

FIGFLEX: A Mixed-Precision FP-INT GEMM Accelerator Co-Designed with Low-Bit LLM Quantization

Faraz Tahmasebi, Haocheng Xu, Andrew Trauger, Yomal Perera, Zhenyu Tang, Nael Chowdhury, and Sitao Huang
University of California, Irvine, Irvine, CA, USA
{tahmasef, haochx5, atrauger, yperera, zhenyut3, naelc, sitaoh}@uci.edu

Abstract—Weight-only quantization keeps LLM activations in floating point and stores weights as low-bit integers, turning the dominant workload into floating-point-integer (FP-INT) general matrix multiplication. Recent quantizers reach a 2–3-bit average with mixed precision, but hardware has not kept pace: GPUs dequantize weights to FP16 before the tensor-core multiply-accumulate, and bit-parallel FP-INT accelerators run at a single weight precision, so fewer bits save memory but not computation.

In this paper, we present FIGFLEX, an algorithm–hardware co-designed mixed-precision accelerator whose bit-parallel datapath scales throughput as precision falls. FIGFLEX contributes (1) a shift-add, radix-4 reformulation of FP-INT multiplication that fuses the multiply-accumulate into a unified, precision-flexible partial-product reduction and roughly halves the partial products, together with a novel INT3 optimization that reclaims the idle datapath radix-4 leaves at that precision; and (2) a mixed-precision quantizer that assigns each channel group its bit-width according to the error GPTQ cannot compensate, with a per-input-channel column scaling whose power-of-two normalization keeps the scaled activations out of the FP16 subnormal range, so that the quantized model maps directly onto FIGFLEX’s datapath with the same accuracy as on a GPU. On LLaMA-2-7B, FIGFLEX attains 5.85 and 8.77 WikiText-2 perplexity at 3- and 2-bit average precision, lower than GPTQ, AWQ, SLIM-LLM, AMQ, and QuIP# (without fine-tuning), while delivering $1.56\times$ and $1.42\times$ higher whole-chip throughput per area on the linear GEMMs of a 3-bit mixed-precision model than AxCore [60] and FIGLUT [40], respectively.

I. Introduction

Large language models (LLMs) have become foundational to modern natural language processing, and their capability scales with size: state-of-the-art models hold tens to hundreds of billions of parameters, and a 175B-parameter model alone occupies roughly 350 GB in FP16, which is far beyond the memory of a single accelerator [40, 58, 60]. In inference, the linear layers of the transformer dominate both the parameter count and the runtime, and their core operation is general matrix multiplication (GEMM). Executed in floating point (FP16/BF16), these GEMMs are both memory- and compute-bound and are the primary obstacle to efficient LLM serving.

Weight-only quantization has become the accuracy-preserving standard for compressing LLMs. Because weights vastly outweigh activations in memory while activations carry accuracy-critical outliers, compressing weights to low-bit

integers and keeping activations in floating point preserves quality better than quantizing both [10, 32]. This turns the dominant workload into a floating-point-integer (FP-INT) GEMM between FP16 activations and low-bit integer weights.

While INT4 is now an established quality-preserving precision [10, 32, 35], the frontier has moved lower. Uniform methods push weights to 2–3 bits with incoherence processing and lattice codebooks [5, 9, 53], and *mixed-precision* methods go further, spending 4 bits only on the small fraction of salient weight groups and 1–2 bits elsewhere to reach a low *average* precision at little quality cost [19, 30]. Mixed precision in particular produces a *per-group bit budget*: the number of bits each weight group actually needs, which varies across the model. This heterogeneity should translate not only into a cheaper memory footprint—which efficient low-bit weight-storage kernels already realize [11, 17, 41, 42]—but also into cheaper computation, turning that compression into a real speedup of the linear GEMMs that dominate LLM runtime.

On the compute side, however, that budget goes unspent. General-purpose GPUs have no native FP16 $\times$ INT datapath, so they *dequantize the weights back to FP16* before the multiply-accumulate and run it on floating-point units [10, 32]; fewer bits therefore cut memory and bandwidth but not computation. State-of-the-art dedicated FP-INT accelerators remove this dequantization by *pre-aligning* activations into integer mantissas [24], but their efficiency is realized for a fixed precision. FIGNA [20] commits to a single bit-width (INT4) and is inefficient below it or for mixed precision. AxCore [60] fixes the weight quantization to the FP4 format across its E/M variants, and approximates the multiplier with a logarithmic-domain addition and an error-compensation term, while keeping the accumulation in floating point—a far larger unit than integer accumulation. For both, a weight in fewer bits costs the same to compute, so lowering the weight bit-width does not raise throughput. Bit-serial architectures have tried to resolve this inefficiency [2, 21, 24], and FIGLUT [40] is the state of the art among them: it supports mixed precision on a single substrate. It splits each weight into binary planes and reads a look-up table of pre-summed activations selected by the weight pattern. However, its work grows with the weight bit-width and the table must precompute every signed

combination of its μ grouped activations (exponential in μ), requiring a large, many-ported flip-flop LUT.

In this paper, we present FIGFLEX, an algorithm–hardware co-designed accelerator whose bit-parallel datapath raises throughput as precision falls. Contributions are summarized as:

- We identify and quantify the gap left by fixed-precision, bit-parallel FP-INT accelerators: their compute cost does not fall as weight bit-width falls, so they cannot exploit the mixed precision that low-bit LLMs now rely on.
- We design a *bit-parallel, precision-flexible* FP-INT processing element built on a radix-4 weight recoding. A dedicated INT3 optimization fuses the two high bits of a weight pair into one shared lane, recovering the throughput a naive radix-4 wastes on odd precision. The element adds no error of its own (the radix-4 recoding is an exact regrouping and the integer accumulation is exact) with exponent alignment, multiple generation, and normalization hoisted to a shared array edge.
- We co-design a mixed-precision quantizer with a dequantization-free epilogue. It assigns each channel group a unique bit-width based on the error that GPTQ’s compensation leaves behind, rather than by a salience score computed before quantization, and it normalizes each input channel with a column scale that is rescaled by a power of two (2^s) so that the scaled activations never fall into FP16 subnormal range the hardware flushes. The quantizer maps directly onto the hardware’s precision scaling.

On a 3-bit mixed-precision model, FIGFLEX delivers $1.56\times$ and $1.42\times$ the whole-chip throughput per area of AxCore [60], the state-of-the-art approximate FP-INT engine, and FIGLUT [40], the state-of-the-art precision-scalable one: unlike AxCore, its throughput follows the weight bit-width, and unlike FIGLUT, it does so bit-parallel. The radix-4 recoding is exact, and with the shared-exponent alignment of the RTL the quantized models reach the same perplexity on FIGFLEX as on a GPU. On LLaMA-2-7B, FIGFLEX attains 5.85 and 8.77 WikiText-2 perplexity at 3- and 2-bit average precision, lower than state-of-the-art low-bit post-training methods (GPTQ [10], AWQ [32], AMQ [30], Slim-LLM [19], and QuIP# [53] without fine-tuning); QuIP#, whose weights an FP-INT array cannot consume directly, keeps an edge in C4 perplexity and zero-shot accuracy at 2 bits.

II. Background and Motivation

A. LLM Computation

An LLM is a stack of decoder blocks with two types of GEMMs: the linear layers (Q/K/V and output projections, feed-forward) multiply activations by static weights, while attention multiplies activations by activations. For hidden size d and context length l , a block’s linear GEMMs cost $\mathcal{O}(d^2)$ MACs per token against attention’s $\mathcal{O}(ld)$, so the linear GEMMs dominate compute, and only they involve the quantized integer weights.

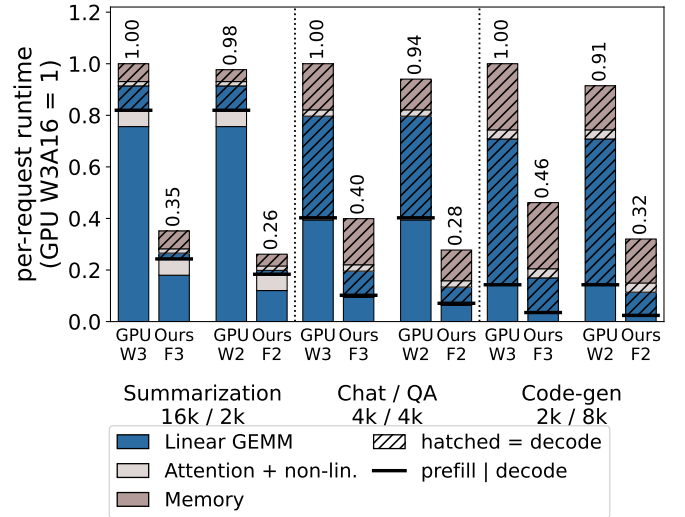


Fig. 1: **The linear GEMM (blue bars) share of the total per-request runtime of LLaMA-3.1-405B in various tasks.** The quantized model (GQA, weights and KV cache at 3 or 2 bits) is deployed on an H200-class GPU; each bar is split into linear GEMM, attention, and memory; solid segments are prefill, hatched are decode. GPU low-bit kernels (W3A16, W2A16; GPU W3A16 = 1 per task) are compared with FIGFLEX at the same bit-width (Ours F3, F2), placed in the GPU’s tensor-core compute area at the same bandwidth: its GEMMs run at the measured compute density (Figure 8) and F3/F2 rates (Table II), and attention and memory are modeled as for the GPU (first-order model).

Figure 1 decomposes a full request of LLaMA-3.1-405B [14] (3-bit weights and KV cache, H200-class GPU, batch 128, vllm [25]) into linear GEMM, attention, and memory. The linear GEMM takes 71–85% of the request time: almost all of prefill, and a substantial share of decode, where streaming the weights and the KV cache becomes equally critical. Lowering the GPU’s weights and KV cache to 2 bits shortens a request by only 2–9%, since the dequantized GEMM still runs at the FP16 rate. This work targets accelerating the linear GEMMs. Our evaluation measures these GEMM layers; the FIGFLEX bars of Figure 1 are a first-order end-to-end estimate built on the measured GEMM rates, with the rest of the request modeled (Section V-E).

B. Weight-Only Quantization

Post-training quantization (PTQ) compresses a trained model without retraining. Weight-only PTQ stores the weights as low-bit integers and keeps the activations in FP16. Weights are quantized per group of 128 input channels with an asymmetric affine code: a p -bit weight is stored as $q = \text{clip}(\lfloor w/s + z \rfloor, 0, 2^p - 1)$ and decoded as $\hat{w} = s(q - z)$, with one scale s and one integer zero-point z per group.

GPTQ [10], the basis of our quantizer, minimizes the layer-wise output error $\|X(W - \hat{W})\|_F^2$ with the calibration Hessian $H = X^\top X$. It quantizes the columns one at a time, most salient first (act-order), and propagates each column’s rounding error to the columns not yet quantized through H^{-1} . Those columns compensate part of the error, and only the remainder reaches the layer output. AWQ [32], OmniQuant [47], and rotation-

based methods [3, 35, 53] reach near-FP16 quality at 4 bits by other means.

