CURVEFP: CO-DESIGNING NUMERICAL REPRESENTATION AND PRODUCT ARITHMETIC FOR LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-precision formats usually optimize scalar fidelity while inheriting conventional product arithmetic. We introduce CURVEFP, a block-scaled family that distributes magnitudes across interleaved logarithmic curves. Uniform curve indices make every nonzero product an exact sign and integer-index update, while a rational radix exposes the finite phase schedule required for accumulation. We instantiate the algebra as CURVEFP8 E4C3/E5C2 for training and CURVEFP7 E3C3 for compact inference. On four 7B–9B models, CURVEFP7 beats tensorwise FP8 perplexity with one fewer element bit and stays within 1.32% of native quality. CURVEFP8 lowers error in all 36 paired training-GEMM comparisons. Across three matched 3B-token pretraining triplets, it reaches mean BF16-inference perplexity 22.5366 versus 22.5407 for FP8 and has a lower format penalty in every seed. Downstream evaluation shows transfer parity and a consistent WikiText-103 gain. In a preliminary 4×4 Nangate45 spatial tile, CURVEFP8 uses one fewer product register and 4.5% less area than timing-closing FP8 at 500 MHz. These results support CURVEFP as a numerical and arithmetic co-design, while leaving system-level efficiency to future study.

1 INTRODUCTION

Low-precision arithmetic has become a primary lever for scaling language-model training and inference, yet prevailing datatypes optimize how individual values are approximated while largely accepting the cost of their induced products. FP8 established that role-specific exponent and mantissa layouts can match 16-bit training quality (Micikevicius et al., 2022); microscaling formats then amortized range through per-block scales (Rouhani et al., 2023). Post-training methods attack the same numerical bottleneck from the data side, moving activation outliers into weights (Xiao et al., 2023) or rotating them away (Ashkboos et al., 2024; Liu et al., 2025). These advances make quantization substantially more accurate, but their element products still require a conventional integer or floating-point multiplier.

Product structure is an underused source of efficiency in low-precision design. Learned product/subspace codebooks offer flexible compression points (Wang et al., 2026; Qiao et al., 2026b), but do not by themselves impose a scalar closed-product algebra. Uniform integers simplify accumulation but depend increasingly on fine-grained scales as activation range widens. Logarithmic representations make multiplication additive, but coarse powers of two sacrifice local resolution and still require an accumulation strategy (Miyashita et al., 2016). Additive powers-of-two codes recover more levels through several shift-add terms (Li et al., 2020). These trade-offs motivate a stronger objective than scalar error alone: jointly design the level geometry and product algebra so that lower precision improves both representation density and arithmetic regularity.

We realize this objective with CURVEFP, a block-scaled closed-product codebook family. As illustrated in Figure 1, CURVEFP spreads quantized magnitudes across K interleaved logarithmic curves. Each nonzero element stores a sign, a two's-complement exponent, and a curve index under a shared power-of-two scale. Uniform curve spacing makes multiplication exact in the code domain: sign XOR, curve-index addition, and exponent addition with a carry. A reduced rational radix $r = 2^{p/q}$ tunes dynamic range against local precision, while the derived phase count $qK/\gcd(p, qK)$

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

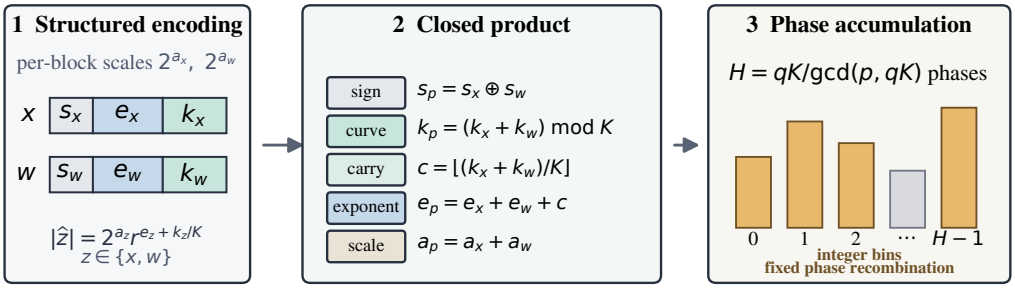


Figure 1: **CURVEFP co-designs representation and arithmetic.** Power-of-two scales provide block range; sign, exponent, and curve fields encode each element. Uniform curve indices make products exactly closed, and the rational radix determines the fixed-weight accumulation phases.

exposes the number of fractional exponent classes that accumulation must combine. Thus every range–resolution choice has an explicit accumulation contract, and every representable product lands on a known phase without lookup, projection, or a general product-forming multiplier.

One algebra supports two practical operating points. CURVEFP8 uses E4C3 for forward weights and activations and E5C2 for backward gradients, directly mirroring the range allocation of FP8 E4M3/E5M2 while replacing mantissa products with closed curve products. CURVEFP7 uses E3C3 to reduce the element width by 12.5% for deployment while retaining eight product phases. Block scaling supplies the local range that seven-bit activations need; compact binary scale exponents preserve the shift-and-count accumulation structure. A single numeric family therefore spans training and inference with the same closure rule.

The evidence spans model quality and arithmetic feasibility. Block-scaled CURVEFP7 beats tensor-wise FP8 perplexity on four 7B–9B models with one fewer element bit and remains within 1.32% of native quality. CURVEFP8 lowers error in all 36 paired forward and backward GEMM windows. Across three matched 3B-token triplets, its mean BF16-inference perplexity is 22.5366 versus 22.5407 for FP8, with a lower format-induced penalty in every seed; the seed variation supports parity rather than statistical superiority. A preliminary matched 4×4 spatial tile closes at 500 MHz with one fewer product register and 4.4% less area than FP8. Restoring that stage leaves CURVEFP8 3.9% smaller and provides additional setup slack, exposing a useful area–pipeline trade-off.

