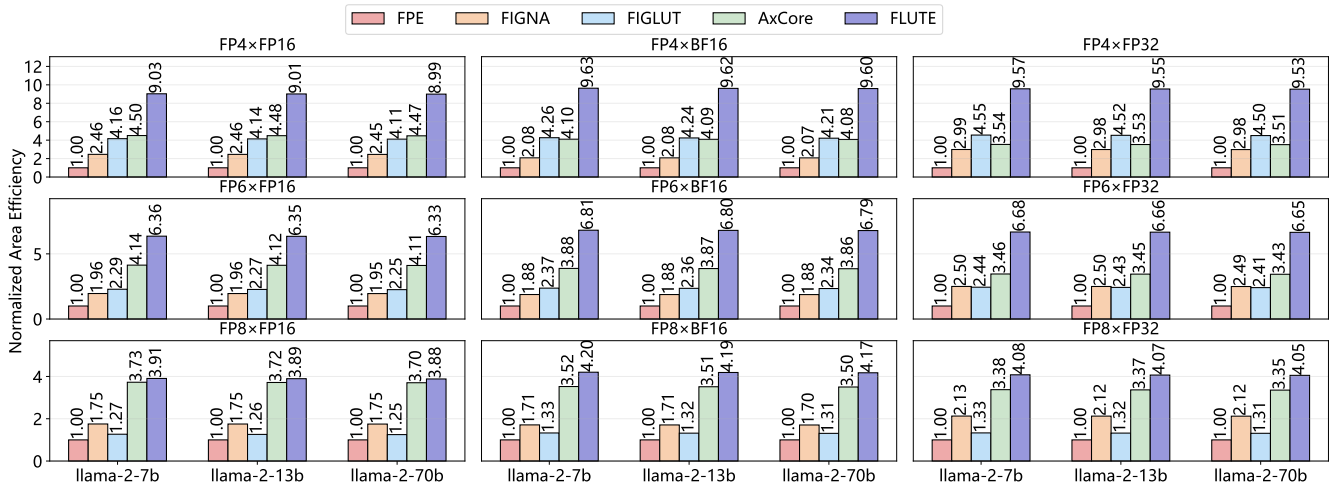


Fig. 10. Normalized area efficiency (Performance/Area) of the evaluated architectures under diverse weight precisions.

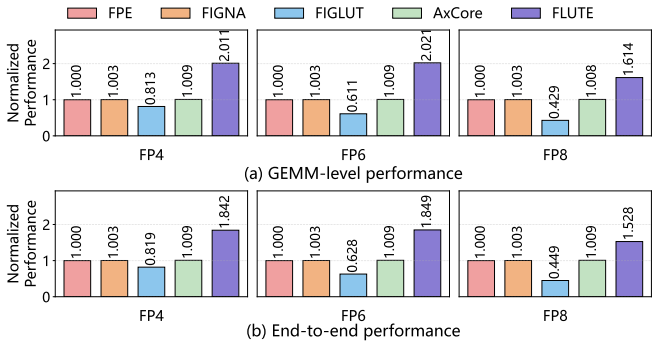


Fig. 11. Normalized performance of various architectures.

Across all configurations, the PEs dominate the total hardware footprint. The overhead of the proposed BEU remains negligible (0.007 mm^2), validating its lightweight design for index decomposition.

E. Energy Efficiency

For energy evaluation, we analyze two aspects: compute-only energy efficiency across diverse LLMs (Fig. 12) and the system-level energy breakdown for a GEMM workload (Fig. 13).

Compute-only efficiency is normalized to the FPE baseline. FIGNA and AxCore achieve 1.32–1.78× and 2.04–3.02× efficiency, respectively. FIGLUT suffers from severe bit-plane explosion. While it reaches 3.19× under FP4, bit-plane explosion under FP8 reduces its efficiency to 0.90×. In contrast, by processing compact bit-planes with the lightweight BEU, FLUTE minimizes dynamic switching, achieving up to 8.51× compute efficiency (under FP4×BF16) and sustaining 3.99× under FP8×FP32. Overall, FLUTE achieves average compute energy-efficiency improvements of 6.01×, 4.02×, 3.35×, and 2.38× over FPE, FIGNA, FIGLUT, and AxCore, respectively.