Below 4 bits, *uniform* methods rely on incoherence processing and lattice, additive, vector, or trellis codebooks [5, 9, 34, 53, 54], whereas *mixed-precision* methods vary the bit-width across weight groups: Slim-LLM [19] assigns widths by a salience metric and AMQ [30] searches a per-layer allocation. The resulting per-group, non-power-of-two bit-widths are what fixed-precision hardware cannot exploit.

C. FP-INT GEMM Computation

GPUs have no FP16 $\times$ INT datapath. Low-bit kernels such as CUTLASS mixed-input [39], Marlin [11], and FLUTE [17] *dequantize* the weights to FP16 on the fly and issue ordinary tensor-core MACs, so their arithmetic cost is independent of the bit-width and their gain saturates once the GEMM is compute-bound (prefill, or decode at moderate batch). LUT-GEMM [41] and Any-Precision LLM [42] process binary weight planes through lookup tables on the CUDA cores, which helps memory-bound decode but is slower than the tensor-core kernels once compute dominates; the latter are our GPU baseline (Figure 1).

FIGNA [20] is a bit-parallel FP16 $\times$ INT4 unit: a block of activations is *pre-aligned* to its maximum exponent [24], multiplied and accumulated against the integer weights in integer hardware, and converted back to floating point once. **AxCore** [60] is multiplier-free but *approximate*: it replaces the multiply with a logarithmic-domain addition and an error-compensation term, accumulates in a per-PE partial floating-point adder whose normalization is deferred to a shared module, and commits to the FP4 format. Both process every weight at one width, so a weight stored in fewer bits is no cheaper to compute, and lowering the weight bit-width does not raise their throughput.

Bit-serial designs make throughput scale with precision by processing one weight bit-plane per step [2, 21]. **FIGLUT** [40], the state of the art, performs μ MACs per bit-plane with one read of a table that holds all 2^μ signed sums of μ grouped activations. The table and its selection logic grow exponentially in μ and dominate the PE, and a b -bit weight still takes b reads.

Each of these designs trades one property for the other: GPUs gain only memory bandwidth, FIGNA and AxCore are efficient but fixed-precision, and FIGLUT parallelizes its bit-serial computation only at the cost of an exponentially larger datapath. We co-design a *bit-parallel* FP-INT datapath whose throughput scales with the weight bit-width.

III. FIGFLEX Architecture

This section presents FIGFLEX, a bit-parallel FP-INT GEMM accelerator. Its processing element has no multiplier: each weight is recoded into radix-4 digits, each digit selects a precomputed multiple of the activation through a 4:1 multiplexer, and one integer adder tree forms the products and their sum at once (Sections III-A and III-B). The same PE therefore runs INT4, INT3, INT2, and INT1 weights at 8,

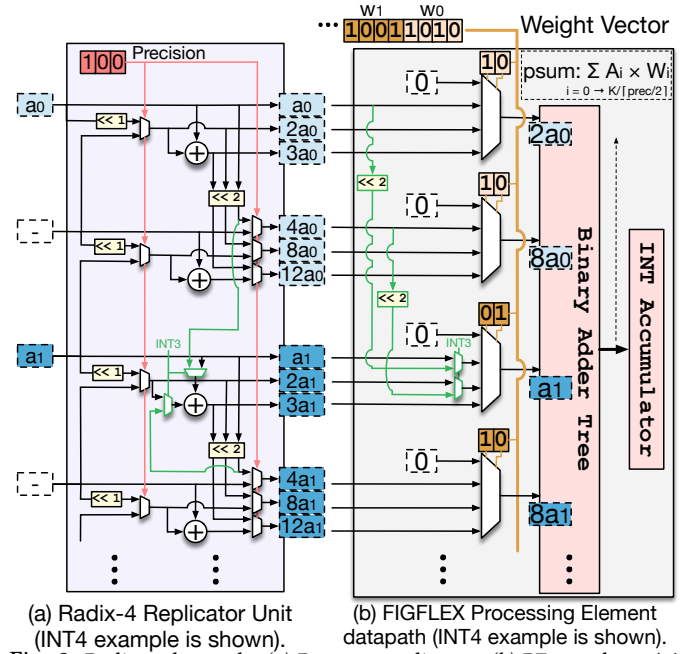


Fig. 2: Radix-4 datapath. (a) Per-row replicator. (b) PE: per-lane 4:1 multiplexers feed one adder tree and an integer accumulator. INT3 fused-lane wiring (Section III-C) in green.

10.67, 16, and 32 MAC/cycle. A high-bit lane fusion removes the half lane that radix-4 leaves idle at INT3 (Section III-C), and Section III-D presents the full accelerator and its mixed-precision datapath.

A. Design Principles

Compute in the integer domain. A GPU dequantizes low-bit weights to FP16 before its tensor cores multiply, so compression saves memory but not arithmetic (Section II-C). FIGFLEX keeps the dot product in the integer domain: as in integer-domain FP-INT units [20, 24], a block of activations is pre-aligned to its maximum exponent with $p+2$ guard bits for p -bit weights, so the dot product becomes an integer reduction that is converted back to floating point once.

Fusing multiplication and accumulation. After pre-alignment, multiplying a significand by an integer weight and accumulating the products are the same primitive: a sum of shifted significands. FIGFLEX performs both in one binary adder tree, with neither a multiplier per product nor a bit-serial look-up [40]. This fusion is what makes the datapath precision-flexible: a weight with fewer bits contributes fewer shifted terms, so it occupies fewer adder-tree inputs, and the freed inputs serve more weights in the same cycle.

B. Radix-4 FP-INT Datapath

While our architecture can be constructed for any activation precision, we elaborate on the design for FP16 $\times$ INT arithmetic. The core operation of weight-only quantized inference is a dot product between an FP16 activation vector and an integer

weight vector,

$$y = \sum_k a_k w_k, \quad a_k = (-1)^{s_k} m_k 2^{e_k - \beta}, \quad w_k = \sum_{i=0}^{P-1} b_{k,i} 2^i, \quad (1)$$

where m_k is the 11-bit activation significand (with implicit one), e_k its exponent, β a fixed bias, and $b_{k,i}$ the i -th bit of the P -bit weight (sign, scale, and zero-point are handled outside the PE-array datapath, Section III-D). The pre-aligner replaces the factor $2^{e_k - \beta}$ by a right shift of each significand to the shared exponent $E = \max_k e_k$, giving aligned integer significands $\hat{m}_k = m_k \gg (E - e_k)$, so the dot product is an integer reduction over $\hat{m}_k w_k$.

Radix-4 recoding. Grouping the weight bits in pairs turns a P -bit weight into $\lceil P/2 \rceil$ radix-4 digits,

$$w_k = \sum_{j=0}^{\lceil P/2 \rceil - 1} d_{k,j} 4^j, \quad d_{k,j} \in \{0, 1, 2, 3\}, \quad (2)$$

and each digit contributes a single partial product $d_{k,j} \hat{m}_k$ instead of one per bit. The partial product is therefore one of four multiples of the aligned significand, $\{0, \hat{m}_k, 2\hat{m}_k, 3\hat{m}_k\}$, and the digit's place value 4^j is a fixed left shift of $2j$ applied at the edge. Radix-4 is the sweet spot: radix-2 needs $1.29\times$ the area per MAC, and radix-8/16 spend more on precomputed operands than they save in the adder tree (Table VI).

Processing element. FIGFLEX realizes this on a K -lane processing element (PE) (Figure 2b shows 4 lanes of the PE). The weight vector is loaded from the weight aligner atop the PE array and distributed among the lanes. Each lane then uses its 2-bit weight digit to drive a 4:1 multiplexer that selects the corresponding multiple. Of the four multiples, three cost no logic: $\hat{m}_k$ and $2\hat{m}_k = \hat{m}_k \ll 1$ are free wires and 0 is trivial; only $3\hat{m}_k$ is a *hard* multiple requiring an addition $\hat{m}_k + 2\hat{m}_k$. Because it depends only on the activation, it is formed once per row at the edge. A PE is thus a bank of K selects feeding one K -input binary adder tree and an integer accumulator; it contains no multiplier and no shifter.

Radix-4 replicator. The shift networks are hoisted to a per-row *radix-4 replicator* (Figure 2a). For each activation it receives the signed significand $\pm \hat{m}_k$ from the exponent aligner, forms its hard multiple $\pm 3\hat{m}_k$ —folding the activation sign in, so the PE needs no negation—applies each lane's radix-4 position shift, and broadcasts the K lane operands to every PE in the row. Note that the datapath is designed for non-negative weight codes, as produced by asymmetric quantization; it can support symmetric quantization by simply using 2^{P-1} as the zero point (P is the weight precision). The two shift networks (exponent alignment and multiple/position generation), which would dominate a per-PE implementation, are thus paid once per row.

Precision scaling. Because each lane carries two weight bits, the K -lane PE consumes $2K$ weight bits per cycle, so a b -bit weight sustains up to $2K/b$ MAC/cycle. INT4 spends $\lceil 4/2 \rceil = 2$ lanes per weight (for our choice $K=16$, 8 MAC/cycle), INT2 one lane per weight (16), and INT1 two weights per lane

(32), the lane's two digit bits selecting $\{0, a_i, a_{i+1}, a_i + a_{i+1}\}$; reconfiguring precision changes only the replicator's routing, while the select lanes, adder tree, and accumulator are shared. INT3 is the exception: $\lceil 3/2 \rceil = 2$ lanes per weight, but the high digit is a single bit occupying a two-bit lane, so half of every second lane is idle and the naive rate is only 8 MAC/cycle—no better than INT4. Recovering that waste is the subject of Section III-C.

C. INT3 Optimization

Our novel *high-bit lane fusion* reclaims the idle half lane by processing INT3 weights *in pairs* and *fusing* their two high bits into one shared lane.

Fused high-bit lane. Take a weight pair (w_0, w_1) with aligned significands a_0, a_1 . Their low digits $w_{0[1:0]}, w_{1[1:0]}$ occupy two ordinary radix-4 lanes over $\{0, a_i, 2a_i, 3a_i\}$. The two high bits contribute $4w_{0[2]}a_0 + 4w_{1[2]}a_1$. If the corresponding weight bits are aligned and paired, the PE can use a single 4:1 multiplexer to produce this contribution: fed the inputs $\{0, 4a_0, 4a_1, 4(a_0 + a_1)\}$ and the 2-bit code $\{w_{1[2]}, w_{0[2]}\}$, it realizes

$$4w_{0[2]}a_0 + 4w_{1[2]}a_1 = \begin{cases} 0 & 00 \\ 4a_0 & 01 \\ 4a_1 & 10 \\ 4(a_0 + a_1) & 11. \end{cases} \quad (3)$$

With high-bit lane fusion, two weights need only three lanes instead of four—two low digits plus one fused high digit—so INT3 packs the array with no idle bits and sustains $2K/3$ MAC/cycle: a frame of 32 weights (16 pairs) fills the $3K=48$ lanes of three consecutive cycles, so no lane is left idle even though $3 \nmid K$ (10.67 MAC/cycle for $K=16$), up from $K/2$. The fused lane reuses the same 4:1 mux as every other lane; only its *inputs* change. The replicator forms $4(a_0 + a_1)$ on the second output using the lane's existing adder (the $\times 4$ is wiring) and leaves the first output empty. The PE taps the required $4a_0, 4a_1$ from the two preceding lanes through two 2:1 muxes. Figure 2 shows the INT3 high-bit lane fusion datapath in green.