Our contributions are:

- We introduce a block-scaled codebook family distributed across rational-radix logarithmic curves, make its curve indices exactly closed under multiplication, and derive the finite phase count connecting representation choice to accumulation.
- We instantiate one algebra as CURVEFP8 for FP8-class training and CURVEFP7 for seven-bit deployment, with compact power-of-two scale metadata and no general multiplier in product formation.
- We provide end-to-end evidence spanning four-model inference, rational-radix and scale ablations, all three training GEMMs, three matched 3B-token runs, a complete downstream transfer study, and a preliminary routed spatial-tile comparison with FP8.

2 RELATED WORK

Low-precision datatypes. FP8 combines E4M3 forward operands with E5M2 gradients to retain training range and has become the principal eight-bit floating-point baseline (Micikevicius et al., 2022). Microscaling extends this approach by sharing scale factors across blocks of narrow floating-point or integer elements (Rouhani et al., 2023); hybrid block floating point likewise shares exponents

during DNN training but retains linear significands (Drumond et al., 2018). At the accelerator level, TeLLMe couples ternary weights with table-lookup matrix multiplication for end-to-end LLM prefill and decode on edge FPGAs (Qiao et al., 2026a).

Logarithmic neural-network arithmetic. Classical LNS encodes scalar magnitudes in the log domain, making multiplication additive; neural-network work has applied it to quantized CNNs/LSTMs, accelerator arithmetic, and low-precision training with multiplicative updates (Miyashita et al., 2016; Kouretas & Paliouras, 2018; Christ et al., 2022; Zhao et al., 2022). Flattening CurveFP's exponent and curve fields into $n = Ke + k$ gives a finite block-scaled LNS lattice, which explains the shared additive product algebra. CurveFP uses that algebra through a different datatype interface: an explicit multi-curve codebook, compact scale contracts, the $H = qK/\gcd(p, qK)$ phase law, and role-specific LLM formats trained with conventional AdamW. APoT instead improves resolution through multiple shift-add terms (Li et al., 2020).

LLM quantization and structured codebooks. Modern LLM post-training quantization often reshapes tensors before applying a conventional datatype. SmoothQuant migrates activation outliers into weights (Xiao et al., 2023); QuaRot and SpinQuant use fixed or learned rotations to suppress outliers (Ashkboos et al., 2024; Liu et al., 2025). Learned representations offer a complementary route: LO-BCQ clusters blocks and assigns each cluster an optimized scalar codebook (Elangovan et al., 2025), while NestQuant uses self-similar nested lattices for low-error matrix products (Savkin et al., 2025). These methods target aggressive W4A4 inference and can exceed simple scalar formats in accuracy. Product/subspace quantization follows another path: APEX-Q permits arbitrary subvector dimensions, while FASQ offers calibration-free flexible compression points and custom GPU kernels (Wang et al., 2026; Qiao et al., 2026b). Our objective is different: CURVEFP parameterizes its levels with a rational radix and uniform curve indices, obtaining a compact closed-product algebra shared by inference and training. This structure reaches the FP8 quality regime without calibration-time transformations.

3 CURVEFP

Overview. CURVEFP starts from a closed-product codebook view: distribute values across structured curves so representation quality and product composition improve together. We first define the rational-radix element code, then derive closed product formation and phase-structured accumulation, and finally instantiate role-specific formats for training and compact inference.

3.1 A RATIONAL-RADIX DATATYPE FAMILY

CURVEFP represents a tensor block with one power-of-two scale and represents each element by a sign, an exponent, and a curve index. Let E and C denote the exponent and curve-index widths, let $K = 2^C$, and choose a reduced rational radix $r = 2^{p/q}$. A nonzero quantized value in block b is

$$\widehat{x} = (-1)^s 2^{a_b} r^{e+k/K}, \qquad e \in \{-2^{E-1}, \dots, 2^{E-1} - 1\}, \quad k \in \{0, \dots, K-1\}, \tag{1}$$

where a_b is a signed shared-scale exponent. The element therefore occupies $1 + E + C$ bits. We reserve magnitude index zero as a sentinel and omit the smallest nonzero exponent/curve combination, leaving $2^{E+C} - 1$ positive magnitudes; values outside the scaled range map to an endpoint. Unlike a learned codebook, the levels require no table of arbitrary constants: adjacent curve indices differ by the fixed ratio $r^{1/K}$, and adjacent exponents differ by r.

Given a_b, quantization preserves the source sign and selects the magnitude whose reconstructed value is nearest in absolute error, with saturation at the endpoints. Static operands choose a representable power-of-two scale by reconstruction-MSE search; dynamic training operands use ceil-absmax scaling. We write element-wise tensor quantization as $Q(\boldsymbol{X})$, so $Q(\boldsymbol{X})_{ij} = \widehat{x}_{ij}$.

The rational radix parameterizes the spacing of this uniform logarithmic lattice. Setting $p = q = 1$ gives levels $2^{e+k/K}$. Choosing $p/q < 1$ contracts the range and increases local resolution without changing the element width, whereas $p/q > 1$ expands the range. CURVEFP associates every spacing with the phase count derived below, exposing a direct quality–complexity control.

3.2 CLOSED PRODUCTS AND PHASE-STRUCTURED ACCUMULATION

Uniform curve indices make the product codebook exactly closed. For two nonzero operands with curve indices i and j, define

$$\ell = (i + j) \bmod K, \qquad c = \left\lfloor \frac{i + j}{K} \right\rfloor. \tag{2}$$

Their product has sign $s_x \oplus s_w$, shared-scale exponent $a_x + a_w$, element exponent $e_x + e_w + c$, and curve index ℓ. Product formation thus uses an XOR and small integer additions, with no general variable-by-variable multiplier and no nearest-code projection. A zero sentinel short-circuits the product to zero without applying the index update. This closed product path feeds a fixed phase reduction with binary scale transport; Section 5 summarizes its deployment implications.