For system-level analysis, Fig. 13 shows the normalized energy breakdown for a GEMM workload. First, FLUTE greatly reduces the compute energy that dominates the FPE baseline. Second, its faster execution (Section VI-C) shortens execution time and proportionally reduces static leakage. FLUTE reduces total energy consumption to 25% (under FP4) and 39% (under FP8) of the FPE baseline.

TABLE VII
PE OPTIMIZATION ABLATION: AREA (μm^2) AND POWER (MW), NORMALIZED TO THE 81-ENTRY PE.

Design	Area	Area Eff.	Power	Power Eff.
81-entry PE	16338	1.00x	4.085	1.00x
8-entry PE	6455	2.53x	1.706	2.39x
+ Shared shifters	5440	3.00x	1.606	2.54x
+ Deferred "+1"	5203	3.14x	1.578	2.59x
+ Tiling	4646	3.52x	1.087	3.76x

F. Energy Breakdown of Dataflow and K_{RAC}

Fig. 14 analyzes the energy breakdown of WS and IS dataflows for K_{RAC} ranging from 1 to 64 at batch size 64. Two trade-offs emerge. First, while WS spends over 20% of its total energy on preprocessing, this overhead is reduced under the IS dataflow. Second, scaling K_{RAC} incurs a memory-access penalty: in the IS dataflow, larger K_{RAC} increases SRAM access energy due to more frequent weight-buffer fetches. Balancing static power and SRAM overhead, IS with $K_{RAC} = 16$ minimizes total energy consumption and is selected as the final hardware configuration.

G. Ablation Study of PE Optimizations

To isolate the efficacy of our proposed microarchitectural techniques, we conduct a detailed ablation study on a single PE. Table VII tracks the physical area and power metrics as each optimization introduced in Section V is incrementally applied. The largest reduction comes from replacing the naive 81-entry lookup mechanism with the BEU-driven 8-entry PE. This shift reduces area by over 60% and more than doubles power efficiency (2.39×). Following this, integrating sparsity-aware shared shifters and deferring the high-bit two's complement "+1" addition further reduces the combinational logic cost. Finally, intra-tile spatial accumulation (Tiling) reduces sequential register overheads, yielding a compact area of 4646 μm^2. Collectively, these optimizations improve area efficiency by 3.52× and power efficiency by 3.76× over the baseline.

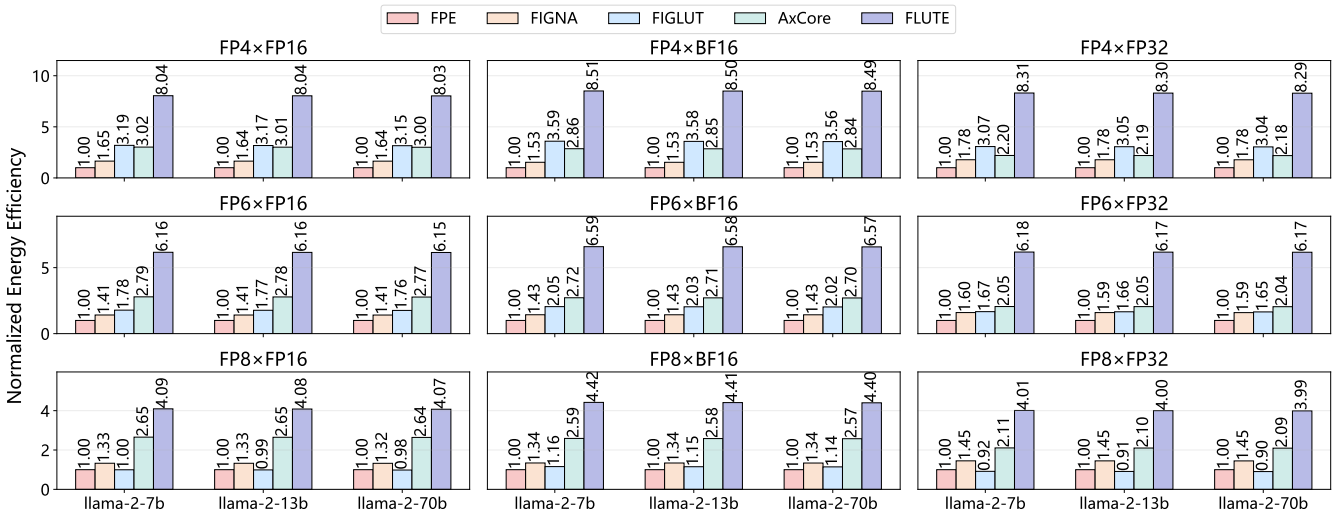


Fig. 12. Normalized compute energy efficiency across diverse models and precision configurations.