Weight aligner. Because the fused lane is driven by the two high bits of a pair, the incoming INT3 weight vector is

TABLE I: Per-lane radix-4 operation: every lane is the same 4:1 mux, and only the replicator's routing changes with precision. a_i : aligned activations; $\times 4$ shifts are wiring.

Lane	INT4 (2 lanes/wt)		INT3 (3 lanes/2 wt)	
	mux inputs	digit	mux inputs	digit
1	$\{0, a_0, 2a_0, 3a_0\}$	$w_{0[1:0]}$	$\{0, a_0, 2a_0, 3a_0\}$	$w_{0[1:0]}$
2	$\{0, 4a_0, 8a_0, 12a_0\}$	$w_{0[3:2]}$	$\{0, a_1, 2a_1, 3a_1\}$	$w_{1[1:0]}$
3	$\{0, a_1, 2a_1, 3a_1\}$	$w_{1[1:0]}$	$\{0, 4a_0, 4a_1, 4(a_0 + a_1)\}^\dagger$	$\{w_{1[2]}, w_{0[2]}\}$
4	$\{0, 4a_1, 8a_1, 12a_1\}$	$w_{1[3:2]}$	$\{0, a_2, 2a_2, 3a_2\}$	$w_{2[1:0]}$
5	$\{0, a_2, 2a_2, 3a_2\}$	$w_{2[1:0]}$	$\{0, a_3, 2a_3, 3a_3\}$	$w_{3[1:0]}$
6	$\{0, 4a_2, 8a_2, 12a_2\}$	$w_{2[3:2]}$	$\{0, 4a_2, 4a_3, 4(a_2 + a_3)\}^\dagger$	$\{w_{3[2]}, w_{2[2]}\}$

INT2 (1 lane/wt): lane $i = \{0, a_i, 2a_i, 3a_i\}$, digit $w_{i[1:0]}$

INT1 (2 wt/lane): lane $i = \{0, a_{2i}, a_{2i+1}, a_{2i} + a_{2i+1}\}$, digit $\{w_{2i+1}, w_{2i}\}$

[†]Fused high-bit lane (Section III-C).

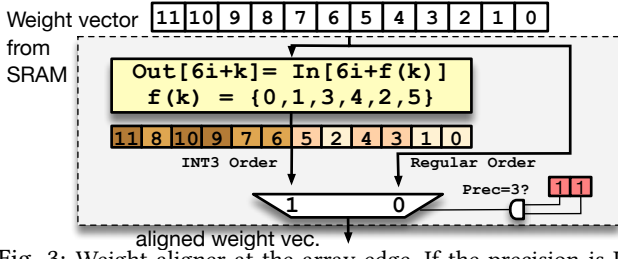


Fig. 3: Weight aligner at the array edge. If the precision is INT3, the MSBs of each weight pair $\{w_{(i+1)[2]}, w_{i[2]}\}$ are concatenated at the end.

repacked at the array edge by a small *weight aligner* (Figure 3): for each pair it emits the two low digits followed by the two MSBs $\{w_{1[2]}, w_{0[2]}\}$ concatenated at the end, so the PE consumes the three lane selectors contiguously and every third lane receives the fused code. Like the recoding, the transformation is exact—it merely reorders bits. For INT1, two consecutive 1-bit weights share one lane; their natural packing already places them on that lane, so INT1 needs no repacking and runs at twice the INT2 rate.

Table I summarizes the per-lane operation across precisions.

D. Full Accelerator

Figure 4 shows the complete accelerator: an $M \times N$ array of radix-4 PEs, each row served by three shared edge units—the exponent aligner, the radix-4 replicator (Section III-B), and the weight aligner (Section III-C)—so the array cost is dominated by the small mux-plus-adder-tree PEs. Each PE column drains its integer accumulator through a shared INT $\rightarrow$ FP converter that restores the block exponent, and a zero-point unit forms the per-group activation sums used by the epilogue.

Mixed-precision dataflow.

Figure 5 shows how a layer is processed, including how the column scales are absorbed into the activations. Our quantizer (Section IV) assigns precisions per output channel. Offline, the channels of each 128-input-channel group are sorted by precision, which splits a layer into three uniform-precision GEMMs (W_{x+1} , W_x , W_{x-1}) that any dataflow can run; the stored precision tags identify each partial sum’s channel, so outputs land in natural order without an index table. Within a group the array is output-stationary; across groups the loop order is input-stationary: a group’s activations stay in the row buffers while the output tiles of all three precision parts stream through, so they are fetched once. A 128-channel tile that straddles two precisions runs in the higher mode (lower-precision codes zero-pad), which keeps utilization above 99% on Llama-3.1-8B (Table II) and above 93% for its smallest layers. Tokens map to the 16 PE rows, so a batch of 16 or more fills the array; smaller decode batches are memory-bound (Figure 1) and gain from the reduced weight traffic. Every mode reads one 32-bit weight word per column per cycle, the act-order permutation is a static gather on the activation read path, and the epilogue drains each group while the next computes, even at the INT1 rate.

Dequantization-free epilogue. Sign, per-channel scale, and the per-group affine are applied to the *scalar* accumulator

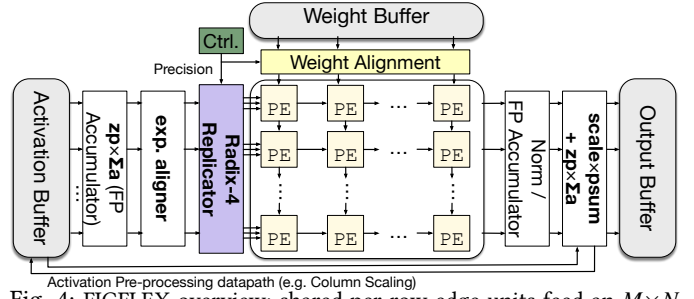


Fig. 4: FIGFLEX overview: shared per-row edge units feed an $M \times N$ array of radix-4 PEs; per-column converters and the epilogue produce the FP output.

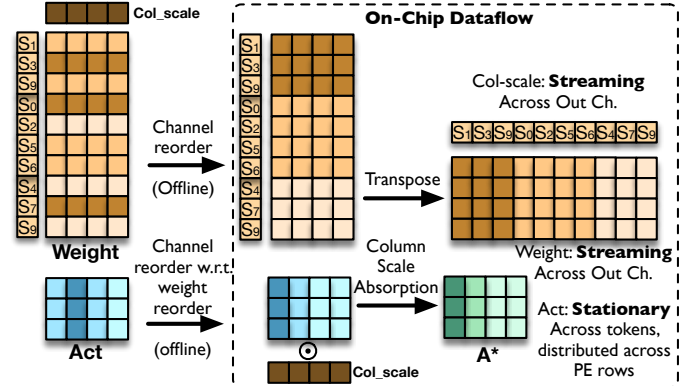


Fig. 5: Dataflow example. Offline, the weight channels are reordered by precision and the activation channels follow the same order. On chip, the column scales are absorbed into the activations (A^*), which stay stationary across the PE rows while the weights stream across output channels.

output, never to the weights, so no dequantization datapath is required. Folding the per-input-channel column scale c_k into the activation at the pre-aligner ($a'_k = a_k c_k$), a PE emits the integer dot $S_q = \sum_k a'_k q_k$ over a group, and the epilogue computes

$$Y[m, n] = \sum_g \text{scale}_{n,g} \left(S_q - \text{zp}_{n,g} \sum_{k \in g} a'_k \right), \quad (4)$$

where $\sum_{k \in g} a'_k$ is formed once per group and reused across output channels. No weight is ever reconstructed to FP16; the epilogue is a small pipelined FP unit that fires once per quantization group.

IV. Proposed Mixed-Precision Quantization

A. Overview

We build on GPTQ [10] (Section II-B), which quantizes every group of a layer at the same bit-width. Our quantizer (Figure 6) adds two contributions: (i) **column scaling**, a per-input-channel normalization (Subsection IV-B), and (ii) a **compensation-aware (CA) mixed-precision** assignment over $\{W_4, W_3, W_2, W_1\}$ per output channel (Subsection IV-C). Everything else follows prior work. GPTQ runs with act-order (groups are formed in that order) and 1% damping, block by block on the outputs of the already-quantized blocks (calibration in Subsection V-A). A group is 128 input channels of one output channel with an FP16 scale and an integer zero

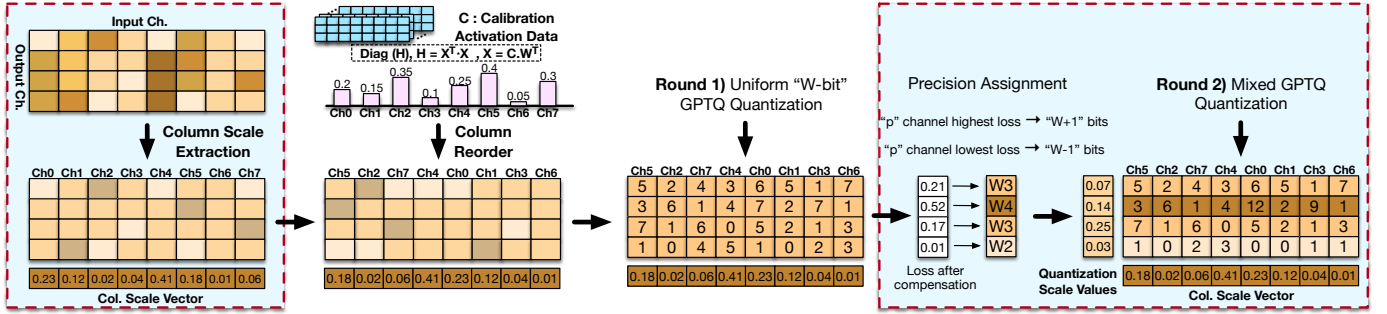


Fig. 6: FIGFLEX’s Mixed-Precision Quantization Framework. The FIGFLEX quantizer’s contributions are highlighted by dashed boxes.

point [10, 19, 32], and a channel’s precision is set separately in each of its groups. Scales and zero points are fitted by the clip-range search of SliM-LLM [19]. W1 groups use SliM-LLM’s mean-centred binarization $\mu \pm \alpha$, which is affine in the 1-bit code (scale = 2α). Every precision therefore reduces to $W_{n,k} = c_k(q_{n,k} - zp_{n,g})\text{scale}_{n,g}$, the form the epilogue consumes without dequantization (Eq. (4)).