Closure holds on widened product coordinates. Requantizing to a finite E-bit destination may saturate; no codebook projection is needed before that explicit output requantization.

Rational radices preserve this closure while changing the number of binary phases needed by a dot product. From Equation 1, every unscaled magnitude is

$$r^{e+k/K} = 2^{p(eK+k)/(qK)}. \tag{3}$$

Let $N = qK$. As the integer $n = eK + k$ advances, the fractional exponent is the residue $pn \bmod N$ divided by N. The additive subgroup generated by p in $\mathbb{Z}_N$ has order

$$H = \frac{qK}{\gcd(p, qK)} \tag{4}$$

and therefore defines H phase classes on the underlying lattice. A finite format may leave some classes empty, but every dot product can be scheduled over these H phases and written as

$$\widehat{\boldsymbol{x}}^{\mathsf{T}} \widehat{\boldsymbol{w}} = \sum_{h=0}^{H-1} 2^{h/H} \sum_{t} n_{h,t} 2^{t}, \tag{5}$$

where each $n_{h,t}$ is a signed integer count. Products are routed exactly to integer counts indexed by binary shift; only the H fixed phase weights remain after reduction.

Constructive arithmetic realization. Let $g = \gcd(p, qK)$, $n_x = e_x K + k_x$, $n_w = e_w K + k_w$, and $u = p(n_x + n_w)$. Each nonzero lane increments $A_{h,t}$ by $\delta = (-1)^{s_x \oplus s_w}$ at $h = (u \bmod qK)/g$ and $t = a_x + a_w + \lfloor u/(qK) \rfloor$; zero suppresses the update. For binary radix, $h = (k_x + k_w) \bmod K$ and $t = a_x + a_w + e_x + e_w + \lfloor (k_x + k_w)/K \rfloor$. Product formation therefore uses only zero detection, sign XOR, one $(C + 1)$-bit curve-index addition, and signed exponent addition. A length-D dot needs $1 + \lceil \log_2(D + 1) \rceil$-bit signed counters. Equal addresses can be merged with signed popcounts in a histogram design, or shifted directly into one wide accumulator per phase. Both layouts produce the same phase sums before the fixed-coefficient reduction and final requantization. Appendix A.1 gives the subgroup proof and full state bound.

3.3 SHARED SCALES, OPERATING POINTS, AND TRAINING PROTOCOL

Shared scales let the element code focus on within-block variation while amortizing range metadata. For a block of G values and an S-bit scale exponent, the effective storage rate is

$$R = 1 + E + C + \frac{S}{G} \quad \text{bits per value.} \tag{6}$$

Because 2^{a_b} is itself binary, it shifts the exponent-bin index without introducing another product phase. We evaluate tensorwise, rowwise, and fixed-size block scaling, but use block scaling as the compact deployment contract because activations require local range adaptation.

Table 1 gives the principal operating points. The eight-bit training pair follows the familiar split between a precision-oriented forward format and a range-oriented gradient format. CURVEFP8 E4C3 quantizes weights and activations in the forward linear layers, while E5C2 quantizes the output gradients used by both activation- and weight-gradient matrix products. Moving one bit from C to E

Table 1: Principal CURVEFP formats. Positive levels exclude zero; H is the phase count from Equation 4.

Format	Bits	E	C	Positive	H	Role
CURVEFP7 E3C3	7	3	3	63	8	Compact inference
CURVEFP8 E4C3	8	4	3	127	8	Forward training
CURVEFP8 E5C2	8	5	2	127	4	Backward gradients

preserves 127 positive levels, expands gradient range, and reduces the phase count from eight to four; this mirrors the precision-oriented E4M3/range-oriented E5M2 FP8 split. Parameters and optimizer states remain full precision, and nonlinear operations execute in bfloat16. CURVEFP7 E3C3 keeps eight curve phases in a seven-bit element for compact post-training deployment.

Training quantizes the operands of every non-head linear layer. The forward pass evaluates $Q_A(\boldsymbol{X})Q_W(\boldsymbol{W})^\mathsf{T}$. The backward pass reuses the quantized forward operands and quantizes the output gradient $Q_G(\nabla \boldsymbol{Y})$, yielding $Q_G(\nabla \boldsymbol{Y})Q_W(\boldsymbol{W})$ for $\nabla \boldsymbol{X}$ and $Q_G(\nabla \boldsymbol{Y})^\mathsf{T}Q_A(\boldsymbol{X})$ for $\nabla \boldsymbol{W}$. E4C3 supplies forward weights and activations, while E5C2 supplies output gradients to both backward GEMMs. Parameters, optimizer states, nonlinear operations, and the tied output head remain at higher precision.

4 EXPERIMENTS

4.1 EXPERIMENTAL DESIGN

We evaluate CURVEFP across post-training inference, all three training GEMMs, matched from-scratch pretraining, and downstream format transfer. The baselines are BF16/FP16, standard FP8 E4M3/E5M2 (Micikevicius et al., 2022), microscaling formats, and signed integers with practical real-valued scales. Unless stated otherwise, we quantize every linear layer except the tied vocabulary head and keep normalization and nonlinear operations in BF16/FP16. Every comparison shares data, tokenizer, model weights or initialization, and evaluation code; reported values come from saved artifacts. Appendix B provides the complete protocols.

4.2 CURVEFP7 PROVIDES AN INT8-CLASS INFERENCE ALTERNATIVE AT SEVEN BITS

We first quantize weights and activations of Llama-3-8B, Qwen3-8B, Qwen3.5-9B, and Falcon-H1-7B (Grattafiori et al., 2024; Yang et al., 2025; Qwen Team, 2026; Zuo et al., 2025). Every run scores the complete WikiText-2 test split with the same 2,048-token rolling-loglikelihood protocol. Within a model, methods share weights, output-head exclusion, and K/V precision; Llama and Qwen3 keep dense K/V at common tensorwise FP8 to isolate weight/activation arithmetic. We compare tensorwise and rowwise FP8 E4M3 with CURVEFP7 E3C3 under compact binary scales. Including scales, G64 costs 7.062 weight and 7.078 activation bits per value; G512 reduces these rates to 7.008 and 7.010.