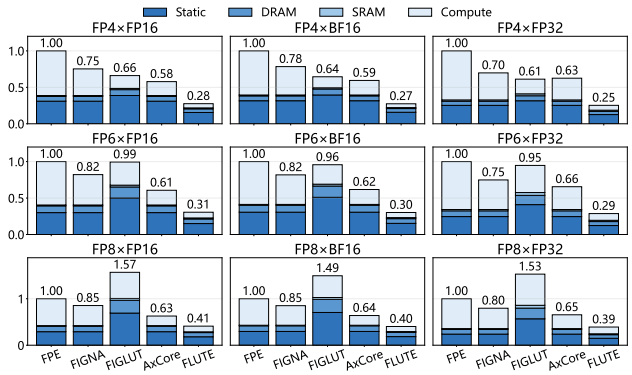


Fig. 13. System-level energy breakdown normalized to the FPE baseline when executing GEMM.

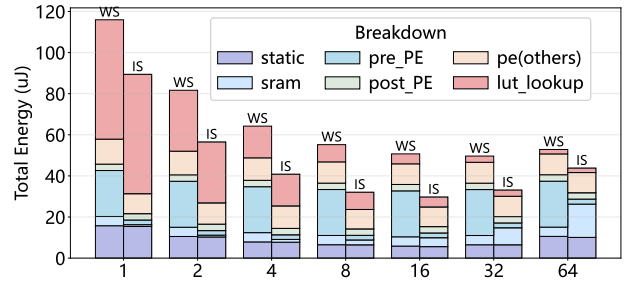


Fig. 14. Energy breakdown comparison between WS and IS dataflows across varying K_{RAC} values (batch size = 64).

H. System-level Benefits Study

To understand the source of efficiency gains, Fig. 15 shows a cumulative breakdown from the baseline to the full FLUTE design. We progressively add CT-HEA, BEU, microarchitectural optimizations, and dataflow optimization, and report normalized area and energy efficiency under FP4, FP6, and FP8 weights with FP16 activations.

CT-HEA converts FP weights into a compact LUT-friendly representation, reducing bit-plane iterations and LUT accesses. BEU maps this representation to an efficient LUT datapath, providing substantial area- and energy-efficiency gains. Microarchitectural optimizations further improve PE efficiency.

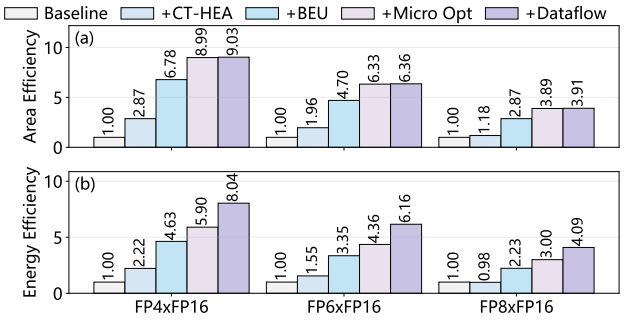


Fig. 15. System-level benefit breakdown.

Dataflow optimization brings limited area-efficiency improvement but noticeably improves energy efficiency by increasing LUT reuse and reducing activation/LUT movement. Overall, FLUTE's gains come from the cumulative effect of representation, datapath, microarchitecture, and dataflow optimizations.

VII. Conclusion

In this paper, we propose FLUTE, an algorithm–hardware co-design accelerator for efficient LUT-based inference on FP-quantized LLMs. We identify an exponent clustering effect in FP-quantized LLM weights. Guided by this insight, FLUTE introduces CT-HEA, which enables hierarchical core-tail alignment in a LUT-friendly representation. On the hardware side, FLUTE further introduces the BEU to decode the ternary bit-planes induced by CT-HEA, and improves efficiency through PE-level microarchitectural co-optimizations and multi-granularity exploration. Extensive experiments demonstrate that FLUTE significantly outperforms the evaluated baselines in both algorithmic accuracy and hardware efficiency.