B. Column Scaling

For each input channel k we compute

$$c_k = \frac{1}{N} \sum_n |W_{n,k}|, \quad (5)$$

normalize $\tilde{W}_{n,k} = W_{n,k}/c_k$ before quantization, and fold c_k back at inference. Because c_k is per input channel, it is equivalently a *per-channel activation scale* $a'_k = a_k c_k$ and costs nothing in the weight path (Section III-D). Figure 5 shows that the column scales can be absorbed into the activations as scalar values prior to the linear GEMM. The multiply happens once per activation element at the pre-aligner and is shared by all N output channels ($1/N$ of the MACs). Because c_k is small, $a_k c_k$ can underflow into the FP16 subnormal range, which the pre-aligner flushes to zero. We therefore rescale c per layer by a power of two, $c' = c/2^s$, and multiply the group scales by 2^s , which leaves the quantized model unchanged; s places the largest calibration product $|a_k c'_k|$ at 2^{12} . Column scaling flattens the per-channel dynamic range and is the main reason the aggressive W1 groups remain usable: on Llama-3.1-8B at 2 bits, removing it raises perplexity from 13.77 to 16.48 (Table V). The scaling runs in the opposite direction to AWQ [32]. AWQ multiplies the weight columns of activation-salient channels by $s_k = \overline{|x_k|}^\alpha$, so that they use more of the grid, and divides the activations by s_k . Column scaling instead divides every weight column by its own magnitude, $\tilde{W}_{n,k} = W_{n,k}/c_k$, so that no outlier column sets the step ($\max_{k \in g} \tilde{W}_{n,k} - \min_{k \in g} \tilde{W}_{n,k}$)/($2^b - 1$) of a 128-channel group; the step then follows the typical columns, which lowers the rounding error of every weight in the group, and the factor returns through the activation, $a'_k = a_k c_k$.

C. Compensation-Aware Assignment

Our quantizer decides which output channels receive more bits by the error that GPTQ *cannot* compensate, rather than by a salience score computed before quantization, as in SliM-LLM’s salience-determined bit allocation (SBA) [19] and PB-LLM [46]. A pre-quantization score misses two effects. GPTQ’s

Algorithm 1 Proposed per-layer quantization

Require: weights W , calibration Hessian H , target width b^* , share p

- 1: $c_k \leftarrow \frac{1}{N} \sum_n |W_{n,k}|$; $\tilde{W} \leftarrow W/c$; fold c into H
- 2: permute columns by descending $\text{diag}(H)$ $\triangleright$ act-order
- 3: $\rho_{n,g} \leftarrow$ uncompensated error of a uniform b^* -bit GPTQ pass (Eq. 6)
- 4: b^*+1 bits to the top- p channels by ρ , b^*-1 to the bottom- p , b^* to the rest
- 5: **for** each group g in processing order **do**
- 6: fit $(\text{scale}_{n,g}, zp_{n,g})$ at its precision (W1: $\mu \pm \alpha$)
- 7: **for** each column $k \in g$ **do**
- 8: $\hat{w} \leftarrow \text{quantize}(w_k)$ at the row’s precision
- 9: propagate $(w_k - \hat{w})$ to remaining columns via H^{-1}
- 10: **end for**
- 11: **end for**
- 12: **return** q , per-group (scale, zp, 2-bit tag), c

error feedback already moves the rounding error of salient columns onto less salient ones, an implicit bit allocation within each row, and column scaling flattens the spread of group sensitivity ($4.97\times$ to $1.66\times$). A salience ranking therefore promotes groups that GPTQ would have compensated anyway: SBA is no better than uniform 3-bit (7.237 vs. 7.227 on Llama-3.1-8B, Table V) when column scaling is enabled. We instead rank channels by the error that remains *after* compensation. GPTQ’s loss is a sum of per-column terms $((w_{n,j} - q_{n,j})/U_{jj})^2$, the part of each rounding error that the remaining columns cannot absorb, where U is the upper Cholesky factor of H^{-1} [10]. A first GPTQ pass at the target width b^* gives each channel’s share of this loss in group g ,

$$\rho_{n,g} = \sum_{j \in g} \left(\frac{w_{n,j} - q_{n,j}^{(b^*)}}{U_{jj}} \right)^2. \quad (6)$$

One more bit shrinks a channel’s error, and with it $\rho_{n,g}$, by a roughly constant factor r , so promoting a channel saves $(1 - 1/r)\rho_{n,g}$ and demoting one costs $(r - 1)\rho_{n,g}$. For a given number of channels moved, promoting those with the largest ρ by one bit and demoting as many with the smallest ρ by one bit therefore minimizes the loss to first order while keeping the average at exactly b^* . A second GPTQ pass then quantizes the original weights at these widths (Algorithm 1).

The number of channels moved is one global share (1% for F3, 20% for F2; Subsection V-A), so unlike SBA and AMQ [30], which search an allocation per layer, the assignment needs no per-layer search; its cost is one extra GPTQ pass per layer.

D. Storage Overhead

Each layer stores the codes q at their assigned widths, one FP16 scale and integer zero point per group as in GPTQ, a 2-bit precision tag per group, i.e., 0.016 bits per weight, and the vector c : one FP16 value per input channel (below 0.004 bits per weight for $N \geq 4096$) and one power-of-two exponent. The reported 3.0- and 2.0-bit averages count the codes only, as is standard.

V. Evaluation

A. Experimental Setup

Quantization baselines and model quality.

We evaluate six open models spanning three families and two size classes: LLaMA-2-7B [52], Llama-3.1-8B [14], Qwen2.5-7B [56], Qwen3-8B and Qwen3-32B [55], and Gemma4-31B [50]. We report WikiText-2 [1] and C4 [43] perplexity (2048-token sequences; C4 on the GPTQ evaluation split) and zero-shot accuracy on PIQA [4], ARC-easy/challenge [6], HellaSwag [57], and Winogrande [45] (lm-eval-harness [13]). Two operating points are used throughout, **F3** and **F2** (3.00 and 2.00 average bits, Table II); their share p (Subsection IV-C) is the best of a small grid on the WikiText-2 validation split of LLaMA-2-7B and is fixed for all models. We compare against GPTQ [10], AWQ [32], Slim-LLM [19], AMQ [30] (at its paper’s search budget), and QuIP# [53], each with its official implementation. GPTQ, Slim-LLM, and FIGFLEX calibrate on the same 128 random 2048-token WikiText-2 training segments; AWQ, AMQ, and QuIP# use their authors’ calibration sets (Pile [12], RedPajama [51]). Only the decoder linear layers are quantized; embeddings, the LM head, and activations stay in FP16, and no method is fine-tuned.

Hardware Baselines. We implement floating-point Fused Multiply-Add (FMA) and FP dot-product engines as conventional baseline engines. Beyond them, we compare FIGFLEX against three state-of-the-art FP-INT accelerators, all re-implemented and synthesized in our own flow. **FIGNA** [20] aligns each activation block to a shared exponent and multiply-accumulates the aligned significands against the INT4 weights

TABLE II: Precision fractions at each operating point (same for all models; p selected on the WikiText-2 validation split). MAC/cycle: effective rate of the mix on a 16-lane PE (INT4/3/2/1: 8/10.67/16/32); $\times$ INT4: speedup over uniform INT4; Util.: array utilization of the sorted tile schedule (Section III-D) on Llama-3.1-8B.

Point	p	W4/W3/W2/W1 (%)	Bits	MAC/cycle	$\times$ INT4	Util.
F3	1%	1/98/1/-	3.00	10.67	1.33 $\times$	99.4%
F2	20%	-/20/60/20	2.00	16.0	2.00 $\times$	99.0%

TABLE III: Datapaths under comparison; only FIGFLEX is accurate, bit-parallel, and precision-scaling.

Design	Key datapath
FP FMA	FP mult. + align + normalize $\rightarrow$ FP acc.
Dot-product	FP $\times$ INT4, per-element normalize $\rightarrow$ FP acc.
FIGNA [20]	shared-exp. align + INT mult. $\rightarrow$ INT acc.
FIGLUT [40]	activation LUT read (bit-serial) $\rightarrow$ INT acc.
AxCore [60]	log-approx. mult. $\rightarrow$ partial-FP acc.
FIGFLEX	radix-4 4:1 mux, no mult. $\rightarrow$ INT acc.

as integers, returning to floating point once per output. **AxCore** [60] is approximate: a shared pre-add applies the Mitchell compensation bias, the FP $\times$ INT multiply reduces to a logarithmic-domain integer addition, and the result is accumulated in a per-PE partial floating-point adder that defers normalization and rounding to a shared module, as in the published design. **FIGLUT** [40] replaces the multiplier with a table read, pre-computing sums of aligned activations once and reading one entry per weight digit; we implement the half-size LUT of its paper as real register storage and take its scaling with activation width from the published cross-format comparison. Table III summarizes and compares the key datapaths of each baseline. Our re-implementations track the published area ratios: averaged over FP16 and FP32, our AxCore-to-FIGNA ratio is within 2.5 points of the AxCore paper and our FIGLUT-to-FIGNA ratio within 2.4% of the FIGLUT paper.

Hardware evaluation setup. We implement FIGFLEX and all baselines as parameterized SystemVerilog RTL and synthesize them using Synopsys Design Compiler in the NanGate 15 nm technology, targeting 1 GHz. All results use this common clock. Every design meets it in all three activation formats, and all but the dot-product engine (1 ps of slack in FP32) close with margin, so the common clock does not favor FIGFLEX. For a fair comparison, all designs are adjusted to the same maximum-performance (TOPS) target on a 128×128 array (16,384 MAC/cycle). All designs are evaluated for nine cases—activations in FP32, FP16, or BF16 and weights in INT4, INT3, or INT2—with each design fully re-synthesized per activation format rather than analytically scaled. INT4 is a uniform 4-bit quantization; INT3 and INT2 are our proposed mixed-precision quantization averaging 3.0 and 2.0 bits per weight (the F3 and F2 mixes of Table II). Rigid baselines process any sub-INT4 weight by zero-padding their INT4 datapath, as published, and so gain nothing below 4 bits. Power is obtained under an identical switching activity of 0.2 toggles per clock cycle for every design. To model memory properly, input, weight, and output buffers are compiled as SRAM and characterized with CACTI [38], sized identically across designs. They hold 1.2 MB in total. Off-chip memory is one HBM3E stack at 3.44 pJ/bit [37] with 1280 GB/s [27]; its traffic follows from tiling each GEMM with a 512×512 on-chip output block and is the same per MAC for every design, so HBM power differs only through throughput. Total energy comprises three terms: (1) memory energy, the access

TABLE IV: WikiText-2 (Wiki2) and C4 perplexity ($\downarrow$) and zero-shot accuracy ($\%$, $\uparrow$; HellaSwag (HS) length-normalized) at 3- and 2-bit average weight precision. L2-7B: Llama-2-7B [52]; L3.1-8B: Llama-3.1-8B [14]; Q2.5-7B: Qwen2.5-7B [56]; Q3-8B/32B: Qwen3-8B/32B [55]; G4-31B: Gemma4-31B [50]. $\dagger$ AMQ runs at 3.03–3.05 and 2.25–2.40 average bits. AWQ diverges at 2 bits ($\text{PPL} > 10^3$) and is omitted there.