At G64, CURVEFP7 beats tensorwise FP8 on all four models and remains within 1.32% perplexity of native. Against rowwise FP8 it wins on Llama and Qwen3.5 and trails by less than 0.016 PPL on Qwen3 and Falcon-H1. Figure 2b shows the broader scaling trend: INT8 leads narrowly at G128/G256, CURVEFP7 crosses at G512, and its advantage grows under rowwise and tensorwise sharing. The seven-bit format therefore retains FP8-class quality while reducing element width and preserving closed-product arithmetic.

4.3 RATIONAL RADIX VALIDATES THE QUALITY–PHASE CONTROL

All principal formats use binary radix. To test the rational generalization, we vary only the E4C3 radix while holding element width, G512 blocks, and scale count fixed. A near-binary point adds one predicted phase and improves all four models slightly. A denser point reaches the best perplexity but needs 32 phases for only 0.0055–0.0259 lower PPL than binary radix. This confirms the quality–phase law and motivates the eight-phase operating point; full results are shown in Table 2.

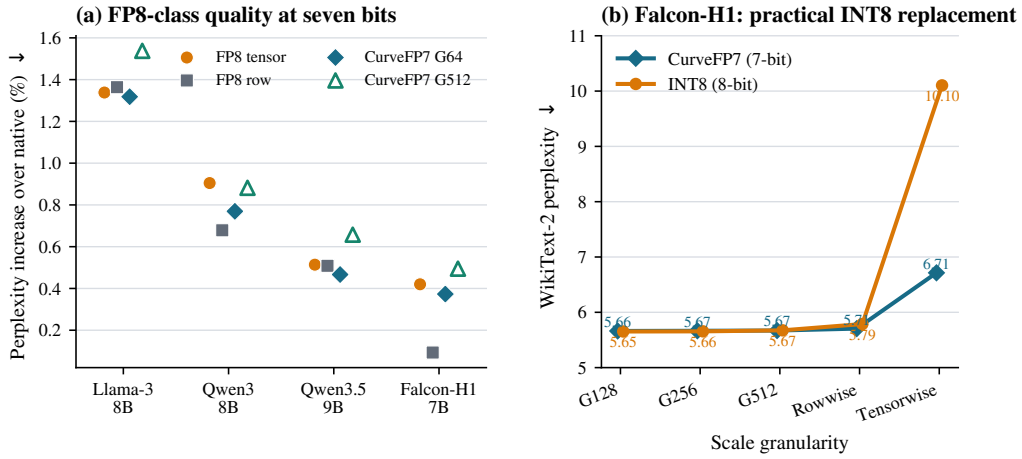


Figure 2: **Seven-bit CurveFP7 reaches the FP8 quality regime.** Left: perplexity increase over native on four models. Right: the Falcon-H1 granularity sweep, where CurveFP7 crosses INT8 at G512.

Table 2: E4C3 WikiText-2 perplexity at G512 as radix trades phase count for local resolution. Lower is better.

Model	$r = 2$ 8 phases	$r = 2^{8/9}$ 9 phases	$r = 2^{3/4}$ 32 phases
Llama-3-8B	6.2160	6.2052	**6.1901**
Qwen3-8B	7.0536	7.0426	**7.0420**
Qwen3.5-9B	7.0159	7.0070	**7.0034**
Falcon-H1-7B	5.6654	5.6652	**5.6599**

4.4 CurveFP8 reduces error in all three training GEMMs

Before pretraining, we capture real forward XW^{T}, activation-gradient $\nabla Y W$, and weight-gradient $\nabla Y^{\mathsf{T}} X$ operands from Pythia-410M (Biderman et al., 2023) and Llama-3.2-3B (Grattafiori et al., 2024). Each window is a disjoint 64-token segment and one real backward pass; three layers per linear role are spaced through model depth. Under identical tensors, CurveFP8 lowers NMSE in all 36 paired comparisons and by 8.2–11.1% in aggregate (Table 3). The gain therefore covers the complete linear-layer training lifecycle rather than only the forward product.

Table 3: CurveFP8/FP8 NMSE on paired training-GEMM windows. Values below one favor CurveFP8; all 36 comparisons are wins.

Model	Windows	Forward	∇X	∇W
Pythia-410M	8	0.8927	0.8887	0.9012
Llama-3.2-3B	4	0.9044	0.8952	0.9180

4.5 CurveFP8 completes three matched 3B-token runs

The decisive test trains three matched BF16/FP8/CurveFP8 triplets: nine 128.3M-parameter Llama-style decoders, each on 2,999,943,168 FineWeb-Edu tokens (Penedo et al., 2024). The model has 12 layers, width 768, intermediate width 2560, 12 attention heads, and four K/V heads. We train for 30,517 updates at sequence length 1024 and global batch size 96 using AdamW, a 1,000-step warmup, and cosine decay. Modes within a triplet share initialization, token order, optimizer, and schedule. FP8 uses E4M3/E5M2; CurveFP8 uses E4C3/E5C2 with power-of-two scales. Parameters and optimizer states remain FP32, while nonlinear operations use BF16.