VIII. Acknowledgments

This work was supported in part by the Brain Science and Brain-like Intelligence Technology-National Science and Technology Major Project (No. 2021ZD0201504) and Beijing Natural Science Foundation (No.4254088).

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

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

[3] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.

[4] M. R. Costa-Jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard *et al.*, "No language left behind: Scaling human-centered machine translation," *arXiv preprint arXiv:2207.04672*, 2022.

[5] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language models for dialog applications," *arXiv preprint arXiv:2201.08239*, 2022.

[6] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[7] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[8] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International conference on machine learning*. PMLR, 2023, pp. 38 087–38 099.

[9] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of machine learning and systems*, vol. 6, pp. 87–100, 2024.

[10] W. Huang, X. Zheng, X. Ma, H. Qin, C. Lv, H. Chen, J. Luo, X. Qi, X. Liu, and M. Magno, "An empirical study of LLaMA3 quantization: from LLMs to MLLMs," *Visual Intelligence*, vol. 2, no. 1, Dec. 2024. [Online]. Available: https://doi.org/10.1007/s44267-024-00070-x

[11] X. Zheng, Y. Li, H. Chu, Y. Feng, X. Ma, Z. Wang, J. Luo, J. Guo, H. Qin, M. Magno, and X. Liu, "An empirical study of Qwen3 quantization," *Visual Intelligence*, vol. 4, no. 1, Apr. 2026. [Online]. Available: https://doi.org/10.1007/s44267-026-00114-4

[12] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[13] H. Shen, N. Mellempudi, X. He, Q. Gao, C. Wang, and M. Wang, "Efficient post-training quantization with fp8 formats," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 483–498, 2024.

[14] L. Zou, W. Zhao, S. Yin, C. Bai, Q. Sun, and B. Yu, "Bie: Bi-exponent block floating-point for large language models quantization," in *Forty-first International Conference on Machine Learning*, 2024.

[15] X. Han, Y. Cheng, J. Wang, J. Lu, H. Wang, X. Zhang, N. Xu, D. Yang, and Z. Jiang, "Bbal: A bidirectional block floating point-based quantisation accelerator for large language models," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.

[16] J. Lee, J. Park, J. Kim, Y. Kim, J. Oh, J. Oh, and J. Choi, "Amxfp4: Taming activation outliers with asymmetric microscaling floating-point for 4-bit llm inference," in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025, pp. 14 993–15 013.

[17] X. Wu, H. Xia, S. Youn, Z. Zheng, S. Chen, A. Bakhtiari, M. Wyatt, R. Y. Aminabadi, Y. He, O. Ruwase *et al.*, "Zeroquant (4+ 2): Redefining llms quantization with a new fp6-centric strategy for diverse generative tasks," *arXiv preprint arXiv:2312.08583*, 2023.

[18] G. Park, B. Park, M. Kim, S. Lee, J. Kim, B. Kwon, S. J. Kwon, B. Kim, Y. Lee, and D. Lee, "Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models," *arXiv preprint arXiv:2206.09557*, 2022.

[19] D. C. Ganji, S. Ashfaq, E. Saboori, S. Sah, S. Mitra, M. Askarihemmat, A. Hoffman, A. Hassanien, and M. Leonardon, "Deepgemm: Accelerated ultra low-precision inference on cpu architectures using lookup tables," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 4656–4664.

[20] G. Li, S. Ye, C. Chen, Y. Wang, F. Yang, T. Cao, C. Liu, M. M. S. Aly, and M. Yang, "Lut-dla: Lookup table as efficient extreme low-bit deep learning accelerator," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 671–684.

[21] J. Wei, S. Cao, T. Cao, L. Ma, L. Wang, Y. Zhang, and M. Yang, "T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge," in *Proceedings of the Twentieth European Conference on Computer Systems*, 2025, pp. 278–292.