Method	Bits	L2-7B							L3.1-8B							Q2.5-7B						
		Wiki2	C4	PIQA	ARcE	ARcC	HS	Wino	Wiki2	C4	PIQA	ARcE	ARcC	HS	Wino	Wiki2	C4	PIQA	ARcE	ARcC	HS	Wino
FP16	16	5.47	6.97	78.1	75.5	43.0	76.2	69.2	6.24	8.95	79.1	82.2	52.1	79.3	74.5	6.85	10.44	78.8	80.4	47.8	60.0	72.9
GPTQ	3	6.14	8.05	76.0	73.1	39.0	71.9	67.6	7.74	11.95	75.5	72.8	40.2	73.9	70.7	7.69	11.77	76.9	77.1	45.9	55.8	68.1
AWQ	3	6.18	7.81	76.6	74.3	40.6	54.8	68.0	8.15	11.56	76.6	76.1	45.1	56.1	70.9	8.14	11.75	77.5	79.4	46.8	56.0	69.5
QuIP#	3	6.24	7.70	76.5	74.4	41.2	53.4	69.1	7.79	10.89	77.3	75.3	43.6	56.1	70.8	7.45	11.05	78.4	80.4	48.6	56.9	72.0
AMQ †	3.05	6.70	8.83	76.4	69.4	42.4	72.1	68.7	9.20	14.29	76.6	70.8	43.9	71.3	70.0	12.30	20.41	73.9	66.5	41.5	67.0	63.3
SlM-LLM	3	6.40	8.57	75.4	52.2	37.4	69.1	64.5	7.70	13.71	77.2	68.4	44.5	73.9	71.1	7.60	13.43	77.5	72.3	48.3	75.4	69.1
FIGFLEX	3	5.85	7.67	76.3	74.5	42.3	73.3	68.7	7.19	10.95	78.1	78.2	46.8	76.0	71.3	7.46	11.37	78.8	79.1	46.1	57.0	70.6
GPTQ	2	32.61	59.80	53.2	25.6	21.5	25.6	48.1	98.94	310.8	54.2	26.1	20.5	27.5	50.9	19.62	60.83	56.8	33.9	24.8	31.4	50.4
QuIP#	2	13.24	12.83	71.1	61.7	30.0	41.2	62.8	21.20	23.35	69.1	65.1	21.2	39.7	62.7	14.16	15.25	74.1	76.6	42.1	47.6	67.6
AMQ †	2.40	10.60	13.99	72.1	55.1	35.2	62.4	61.6	22.90	33.87	70.2	54.3	32.4	54.2	62.3	44.40	62.04	66.7	55.0	32.9	54.5	58.4
SlM-LLM	2	30.80	36.83	62.0	39.1	27.1	39.8	55.6	63.10	298.76	54.8	32.5	31.3	29.4	50.8	15.00	48.54	61.8	47.0	28.2	46.1	53.9
FIGFLEX	2	8.77	13.86	66.9	54.5	28.7	55.2	59.3	13.77	27.36	64.2	50.6	26.1	49.4	59.6	10.47	17.43	67.7	57.9	30.1	45.1	60.1

Method	Bits	Q3-8B							Q3-32B							G4-31B						
		Wiki2	C4	PIQA	ARcE	ARcC	HS	Wino	Wiki2	C4	PIQA	ARcE	ARcC	HS	Wino	Wiki2	C4	PIQA	ARcE	ARcC	HS	Wino
FP16	16	9.72	13.29	76.9	83.6	55.8	57.1	67.8	7.60	10.77	81.1	84.3	57.9	63.9	73.1	6.38	9.29	82.9	88.4	64.9	65.9	80.0
GPTQ	3	10.66	14.64	74.3	76.7	47.5	53.2	65.8	8.48	11.88	77.7	78.5	51.5	61.2	67.9	7.23	10.72	81.6	83.7	56.1	61.8	76.9
AWQ	3	11.48	15.08	75.1	78.2	47.8	52.8	65.0	8.63	11.74	79.3	81.1	53.7	61.2	72.5	8.26	11.43	81.3	86.2	61.3	62.6	77.7
QuIP#	3	10.41	14.20	75.8	81.9	52.2	53.9	68.4	8.04	11.18	80.0	83.0	59.3	62.3	72.1	7.21	10.13	82.2	86.6	60.6	62.4	79.0
AMQ †	3.05	12.90	19.35	74.8	74.6	49.7	68.2	64.0	9.13	14.24	80.2	79.7	57.7	79.2	68.4	8.61	13.28	82.6	83.8	60.0	80.8	77.3
SlM-LLM	3	11.60	18.54	75.5	70.4	48.4	69.9	65.2	8.41	13.78	80.4	78.5	58.2	80.2	71.4	7.20	11.57	83.4	86.1	62.5	82.1	77.0
FIGFLEX	3	10.30	14.24	74.7	80.1	50.1	53.5	66.8	8.11	11.48	78.5	81.7	54.1	61.5	71.4	6.99	10.27	81.9	85.9	59.1	62.8	77.5
GPTQ	2	35.21	74.14	51.9	27.8	22.4	27.9	51.3	23.31	34.90	57.9	31.8	22.8	36.3	49.3	53.97	150.67	58.4	36.8	23.9	33.7	51.9
QuIP#	2	21.18	21.02	71.1	75.3	43.0	45.4	65.5	9.59	13.05	76.3	83.5	55.8	57.1	72.5	13.01	15.36	76.7	80.9	48.7	49.5	74.1
AMQ †	2.40	32.30	41.65	69.0	51.9	32.3	53.0	57.6	11.94	18.00	76.1	74.5	48.9	70.6	64.3	36.19	47.89	74.3	64.6	40.0	64.5	62.0
SlM-LLM	2	26.50	57.24	62.0	39.2	26.5	43.1	50.9	13.90	58.11	66.8	46.5	32.5	61.9	58.0	14.10	28.85	68.2	51.5	33.4	53.0	61.3
FIGFLEX	2	14.62	23.96	65.8	52.2	27.7	39.4	56.8	11.20	17.02	72.0	60.4	34.6	50.1	61.5	10.95	17.99	75.5	71.7	39.8	47.8	66.1

count times the energy per bit, (2) logic dynamic energy, the synthesized per-operation energy times the operation count, and (3) leakage energy, the leakage power times the latency. Following FIGNA [20], we report TOPS/mm² and normalized energy, including SRAM and off-chip communication. Results are given for four open LLMs (Llama-3-8B, Qwen3-8B, Qwen3-32B, Gemma4-31B), each normalized to the same-format, same-precision floating-point FMA engine.

B. Numerical Fidelity

To verify that model quality is preserved on real end-to-end requests, we run the quantized F3 and F2 Llama-3.1-8B models through a bit-accurate model of the FIGFLEX datapath, including the column scales, the shared-exponent alignment (6 guard bits, truncation, subnormals flushed), the integer accumulation, and the group epilogue. Their WikiText-2 perplexity matches the GPU to the third decimal (F3: 8.2511 vs. 8.2508; F2: 15.3005 vs. 15.3004 on four test windows), so the quality in Table IV holds on FIGFLEX. We validate the model against the RTL: the integer datapath of the full 16×128 -PE array, simulated in Verilator on all 21 projections of blocks 0, 15, and 31 with real activations and each group in its own mode, matches it bit for bit on all 10.0 million group results; the FP32 post-processing uses DesignWare units modeled from their specification.

Figure 7 compares the per-projection error against the float64 dot product. The radix-4 recoding and the integer accumulation are lossless, so the only rounding is the shared-exponent alignment, and FIGFLEX is at least as accurate as a tensor core that multiplies in the activation format and accumulates in FP32, for FP16, BF16, and FP32 activations (FP16, F3: 2.0×10^{-3} vs. 2.5×10^{-3}). Finer exponent blocks or 24 guard bits leave the error unchanged, and the power-of-two rescaling (Subsection IV-B) keeps the column-scaled activations out of the FP16 subnormal range.

C. Model Quality

Table IV reports perplexity and zero-shot accuracy at 3- and 2-bit average precision. GPTQ [10], AWQ [32], SlM-LLM [19], AMQ [30], and FIGFLEX produce uniform affine integer codes that an FP-INT array consumes directly. QuIP# [53] is run without fine-tuning and is a reference for quality without the affine-grid constraint, not a ceiling.

Perplexity. At 3 bits, FIGFLEX has the lowest WikiText-2 perplexity among the affine methods on all six models and the lowest overall on four; QuIP# leads on the other two by at most 0.07. At 2 bits, where a uniform affine grid normally fails, the gap widens: GPTQ, AWQ, and SlM-LLM collapse, while FIGFLEX stays within $1.5\text{--}2.2\times$ of FP16 and is the best method overall on five of six models, 21% below QuIP# on average and below AMQ on all six despite AMQ’s 2.40-bit

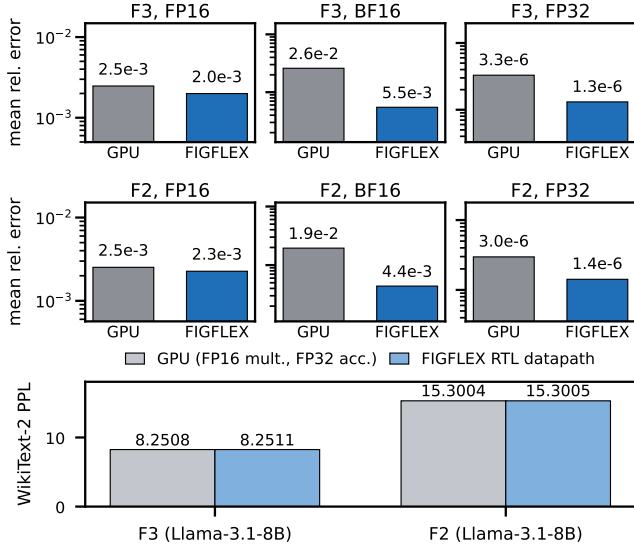


Fig. 7: Numerical fidelity on the F3 and F2 models (Llama-3.1-8B). *Top*: mean relative error against the float64 dot product over 21 projections; the tensor core multiplies in the activation format and accumulates in FP32. *Bottom*: WikiText-2 perplexity on the GPU and through the FIGFLEX datapath.

average. On the held-out C4, FIGFLEX remains the best affine method on every model at both widths and is within 0.32 of QuIP# at 3 bits.

Zero-shot accuracy. Averaged over PIQA, ARC-e, ARC-c, and Winogrande on the six models (FP16: 72.1%), FIGFLEX is the most accurate affine method at 3 bits (69.7%, vs. 69.2% for AWQ and 67.5% for GPTQ), within 0.7 points of QuIP#. At 2 bits it leads GPTQ and SLiM-LLM at the same 2.00 bits by 14.9 and 7.0 points, and trails AMQ by 2.6 points and QuIP# by 9.1.