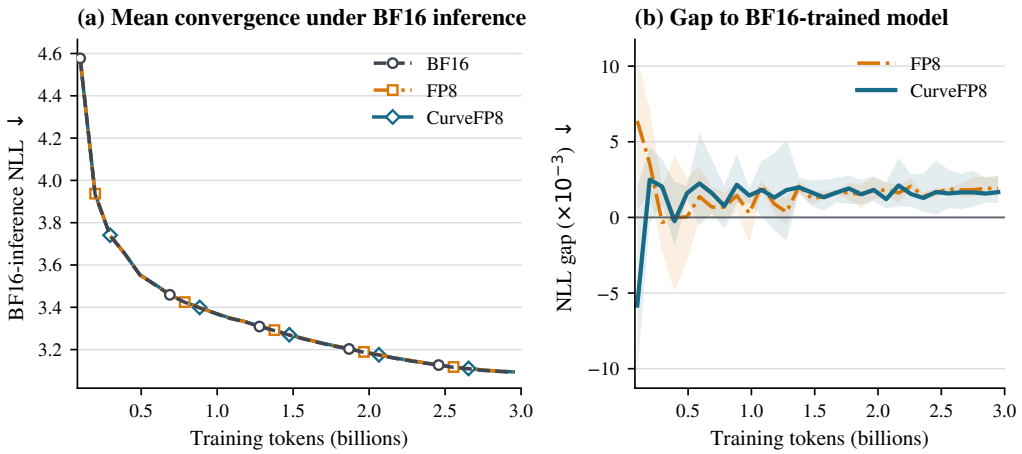


Figure 3: **CURVEFP8 and FP8 track BF16 across three seeds.** Left: BF16 evaluation isolates learned checkpoint quality. Right: each quantized mode after subtracting seed-matched BF16. Lines are seed means; bands span the seed range.

We evaluate every checkpoint in two ways. First, each model runs in the arithmetic used during training: BF16, FP8, or CURVEFP8. This is the deployment result. Second, we run the same learned weights in BF16 for all three models. This common-arithmetic comparison reveals which training mode learned the better checkpoint. Within each row of Table 4, the gap between these two evaluations is the inference quantization loss.

Table 4: Held-out perplexity after matched 3B-token pretraining. Values are three-seed mean ± sample standard deviation.

Training mode	Training-format PPL ↓	BF16 PPL ↓
BF16	22.4899 ± 0.0198	22.4899 ± 0.0199
FP8	22.6164 ± 0.0130	22.5407 ± 0.0182
CURVEFP8	**22.6054** ± 0.0199	**22.5366** ± 0.0207

The low-precision endpoints are statistically unresolved. CurveFP8-minus-FP8 perplexity is -0.0110 ± 0.0217 in training arithmetic and -0.0040 ± 0.0206 under BF16 inference, with CURVEFP8 winning two of three seeds in both views. The format-induced penalty is more consistent: 0.0688 versus 0.0758 PPL, lower for CURVEFP8 in every seed. The trajectories also overlap, and all nine lanes complete without divergence or a non-finite update. These results establish full-run FP8-class training parity, not statistical superiority.

4.6 CURVEFP8 TRANSFERS ACROSS SEEDS AND RUNTIME FORMATS

We next evaluate WikiText-103 and PG-19 OOD language modeling plus zero- and five-shot suites spanning HellaSwag, PIQA, ARC, WinoGrande, LAMBADA, and BLiMP. Every cell shares the tokenizer, 1,024-token context, and lm-evaluation-harness protocol; Appendix B.3 gives task and bootstrap details plus the full training/runtime matrix.

Table 5: Final checkpoints benchmarks evaluation. Values are three-seed mean ± sample standard deviation; task values are percentages.

Trained	WikiText PPL	PG-19 PPL	Zero-shot	Five-shot	BLiMP
BF16	40.384 ± .071	43.728 ± .609	**40.128 ± .164**	**43.096 ± .531**	79.786 ± .343
FP8	40.487 ± .034	43.886 ± .669	39.713 ± .299	42.446 ± .563	79.597 ± .298
CURVEFP8	**40.083 ± .010**	**43.132 ± .528**	39.906 ± .214	42.629 ± .118	**79.938 ± .201**

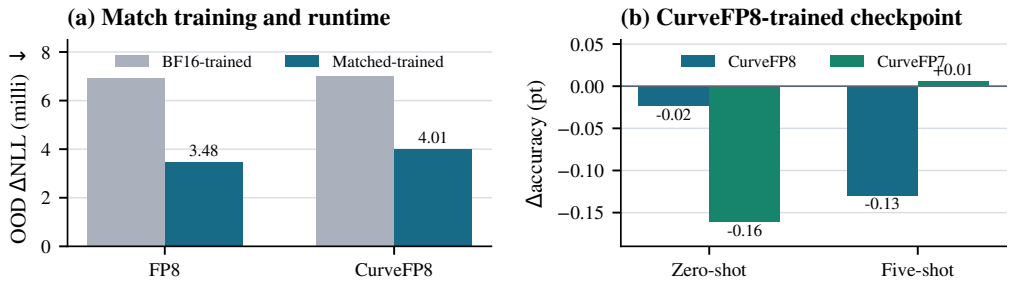


Figure 4: **Matching training and runtime reduces format-transfer loss.** Left: OOD penalty for BF16-trained and matched-trained checkpoints. Right: zero- and five-shot changes for CURVEFP8 and CURVEFP7 on the CURVEFP8-trained checkpoint.

Table 5 evaluates learned checkpoint quality under a common BF16 runtime. CURVEFP8 has the best mean perplexity on both OOD corpora and the best BLiMP mean, while BF16 leads the aggregate zero- and five-shot suites. Figure 4 then isolates runtime format sensitivity: matching training and runtime nearly halves the OOD penalty for both FP8 and CURVEFP8, and the CURVEFP8-trained checkpoint transfers cleanly to seven-bit deployment. Across runtime formats, CURVEFP8 improves WikiText-103 in all 12 paired comparisons; PG-19 and task outcomes are mixed. This supports transfer parity with one consistent OOD advantage, not broad downstream superiority.