[22] Y. Chen, Y. Zhao, Y. Hao, Y. Wen, Y. Dai, X. Li, Y. Liu, R. Zhang, M. Zou, X. Song *et al.*, "Cambricon-c: Efficient 4-bit matrix unit via primitivization," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 538–550.

[23] Z. Mo, L. Wang, J. Wei, Z. Zeng, S. Cao, L. Ma, N. Jing, T. Cao, J. Xue, F. Yang *et al.*, "Lut tensor core: A software-hardware co-design for lut-based low-bit llm inference," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 514–528.

[24] G. Park, H. Kwon, J. Kim, J. Bae, B. Park, D. Lee, and Y. Lee, "Figlut: An energy-efficient accelerator design for fp-int gemm using look-up tables," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 1098–1111.

[25] J. H. Wilkinson, *Rounding errors in algebraic processes*. SIAM, 2023.

[26] J. Jang, Y. Kim, J. Lee, and J.-J. Kim, "Figna: Integer unit-based accelerator design for fp-int gemm preserving numerical accuracy," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 760–773.

[27] J. Zou, Y. Chen, X. Chen, C. Xu, and X. Chen, "Axcore: A quantization-aware approximate gemm unit for llm inference," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, pp. 839–853.

[28] 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.

[29] B. Zhang and R. Sennrich, "Root mean square layer normalization," *Advances in neural information processing systems*, vol. 32, 2019.

[30] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[31] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, Y. Dong, and Y. Wang, "Flashdecoding++: Faster large language model inference on gpus," *arXiv preprint arXiv:2311.01282*, 2023.

[32] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[33] 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 *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 1029–1042.

[34] J. Lee, W. Lee, and J. Sim, "Tender: Accelerating large language models via tensor decomposition and runtime requantization," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1048–1062.

[35] A. Ramachandran, S. Kundu, and T. Krishna, "Microscopiq: Accelerating foundational models through outlier-aware microscaling quantization," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1193–1209.

[36] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

[37] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *Advances in neural information processing systems*, vol. 35, pp. 30 318–30 332, 2022.

[38] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, and D. Alistarh, "Spqr: A sparse-quantized representation for near-lossless llm weight compression," *arXiv preprint arXiv:2306.03078*, 2023.

[39] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in neural information processing systems*, vol. 36, pp. 10 088–10 115, 2023.

[40] IEEE, *IEEE Standard for Floating-Point Arithmetic*, IEEE Std. IEEE Std 754-2019, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8766229

[41] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[42] A. H. Zadeh, M. Mahmoud, A. Abdelhadi, and A. Moshovos, "Mokey: Enabling narrow fixed-point inference for out-of-the-box floating-point transformer models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 888–901.

[43] H. Xia, Z. Zheng, X. Wu, S. Chen, Z. Yao, S. Youn, A. Bakhtiari, M. Wyatt, D. Zhuang, Z. Zhou *et al.*, "Fp6-llm: Efficiently serving large language models through fp6-centric algorithm-system co-design," *arXiv preprint arXiv:2401.14112*, 2024.

[44] B. Darvish Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar *et al.*, "With shared microexponents, a little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[45] B. D. Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner *et al.*, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point." in *NeurIPS*, 2020.

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

[47] C. Si, J. Jiang, and W. Shen, "Unveiling the mystery of weight in large foundation models: Gaussian distribution never fades," *arXiv preprint arXiv:2501.10661*, 2025.

[48] D. C. Montgomery and G. C. Runger, *Applied statistics and probability for engineers*. John wiley & sons, 2010.

[49] F. Pukelsheim, "The three sigma rule," *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994.

[50] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

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

[52] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[53] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[54] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[55] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[56] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[57] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," in *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, 2019, pp. 2924–2936.

[58] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" in *Proceedings of the 57th annual meeting of the association for computational linguistics*, 2019, pp. 4791–4800.

[59] D. Paperno, G. Kruszewski, A. Lazaridou, N.-Q. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández, "The lambada dataset: Word prediction requiring a broad discourse context," in *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, 2016, pp. 1525–1534.

[60] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[61] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[62] L. Gao, J. Tow, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, K. McDonell, N. Muennighoff *et al.*, "A framework for few-shot language model evaluation," *Zenodo*, 2021.

[63] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 3–14.