D. Quantizer Ablation

Table V adds the components of our quantizer one at a time on Llama-3.1-8B with the same calibration. The tuned uniform row adds clip search and column scaling to GPTQ; the next rows add our compensation-aware (CA) mixed assignment without and with column scaling, and SBA-Mixed replaces it with SLiM-LLM’s salience-determined bit allocation [19] (a pre-quantization ranking w^2/U_{jj}^2 with a per-layer search of the share) on the same backbone. Latency is the linear-GEMM time of one 2048-token prefill on the iso-TOPS array, including the column-scale multiply and the measured tile utilization (Table II).

At 3 bits, CA-Mixed lowers perplexity at 0.6% higher latency. The gain is small because 3-bit error is already low; F3’s hardware benefit is throughput instead (10.67 vs. 8 MAC/cycle for INT4). At 2 bits, mixed precision is the larger effect: on its own, CA-Mixed lowers perplexity from 98.94 (GPTQ) to 16.48, below the tuned uniform model (29.28), and column scaling brings it to 13.77. The ranking matters: SBA-Mixed is no better than uniform at 3 bits (7.237 vs. 7.227) and reaches only 18.84 at 2 bits, because a salience

TABLE V: Quantizer ablation on Llama-3.1-8B (FP16: 6.238 WikiText-2 PPL). Every row after GPTQ builds on it (act-order GPTQ). CA: compensation-aware; SBA: salience-determined bit allocation. Latency: linear-GEMM time of one 2048-token prefill, including the column-scale multiply.

Configuration	WikiText-2 PPL		GEMM latency (ms)	
	3-bit	2-bit	3-bit	2-bit
GPTQ (official, act-order)	7.744	98.94	654.3	436.2
Tuned uniform (col. scaling)	7.227	29.28	666.9	448.8
CA-Mixed, no col. scaling	7.370	16.48	658.3	440.6
SBA-Mixed + col. scaling [19]	7.237	18.84	670.8	453.2
CA-Mixed + col. scaling (ours)	7.193	13.77	670.8	453.2

TABLE VI: Datapath ablation, synthesized in the same flow (FP16 activations). Op.: precomputed operands per lane; PE area includes registers; Repl.: row-replicator area per PE; area and power per MAC at the INT4 rate. All designs run INT4/INT2/INT1 at 8/16/32 MAC/cycle.

Datapath	Op.	PE (μm^2)	Repl. (μm^2)	μm^2 / MAC	mW/ MAC	INT3 MAC/cyc.
Radix-2 tree	1	1144	12.9	144.6	0.167	10.67
Radix-4	3	873	12.7	110.8	0.162	8
Radix-4+INT3	3	881	16.3	112.2	0.165	10.67
Radix-8	7	911	21.3	116.6	0.168	10.67
Radix-16	15	1239	30.7	158.7	0.201	10.67

score computed before quantization cannot see which groups GPTQ will compensate. Because every precision runs at 32/ b MAC/cycle, equal average bits mean equal throughput: F2 runs at the rate of uniform W2 up to its 1% tile-packing loss (453.2 vs. 448.8 ms) at $2.1\times$ lower perplexity. Column scaling costs 12.6 ms per prefill (1.9–2.8%).

E. Hardware Efficiency

All hardware results in this section are for the linear GEMM layers, which dominate LLM runtime (Figure 1); attention and the rest of the model are outside FIGFLEX. We estimate the end-to-end effect at the end of the section.

Datapath ablation. Table VI isolates the datapath choices, each synthesized in the same flow. Moving from a radix-2 fused adder tree to radix-4 with the multiples hoisted to the row edge cuts the PE area by 24% and the area per MAC by 23% at the same power per MAC, because each lane resolves two weight bits with one 4:1 mux and the adder tree halves. The INT3 fusion costs 1.3% area (the replicator grows from 12.7 to 16.3 μm^2 per PE) and raises the INT3 rate from 8 to 10.67 MAC/cycle, so the area per INT3 MAC drops from 110.8 to 84.1 μm^2 (1.32 $\times$). Higher radices need more precomputed operands per lane (7 for radix-8, 15 for radix-16), whose replicator and wider muxes outweigh the smaller tree: radix-8 costs 3.9% more area and 1.8% more power per MAC than our datapath, and radix-16 41% and 22%. Radix-4 is therefore the sweet spot, and the INT3 fusion is nearly free. Because the INT4 comparison below involves no precision flexibility, it reflects this datapath alone.

Absolute results.

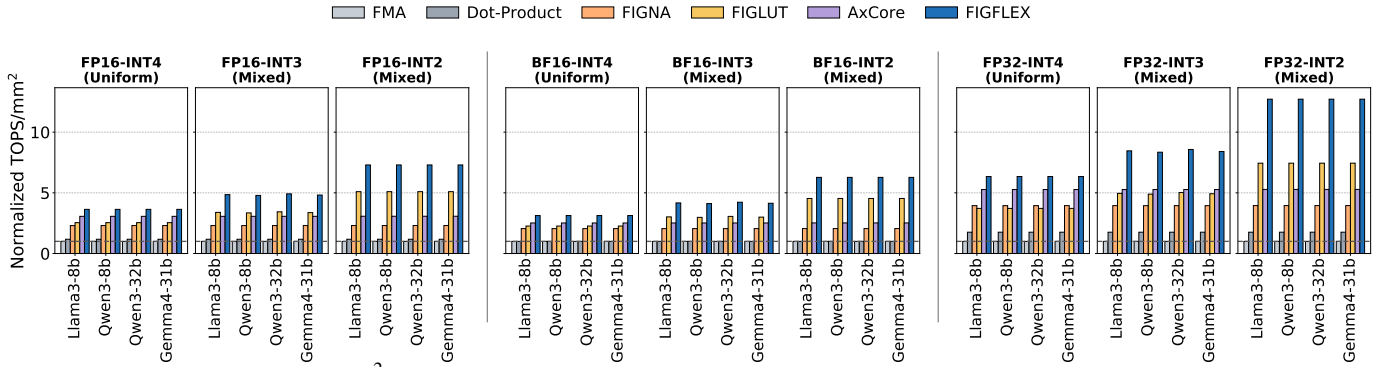


Fig. 8: Compute density (TOPS/mm²) normalized to a same-format FMA core ($\equiv 1.0$), for INT4 and mixed-precision (INT3) operating points across models. FIGFLEX leads every design; its advantage grows at sub-4-bit precision.

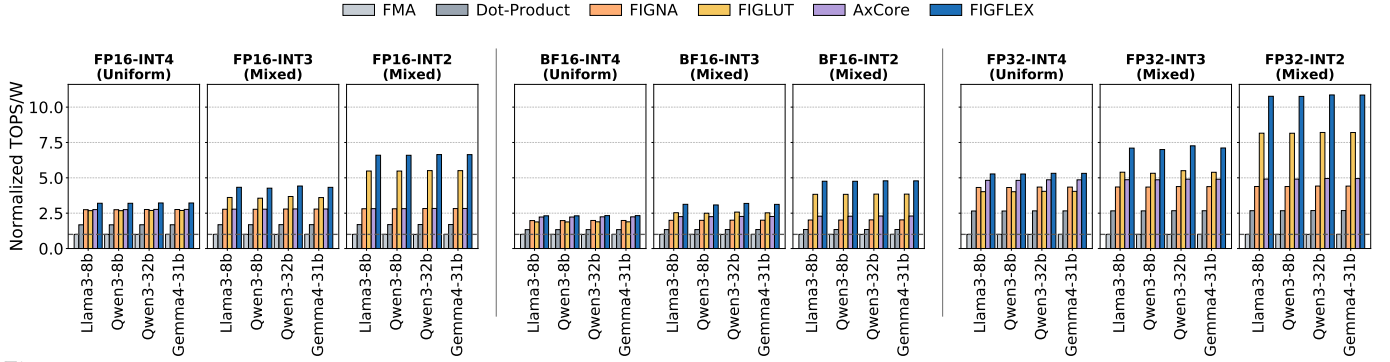


Fig. 9: Energy efficiency (TOPS/W) normalized to the same-format FMA core. FIGFLEX’s integer, multiplier-free datapath is the most energy-efficient, ahead of FIGNA’s integer multiplier, FIGLUT’s LUT reads, and AxCore’s log-approximation.

TABLE VII: Absolute results (FP16 activations, 128×128 MACs, 1 GHz). Area includes the same 1.2 MB SRAM (1.28 mm²) and output conversion (0.52 mm²). Power at INT4 (0.2 toggle rate), chip and with HBM3E traffic (Llama-3.1-8B). All designs reach 32.8 TOPS and a 4.76 s request at INT4. Request: one Llama-3.1-8B chat request (4k/4k tokens, batch 128; time per request), weights and KV cache at the F3/F2 bit-widths, HBM3E at 1280 GB/s, attention on a common FP16 unit.

Design	Area (mm ²)	Power (W)		TOPS		Request (s)	
		chip	+HBM	F3	F2	F3	F2
FP FMA	11.10	17.52	20.17	32.8	32.8	4.58	4.39
Dot-product	9.52	9.59	12.24	32.8	32.8	4.58	4.39
FIGNA [20]	5.11	5.04	7.69	32.8	32.8	4.58	4.39
FIGLUT [40]	4.68	5.22	7.87	43.7	65.5	3.70	2.65
AxCore [60]	3.97	5.02	7.67	32.8	32.8	4.58	4.39
FIGFLEX	3.43	4.02	6.67	43.4	64.9	3.72	2.67

Table VII gives the absolute numbers behind the normalized figures. The physical units differ: FIGFLEX has 2,048 PEs of 16 radix-4 lanes, FIGLUT 16,384 read-accumulate units sharing about 1,107 8-entry LUTs, and the others 16,384 MAC units. FIGFLEX has the smallest area and power because its PE has no multiplier and the activation multiples are formed once per row; its compute array is 25% smaller than AxCore’s. At INT4 every design takes the same request time, since the configuration is iso-TOPS. At F3 and F2, FIGFLEX is 1.23×

and 1.64× faster than the fixed-precision designs, which gain only from the smaller weights and KV cache; FIGLUT, the other precision-scalable design, matches this runtime but needs 1.36× the area and 1.30× the power. Power uses a common 0.2 toggle rate as an estimate. As a check, we replaced it with per-bit toggle rates and static probabilities measured on real Llama-3.1-8B tensors in each design’s input encoding, propagated by the synthesis tool: at INT4, FIGFLEX’s TOPS/W lead remains 1.18–1.50× over FIGNA, 1.15–1.46× over AxCore, and 1.04–1.32× over FIGLUT.

Compute density.