5 ARITHMETIC REALIZATION AND HARDWARE SCOPE

The contribution is primarily algorithmic: the element fields compose exactly, and Equation 4 turns each radix choice into an explicit accumulation contract. Sign XOR, curve-index addition, carry, exponent addition, and binary scale transport determine every nonzero product without lookup or projection. Products then route to one of H phases and one binary bin, with counter width fixed by dot length. A banked histogram can merge equal addresses through signed popcounts; a per-phase Kulisch layout shifts updates directly into wide accumulators. Both realize the same semantics before fixed-coefficient phase reduction. Thus the datatype fixes operations, state bounds, and outputs while leaving lane count, bank organization, and pipeline placement to the architecture. Appendix A gives the full derivation.

Tensor-core-like spatial architecture. A tile is a small matrix-multiplication block intended to be replicated inside an accelerator. To test whether the closed product maps to useful parallel arithmetic, we build such a tile rather than a scalar dot-product unit. It computes a 4×4 output block $C_{i,j} = \sum_{k=0}^{K-1} A_{i,k} B_{k,j}$ as a stream of outer products over the reduction dimension K. On each cycle it accepts four 8-bit values $A_{0:3,k}$ and four 8-bit values $B_{k,0:3}$, broadcasts each A value across one row and each B value down one column, and forms all 16 products $A_{i,k}B_{k,j}$. The 16 processing elements each retain one running output $C_{i,j}$ locally; this output-stationary dataflow avoids reducing the products to a single scalar. A start signal clears the partial sums for a new output block, a last signal captures them after K cycles, and the completed matrix is read out one row at a time. Both tiles can accept one outer product per cycle, so their common peak throughput is 16 multiply–accumulate operations (MACs) per cycle.

The comparison holds the surrounding arithmetic constant. Both tiles consume the same eight input bytes per cycle, use the same matrix controls and row-drain interface, and accumulate without intermediate rounding into 16 signed 64-bit fixed-point registers with 32 fractional bits (Q32). This common accumulator isolates product hardware; it is not intended to reproduce a commercial tensor core's FP32 accumulator. CurveFP forms a product by adding the two magnitude ranks, selecting one fixed phase coefficient, applying the sign, and shifting the result into Q32. The FP8 reference instead decodes E4M3FN fields, inserts the hidden bit or handles a subnormal, multiplies the significands, applies the sign, and aligns the result into the same Q32 accumulator. Thus the experiment isolates the datatype-specific product path; memories, tensor scales, and final output conversion are excluded from both designs.

We independently test two pipeline choices for each datatype. P0 connects product formation directly to the accumulator; P1 inserts one register between them. That register shortens the combinational path and adds one cycle of result latency, but both variants still accept a new outer product every cycle. We place and route all four candidates under the same 2.0 ns clock and I/O constraints, then select the shallowest variant that closes timing. Setup slack is required arrival time minus actual arrival time after routing: positive slack means data arrives before the clock deadline, whereas negative slack is a timing violation. This criterion compares equal semantics and throughput without forcing CurveFP to carry a register that its product path may not need.

We placed and routed these tiles in an open Nangate45 flow (Table 6). At 500 MHz, CURVEFP8 closes without a product register, whereas FP8 requires one; the resulting CURVEFP8 tile is 4.6% smaller at equal peak throughput. Adding the register produces the P1 operating point. When both designs use P1, CURVEFP8 remains 4.1% smaller and has 0.426 ns setup slack versus 0.302 ns for FP8. This extra positive margin suggests headroom for a tighter timing target, hint for higher clock speed. These measurements are a feasibility check rather than a complete accelerator comparison: memories, interconnect, scale handling, boundary conversion, and workload-dependent utilization remain outside both tiles. The complete pipeline sweep, RTL verification, open tool flow, and limitations are in Appendix C.

Table 6: Preliminary 4×4 spatial-tile results at 500 MHz. P0/P1 denotes zero/one product register; the appendix gives the full sweep.

Tile	Product reg.	Area (μm^2)	Setup slack (ns)	pJ/MAC
CURVEFP8 E4C3 P0	**0**	**49,282.9**	+0.064	4.78
CURVEFP8 E4C3 P1	1	49,540.1	**+0.426**	4.78
FP8 E4M3FN P0	0	53,245.5	−0.045	–
FP8 E4M3FN P1	1	51,646.8	+0.302	4.75

6 CONCLUSION

CURVEFP shows that low-precision representation and product algebra can be designed together. Its interleaved logarithmic curves give exact index-domain products, and the phase law makes accumulation cost explicit. CURVEFP7 reaches the FP8 quality regime at seven bits; CURVEFP8 improves every paired training-GEMM diagnostic and matches FP8 across three 3B-token training triplets. The learned checkpoints retain downstream capability across runtime formats. A preliminary routed tile adds a hardware hint: at 500 MHz, CURVEFP8 is 4.6% smaller and one product stage shallower than timing-closing FP8, or 4.1% smaller with the stage restored for additional timing slack. Memory, interconnect, scale transport, and workload effects remain open. Thus the present evidence supports a coherent FP8-class training path, a 7-bit inference point, and an area-efficient closed-product realization.

REPRODUCIBILITY STATEMENT

Section 3 defines the formats, quantization rule, product mapping, and phase count. Section 4 reports the model, data, optimizer, task, and precision contracts used in each principal experiment. The supplementary code fixes the seed and data contracts, implements resumable checkpointing and atomic artifact generation, validates every matrix cell, and regenerates result summaries and figures from produced artifacts. Section 5 and Appendix C report the preliminary spatial-tile realization, verification, and open tool flow.

AI USE STATEMENT

We used generative AI tools to assist with experiment scripts, derivation checking, figure preparation, literature review, and language editing.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, and Torsten Hoefler. QuaRot: Outlier-free 4-bit inference in rotated LLMs. *arXiv preprint arXiv:2404.00456*, 2024. doi: 10.48550/arXiv.2404.00456. URL https://arxiv.org/abs/2404.00456.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*, 2023. doi: 10.48550/arXiv.2304.01373. URL https://arxiv.org/abs/2304.01373.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1911.11641.