Figures 8 and 9 report whole-chip compute density (TOPS/mm²) and energy efficiency (TOPS/W), including the SRAM, normalized to the same-format FMA design. INT4 is uniform GPTQ; INT3 and INT2 use our mixed-precision quantizer. FIGFLEX is the most area-efficient design at every operating point. At INT4, the operating point FIGNA and AxCore are designed for and where no design benefits from precision flexibility, it reaches 1.49× the throughput per area of FIGNA and 1.18× that of AxCore (geometric mean over the three formats), which isolates the benefit of the radix-4 datapath. At INT3 the advantage grows to 1.97× and 1.56×, partly because those baselines are fixed-precision. Re-implementing them for 3 or 2 bits would recover only part of this. In AxCore, narrower weights shrink only the approximate multiplier, a small part of its area; the FP accumulator that follows dominates and stays intact (Figure 11), so its throughput per area barely improves. Against a custom

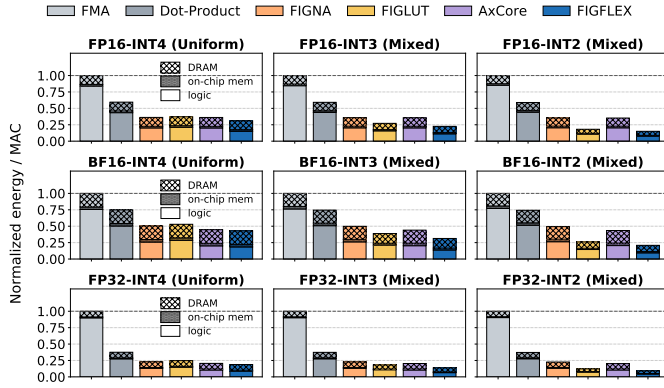


Fig. 10: Normalized energy comparison of baseline accelerators running Qwen3-32B

INT3/INT2 FIGNA, FIGFLEX’s advantage would be similar to the uniform INT4 case, since it comes from the fused radix-4 MAC datapath. Against designs that also scale with precision, it remains: a BitBlade-style [44] radix-2 tree (Table VI) runs every precision at FIGFLEX’s rate but needs $1.29\times$ more area per MAC, and FIGLUT, which scales throughput with precision as FIGFLEX does, trails it by $1.42\times$ at INT3 and INT2 alike, purely from the per-MAC cost. Because every precision runs at $32/b$ MAC/cycle, throughput, density, and energy per operation depend only on the average bit-width, not on how a quantizer distributes it.

FIGFLEX is also the most energy-efficient design at every datapath and operating point, surpassing even FIGLUT—which, being bit-serial, is inherently the most energy-efficient of the baselines below 4 bits—by an average of 25%.

Energy Comparison. Figure 10 compares the energy consumption of the baselines running three variations of quantization of the Qwen3-32B model (Uniform INT4, Mixed Precision INT3, and Mixed Precision INT2). FIGFLEX consumes the least energy at every operating point; AxCore is closest at INT4 and FIGLUT at INT3 and INT2. DRAM and SRAM access energy is also captured, as discussed in Section V-A.

End-to-end estimate.

Figure 1 extends the evaluation to a full request with the first-order model of Section II-A. Only the linear GEMMs are measured for FIGFLEX: we place it in the area of the GPU’s tensor cores, where it runs the GEMMs at its measured compute density relative to an FP16 dot-product engine ($3.2\times$ at INT4, Figure 8) times the F3/F2 rates (Table II), i.e., $4.2\times$ and $6.3\times$ the tensor-core rate. Attention, the non-GEMM layers, and memory traffic are modeled identically for both systems: GPU-equivalent compute and the same bandwidth, with weights and KV cache at the same bit-width. The baseline is the tensor cores running low-bit kernels (W3A16 and W2A16, as in FLUTE [17]), which dequantize to FP16 and compute at the FP16 rate. Replacing them with FIGFLEX shortens a batched request (batch 128) by $2.2\text{--}2.9\times$ at F3 and $2.9\text{--}3.7\times$ at F2. At batch 1, decode is memory-bound and both read the same bytes, so the gain shrinks to $1.02\text{--}1.16\times$ and $1.04\text{--}1.27\times$.

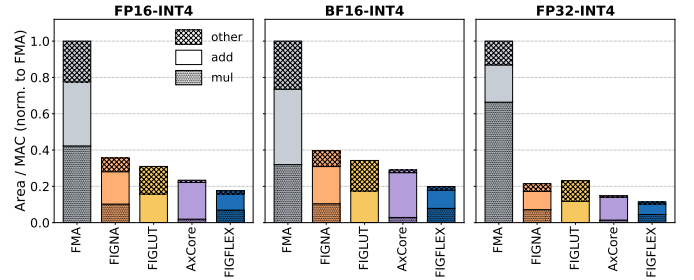


Fig. 11: Accelerator area breakdown (excluding SRAM) across the compared engines. FIGFLEX’s alignment and multiple-generation shifters are amortized at the shared array edge, leaving lean select/adder-tree PEs.

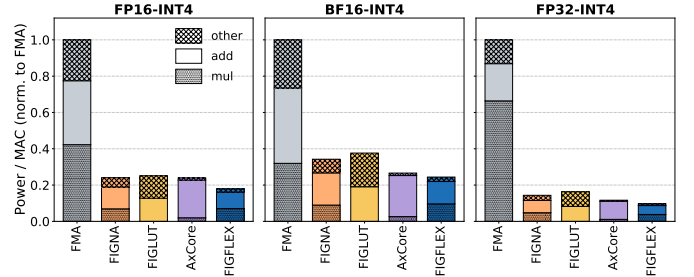


Fig. 12: Accelerator power breakdown (excluding SRAM) across the compared engines under the common iso-frequency budget.

Area and power breakdown. Figures 11 and 12 break down accelerator area and power excluding SRAM, across the compared engines; the SRAM, identical for every design (1.28 mm^2), is included in Table VII. The column-scale multiply of Subsection IV-B reuses the per-channel scale multiply that the activation producer already performs (e.g., RMSNorm’s γ), so it adds no area; its energy, one FP16 multiply (0.36 pJ) per activation element shared by all N output channels, is below 0.1% of the per-MAC energy at $N=4096$. Because FIGFLEX hoists the radix-4 generation logic to the shared per-row aligner and replicator, its per-PE cost reduces to a select/AND bank, a binary adder tree, and an integer accumulator, without adding arithmetic error. FIGFLEX is 25% smaller than AxCore, the state-of-the-art 4-bit quantized accelerator. While it has a larger multiplication unit, our significantly smaller integer accumulator yields less area overall. Note that FIGFLEX’s multiplication area is obtained by adding the selection muxes and the proportion of the binary adder tree that performs partial-product reduction for INT4 quantization (lvl. 0 adders). Comparing power consumption, FIGFLEX requires about 20–23% less power than the other baselines. FIGLUT has no multiplier; the majority of the power is consumed by the mux, output-generation logic, and bit-wise accumulation. AxCore is also equipped with an efficient multiplier approximator, which is essentially a fixed integer adder, and the majority of the computation power is consumed by the FP accumulator. The power breakdown of FIGFLEX is obtained similarly to the area breakdown.

VI. Related Work

A. Precision-Scalable DNN Accelerators

Bit-serial designs (Stripes [21], Bit-Pragmatic [2], UNPU [28]) process one bit-plane per cycle, so cost tracks precision but latency grows with it. *Bit-parallel composable* designs build wide operations from narrow ones: BitFusion [48] fuses 2-bit multipliers, BitBlade [44] amortizes their shifters across the array, and FlexiBit [49] generalizes this to arbitrary formats. FIGFLEX is bit-parallel and, like BitBlade, hoists the shift networks out of the PE, but it scales only the weight operand, allocates by radix-4 digit rather than by bit, and runs the intra- and inter-weight reduction on one integer adder tree after block-floating-point pre-alignment.

B. Beyond Weight-Only Quantization

Activation quantization [7, 15, 16, 29, 33, 59] and KV-cache quantization [18, 22, 36] attack the $\text{FP} \times \text{FP}$ attention operators and the memory-bound decode phase; they are complementary to the FP-INT linear GEMM that FIGFLEX accelerates. Codebook quantizers (SqueezeLLM [23], QuIP# [53], AQLM [9], VPTQ [34], QTIP [54]) and sparse-outlier formats (SpQR [8], OWQ [26]) reach strong quality at 2–3 bits but decode through a codebook or an index, which reintroduces a dequantization step or needs a LUT datapath.

C. Algorithm–Hardware Co-Design for Quantized LLMs

Mixed-precision PTQ (SliM-LLM [19], AMQ [30], LLM-MQ [31]) selects bit-widths for model quality alone, and precision-flexible accelerators [40, 44, 49] are evaluated on externally produced models; Tender [29] adapts the numeric format to the datapath but not the per-group allocation. FIGFLEX closes this loop: the allocator’s precision set $\{W_4, W_3, W_2, W_1\}$ is exactly the set the array executes at $32/b$ MAC/cycle.

VII. Conclusion

We presented FIGFLEX, an algorithm–hardware co-design that makes low-bit weight quantization pay off in computation as well as in memory. Its radix-4 FP-INT processing element is bit-parallel yet precision-scalable: every lane resolves two weight bits, so INT_4 to INT_1 weights run at a rate inversely proportional to their bit-width, and the datapath adds no error beyond shared-exponent alignment. With a compensation-aware mixed-precision quantizer, FIGFLEX attains 5.85 and 8.77 WikiText-2 perplexity at 3- and 2-bit average precision on LLaMA-2-7B, the lowest among the compared affine post-training quantizers. On the linear GEMM layers it delivers $1.56\times$ and $1.42\times$ the whole-chip throughput per area of AxCore [60] and FIGLUT [40] at 3 bits, which a first-order request model translates into $2.2\text{--}2.9\times$ (F3) and $2.9\text{--}3.7\times$ (F2) shorter batched requests than GPU low-bit kernels at the same bit-width and compute area.