Maxime Christ, Florent de Dinechin, and Frédéric Pétrot. Low-precision logarithmic arithmetic for neural network accelerators. In *2022 IEEE 33rd International Conference on Application-Specific Systems, Architectures and Processors*, pp. 72–79, 2022. doi: 10.1109/ASAP54787.2022.00021. URL https://doi.org/10.1109/ASAP54787.2022.00021.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. doi: 10.48550/arXiv.1803.05457. URL https://arxiv.org/abs/1803.05457.

Mario Drumond, Tao Lin, Martin Jaggi, and Babak Falsafi. Training DNNs with hybrid block floating point. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://arxiv.org/abs/1804.01526.

Reena Elangovan, Charbel Sakr, Anand Raghunathan, and Brucek Khailany. LO-BCQ: Block clustered quantization for 4-bit (W4A4) LLM inference. *Transactions on Machine Learning Research*, 2025. doi: 10.48550/arXiv.2502.05376. URL https://arxiv.org/abs/2502.05376.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/arXiv.2407.21783. URL https://arxiv.org/abs/2407.21783.

Ioannis Kouretas and Vassilis Paliouras. Logarithmic number system for deep learning. In *2018 7th International Conference on Modern Circuits and Systems Technologies*, pp. 1–4, 2018. doi: 10.1109/MOCAST.2018.8376572. URL https://doi.org/10.1109/MOCAST.2018.8376572.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1909.13144.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: LLM quantization with learned rotations. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2405.16406.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. doi: 10.48550/arXiv.1609.07843. URL https://arxiv.org/abs/1609.07843.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022. doi: 10.48550/arXiv.2209.05433. URL https://arxiv.org/abs/2209.05433.

Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016. doi: 10.48550/arXiv.1603.01025. URL https://arxiv.org/abs/1603.01025.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel 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)*, pp. 1525–1534. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024. doi: 10.48550/arXiv.2406.17557. URL https://arxiv.org/abs/2406.17557.

Ye Qiao, Zhiheng Chen, Yifan Zhang, Yian Wang, and Sitao Huang. TeLLMe: An efficient end-to-end ternary LLM prefill and decode accelerator with table-lookup matmul on edge FPGAs. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 247–257. ACM, 2026a. doi: 10.1145/3748173.3779191. URL https://doi.org/10.1145/3748173.3779191.

Ye Qiao, Yian Wang, Zhiheng Chen, Hyoukjun Kwon, and Sitao Huang. FASQ: Flexible accelerated subspace quantization for calibration-free LLM compression. *arXiv preprint arXiv:2605.04084*, 2026b. doi: 10.48550/arXiv.2605.04084. URL https://arxiv.org/abs/2605.04084.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019. doi: 10.48550/arXiv.1911.05507. URL https://arxiv.org/abs/1911.05507.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023. doi: 10.48550/arXiv.2310.10537. URL https://arxiv.org/abs/2310.10537.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019. doi: 10.48550/arXiv.1907.10641. URL https://arxiv.org/abs/1907.10641.

Semyon Savkin, Eitan Porat, Or Ordentlich, and Yury Polyanskiy. NestQuant: Nested lattice quantization for matrix products and LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 53042–53062. PMLR, 2025. URL https://proceedings.mlr.press/v267/savkin25a.html.

Yian Wang, Ye Qiao, Sitao Huang, and Hyoukjun Kwon. APEX-Q: Arbitrary-dimension product-extension quantization for accelerated LLM deployment (student abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(48):41424–41426, 2026. doi: 10.1609/aaai.v40i48.42293. URL https://doi.org/10.1609/aaai.v40i48.42293.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for english. *Transactions of the Association for Computational Linguistics*, 8:377–392, 2020. doi: 10.1162/tacl_a_00321. URL https://doi.org/10.1162/tacl_a_00321.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 2023. URL https://proceedings.mlr.press/v202/xiao23c.html.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Jiawei Zhao, Steve Dai, Rangharajan Venkatesan, Brian Zimmer, Mustafa Ali, Ming-Yu Liu, Brucek Khailany, William J. Dally, and Anima Anandkumar. LNS-Madam: Low-precision training in logarithmic number system using multiplicative weight update. *IEEE Transactions on Computers*, 71(12):3179–3190, 2022. doi: 10.1109/TC.2022.3202747. URL https://doi.org/10.1109/TC.2022.3202747.

Jingwei Zuo et al. Falcon-H1: A family of hybrid-head language models redefining efficiency and performance. *arXiv preprint arXiv:2507.22448*, 2025. URL https://arxiv.org/abs/2507.22448.

A Algorithmic Details

A.1 Closed-product addressing and accumulator bounds

Let $g = \gcd(p, qK)$, $n_x = e_x K + k_x$, $n_w = e_w K + k_w$, and $u = p(n_x + n_w)$. Each nonzero product contributes

$$\delta = (-1)^{s_x \oplus s_w}, \qquad h = \frac{u \bmod qK}{g}, \qquad t = a_x + a_w + \left\lfloor \frac{u}{qK} \right\rfloor \tag{7}$$

to integer bin $A_{h,t}$; the zero sentinel suppresses the update. Multiples of g form the additive subgroup generated by p in $\mathbb{Z}_{qK}$, whose order is $H = qK/g$. This proves that all products fall into the H phases used in Equation 5.

For binary radix, $p = q = 1$, so

$$h = (k_x + k_w) \bmod K, \qquad t = a_x + a_w + e_x + e_w + \left\lfloor \frac{k_x + k_w}{K} \right\rfloor. \tag{8}$$

The lane logic is therefore zero detection, sign XOR, one $(C+1)$-bit curve addition, and signed exponent addition. A length-D dot requires $1 + \lceil \log_2(D+1) \rceil$-bit signed counters. Equal addresses can be merged with signed popcounts in a histogram implementation, or shifted directly into one wide accumulator per phase. Both layouts produce the same phase sums before the fixed-coefficient reduction and final requantization.