References

- [1] [Online]. Available: <https://huggingface.co/datasets/Salesforce/wikitext>
- [2] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O’Leary, R. Genov, and A. Moshovos, “Bit-pragmatic deep neural network computing,” in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 382–394.
- [3] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, “Quarot: outlier-free 4-bit inference in rotated llms,” in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS ’24. Red Hook, NY, USA: Curran Associates Inc., 2024.
- [4] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi, “PIQA: Reasoning about physical commonsense in natural language,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 7432–7439.
- [5] J. Chee, Y. Cai, V. Kuleshov, and C. De Sa, “Quip: 2-bit quantization of large language models with guarantees,” in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS ’23. Red Hook, NY, USA: Curran Associates Inc., 2023.
- [6] 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.
- [7] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “Llm.int8(): 8-bit matrix multiplication for transformers at scale,” in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS ’22. Red Hook, NY, USA: Curran Associates Inc., 2022.
- [8] 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,” 2023. [Online]. Available: <https://arxiv.org/abs/2306.03078>
- [9] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, and D. Alistarh, “Extreme compression of large language models via additive quantization,” in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 2024, pp. 12 284–12 303. [Online]. Available: <https://proceedings.mlr.press/v235/egiazarian24a.html>
- [10] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “GPTQ: Accurate post-training quantization for generative pre-trained transformers,” in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: <https://openreview.net/forum?id=tcbBPnfwxS>
- [11] E. Frantar, R. L. Castro, J. Chen, T. Hoefler, and D. Alistarh, “MARLIN: Mixed-precision auto-regressive

- parallel inference on large language models,” in *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP 2025, Las Vegas, NV, USA, March 1-5, 2025*. ACM, 2025, pp. 239–251. [Online]. Available: <https://doi.org/10.1145/3710848.3710871>
- [12] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, “The Pile: An 800GB dataset of diverse text for language modeling,” *arXiv preprint arXiv:2101.00027*, 2020.
- [13] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. L. Noac’h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, “A framework for few-shot language model evaluation,” Zenodo, 2023.
- [14] A. Grattafiori and et.al., “The llama 3 herd of models,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.21783>
- [15] 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*, ser. ISCA ’23. ACM, 2023, p. 1–15. [Online]. Available: <http://dx.doi.org/10.1145/3579371.3589038>
- [16] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, “Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization,” in *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’22. IEEE Press, 2023, p. 1414–1433. [Online]. Available: <https://doi.org/10.1109/MICRO56248.2022.00095>
- [17] H. Guo, W. Brandon, R. Cholakov, J. Ragan-Kelley, E. P. Xing, and Y. Kim, “Fast matrix multiplications for lookup table-quantized llms,” in *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*. Association for Computational Linguistics, 2024, pp. 12 419–12 433. [Online]. Available: <https://aclanthology.org/2024.findings-emnlp.724>
- [18] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, “KVQuant: Towards 10 million context length LLM inference with KV cache quantization,” in *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. [Online]. Available: <https://openreview.net/forum?id=0LXotew9Du>
- [19] W. Huang, H. Qin, Y. Liu, Y. Li, Q. Liu, X. Liu, L. Benini, M. Magno, S. Zhang, and X. Qi, “Slim-llm: Saliency-driven mixed-precision quantization for large language models,” in *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, ser. Proceedings of Machine Learning Research, vol. 267. PMLR, 2025. [Online]. Available: <https://proceedings.mlr.press/v267/huang25aa.html>
- [20] J. Jang, Y. Kim, J. Lee, and J. Kim, “FIGNA: integer unit-based accelerator design for FP-INT GEMM preserving numerical accuracy,” in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2024, Edinburgh, United Kingdom, March 2-6, 2024*. IEEE, 2024, pp. 760–773. [Online]. Available: <https://doi.org/10.1109/HPCA57654.2024.00064>
- [21] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, “Stripes: Bit-serial deep neural network computing,” in *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016*. IEEE Computer Society, 2016, pp. 19:1–19:12.
- [22] H. Kang, Q. Zhang, S. Kundu, G. Jeong, Z. Liu, T. Krishna, and T. Zhao, “GEAR: An efficient KV cache compression recipe for near-lossless generative inference of LLM,” in *First Conference on Language Modeling, COLM 2024, Philadelphia, PA, USA, October 7-9, 2024*, 2024. [Online]. Available: <https://proceedings.mlr.press/v262/kang24a.html>
- [23] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, “Squeezellm: dense-and-sparse quantization,” in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML’24. JMLR.org, 2024.
- [24] Y. Kim, J. Jang, J. Lee, J. Park, J. Kim, B. Kim, B. Park, S. J. Kwon, D. Lee, and J. Kim, “Winning both the accuracy of floating point activation and the simplicity of integer arithmetic,” in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: <https://openreview.net/forum?id=z92lBy1ehJl>
- [25] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, “Efficient memory management for large language model serving with pagedattention,” in *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*. ACM, 2023, pp. 611–626.
- [26] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park, “Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models,” 2024. [Online]. Available: <https://arxiv.org/abs/2306.02272>
- [27] J. Lee, K. Cho, C. K. Lee, Y. Lee, J.-H. Park, S.-H. Oh, Y. Ju, C. Jeong, H. S. Cho, J. Lee, T.-S. Yun, J. H. Cho, S. Oh, J. Moon, Y.-J. Park, H.-S. Choi, I.-K. Kim, S. M. Yang, S.-Y. Kim, J. Jang, J. Kim, S.-H. Lee, Y. Jeon, J. Park, T.-K. Kim, D. Ka, S. Oh, J. Kim, J. Jeon, S. Kim, K. T. Kim, T. Kim, H. Yang, D. Yang, M. Lee, H. Song, D. Jang, J. Shin, H. Kim, C. Baek, H. Jeong, J. Yoon, S.-K. Lim, K. Y. Lee, Y. J. Koo, M.-J. Park, J. Cho, and J. Kim, “13.4 a 48gb 16-high 1280gb/s hbm3e dram with all-around power tsv and a 6-phase rdqs scheme for tsv area optimization,”

- in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 238–240.
- [28] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, “Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019.
- [29] J. Lee, W. Lee, and J. Sim, “Tender: Accelerating large language models via tensor decomposition and runtime requantization,” in *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, 2024, pp. 1048–1062. [Online]. Available: <https://doi.org/10.1109/ISCA59077.2024.00080>
- [30] S. Lee, S. Woo, J. Jin, C. Lee, and E. Park, “AMQ: enabling autml for mixed-precision weight-only quantization of large language models,” in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, Eds. Association for Computational Linguistics, 2025, pp. 35 532–35 550. [Online]. Available: <https://doi.org/10.18653/v1/2025.emnlp-main.1799>
- [31] S. Li, X. Ning, K. Hong, T. Liu, L. Wang, X. Li, K. Zhong, G. Dai, H. Yang, and Y. Wang, “Llm-mq: Mixed-precision quantization for efficient llm deployment.” [Online]. Available: <https://api.semanticscholar.org/CorpusID:281725511>
- [32] J. Lin, J. Tang, H. Tang, S. Yang, W. Chen, W. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, “AWQ: activation-aware weight quantization for on-device LLM compression and acceleration,” in *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*, P. B. Gibbons, G. Pekhimenko, and C. D. Sa, Eds. mlsys.org, 2024. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html
- [33] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, “Qserve: W4a8kv4 quantization and system co-design for efficient llm serving,” 2025. [Online]. Available: <https://arxiv.org/abs/2405.04532>
- [34] Y. Liu, J. Wen, Y. Wang, S. Ye, L. L. Zhang, T. Cao, C. Li, and M. Yang, “VPTQ: Extreme low-bit vector post-training quantization for large language models,” in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024, pp. 8181–8196.
- [35] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, “Spinquant: LLM quantization with learned rotations,” in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. [Online]. Available: <https://openreview.net/forum?id=ogO6DGE6FZ>
- [36] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, “KIVI: A tuning-free asymmetric 2bit quantization for KV cache,” in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 2024. [Online]. Available: <https://proceedings.mlr.press/v235/liu24bz.html>
- [37] K.-I. Moon, H.-Y. Son, and K. Lee, “Advanced packaging technologies in memory applications for future generative ai era,” in *2023 International Electron Devices Meeting (IEDM)*, 2023, pp. 1–4.
- [38] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0,” in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40)*, Chicago, Illinois, USA, December 1-5, 2007. IEEE Computer Society, 2007, pp. 3–14. [Online]. Available: <https://doi.org/10.1109/MICRO.2007.30>
- [39] NVIDIA, “CUTLASS: CUDA templates for linear algebra subroutines,” <https://github.com/NVIDIA/cutlass>, 2025, nVIDIA open-source GPU GEMM template library.
- [40] 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 *IEEE International Symposium on High Performance Computer Architecture, HPCA 2025, Las Vegas, NV, USA, March 1-5, 2025*. IEEE, 2025, pp. 1098–1111. [Online]. Available: <https://doi.org/10.1109/HPCA61900.2025.00085>
- [41] 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,” in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: <https://openreview.net/forum?id=gLARhFLE0F>
- [42] Y. Park, J. Hyun, S. Cho, B. Sim, and J. W. Lee, “Any-precision LLM: Low-cost deployment of multiple, different-sized llms,” in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 2024, pp. 39 682–39 701. [Online]. Available: <https://proceedings.mlr.press/v235/park24e.html>
- [43] 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.
- [44] S. Ryu, H. Kim, W. Yi, and J. Kim, “Bitblade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation,” in *Proceedings of*

- the 56th Annual Design Automation Conference, DAC 2019, Las Vegas, NV, USA, June 2-6, 2019. ACM, 2019.
- [45] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, “Winogrande: an adversarial winograd schema challenge at scale,” *Commun. ACM*, vol. 64, no. 9, p. 99–106, Aug. 2021. [Online]. Available: <https://doi.org/10.1145/3474381>
 - [46] Y. Shang, Z. Yuan, Q. Wu, and Z. Dong, “PB-LLM: Partially binarized large language models,” in *International Conference on Learning Representations (ICLR)*, 2024.
 - [47] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, “Omniquant: Omnidirectionally calibrated quantization for large language models,” in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: <https://openreview.net/forum?id=8Wuvhh0LYW>
 - [48] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, “Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 764–775.
 - [49] F. Tahmasebi, Y. Wang, B. Y. H. Huang, and H. Kwon, “Flexibit: Fully flexible precision bit-parallel accelerator architecture for arbitrary mixed precision AI,” *CoRR*, vol. abs/2411.18065, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2411.18065>
 - [50] G. Team, “Gemma 4 technical report,” 2026. [Online]. Available: <https://arxiv.org/abs/2607.02770>
 - [51] Together Computer, “RedPajama: An open source recipe to reproduce LLaMA training dataset,” <https://github.com/togethercomputer/RedPajama-Data>, 2023.
 - [52] H. Touvron and et.al., “Llama 2: Open foundation and fine-tuned chat models,” 2023. [Online]. Available: <https://arxiv.org/abs/2307.09288>
 - [53] A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. D. Sa, “Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks,” in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 2024, pp. 48 630–48 656. [Online]. Available: <https://proceedings.mlr.press/v235/tseng24a.html>
 - [54] A. Tseng, Q. Sun, D. Hou, and C. D. Sa, “QTIP: Quantization with trellises and incoherence processing,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
 - [55] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, “Qwen3 technical report,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.09388>
 - [56] Q. A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, “Qwen2.5 technical report,” 2025. [Online]. Available: <https://arxiv.org/abs/2412.15115>
 - [57] 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 (ACL)*, 2019, pp. 4791–4800.
 - [58] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, “Opt: Open pre-trained transformer language models,” 2022. [Online]. Available: <https://arxiv.org/abs/2205.01068>
 - [59] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, “Atom: Low-bit quantization for efficient and accurate llm serving,” 2024. [Online]. Available: <https://arxiv.org/abs/2310.19102>
 - [60] J. Zou, Y. Chen, X. Chen, C. Xu, and X. Chen, “Axcure: A quantization-aware approximate GEMM unit for LLM inference,” in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture, MICRO 2025, Seoul, Republic of Korea, October 18-22, 2025*. ACM, 2025, pp. 839–853. [Online]. Available: <https://doi.org/10.1145/3725843.3756094>