A.2 Shared scales and training operators

A block of G elements with an S-bit power-of-two scale has effective rate

$$R = 1 + E + C + \frac{S}{G} \quad \text{bits per value.} \tag{9}$$

Static operands select a representable scale by reconstruction-MSE search; dynamic training operands use ceil-absmax scaling. Because the scale is binary, it changes only the exponent-bin address and introduces no product phase.

For each non-head linear layer, the forward pass evaluates $Q_A(\boldsymbol{X})Q_W(\boldsymbol{W})^\mathsf{T}$. The backward pass reuses the quantized forward operands and quantizes the output gradient, giving $Q_G(\nabla \boldsymbol{Y})Q_W(\boldsymbol{W})$ for $\nabla \boldsymbol{X}$ and $Q_G(\nabla \boldsymbol{Y})^\mathsf{T} Q_A(\boldsymbol{X})$ for $\nabla \boldsymbol{W}$. Parameters, optimizer states, normalization, nonlinear operations, and the tied output head remain at higher precision.

B Experimental Details

B.1 Inference and GEMM diagnostics

Post-training quantization uses the complete WikiText-2 test split (Merity et al., 2016) and a 2,048-token rolling-loglikelihood protocol. Within each model, all methods share weights, output-head exclusion, and K/V precision. Llama-3 and Qwen3 keep dense K/V at common tensorwise FP8 so the comparison isolates weight/activation arithmetic.

The training-GEMM diagnostic records forward XW^T, activation-gradient $\nabla Y W$, and weight-gradient $\nabla Y^\mathsf{T} X$ operands from Pythia-410M and Llama-3.2-3B. Each window is a disjoint 64-token WikiText-2 segment with one real backward pass. Three layers per linear role are spaced through model depth, yielding 12 sampled Pythia linears and 21 sampled Llama linears per window. CurveFP8 lowers aggregate NMSE by 8.2–11.1% and wins all 36 paired model/window/GEMM comparisons reported in Table 3.

B.2 Pretraining protocol

The matched experiment trains three independent BF16/FP8/CurveFP8 triplets. Each lane is a 12-layer, 128.3M-parameter Llama-style decoder with width 768, intermediate width 2560, 12 attention heads, and four key/value heads. Runs use 2,999,943,168 FineWeb-Edu tokens (Penedo et al., 2024),

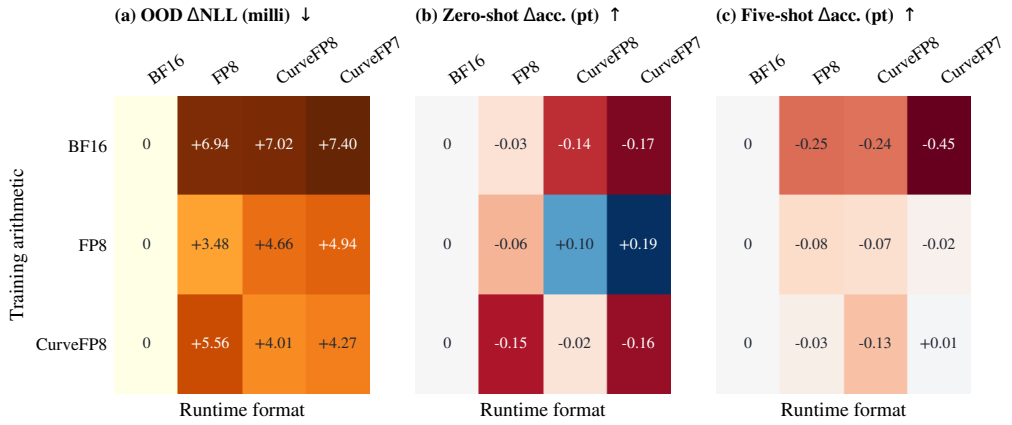


Figure 5: Complete three-seed format-transfer matrix. Each row fixes the training arithmetic and each column the runtime format; values are relative to BF16 runtime on the same checkpoint.

sequence length 1024, global batch size 96, and 30,517 updates. AdamW uses a 1,000-step warmup followed by cosine decay from 6×10^{-4} to 6×10^{-5}. Modes within a triplet share initialization, token order, optimizer, and schedule. Parameters and optimizer states remain FP32; nonlinear operations use BF16. Training disables K/V caching, and each lane applies its forward format to K/V projection linears.

B.3 OOD AND DOWNSTREAM EVALUATION

OOD language modeling uses the complete WikiText-103 and PG-19 test sets (Merity et al., 2016; Rae et al., 2019). The zero-shot suite contains HellaSwag, PIQA, ARC-Easy/Challenge, WinoGrande, LAMBADA, and BLiMP (Zellers et al., 2019; Bisk et al., 2020; Clark et al., 2018; Sakaguchi et al., 2019; Paperno et al., 2016; Warstadt et al., 2020); the five-shot suite repeats all but ARC-Challenge and BLiMP. Evaluation uses lm-evaluation-harness 0.4.11, the training tokenizer, 1,024-token context, and 10,000 bootstrap iterations. Figure 5 gives all training/runtime-format deltas; Table 5 in the main paper gives absolute BF16-runtime checkpoint quality.

CurveFP8 gives the strongest paired result on WikiText-103, lowering perplexity in all 12 seed/runtime-format comparisons. PG-19 favors CurveFP8 in 8/12 comparisons but has larger seed variation; zero- and five-shot means favor it in 7/12 and 6/12 comparisons and remain close across runtime formats. The largest conservative single-seed combined-error score is $|z| = 1.43$. These results support one consistent OOD advantage and broad transfer parity, not universal downstream superiority.

C SPATIAL-TILE HARDWARE MICROBENCHMARK

C.1 ARITHMETIC AND TILE CONTRACT

The hardware check compares CurveFP8 E4C3 with FP8 E4M3FN in matched 4×4 output-stationary tiles. Four A bytes and four B bytes are broadcast per cycle, forming 16 products that enter 16 independent signed 64-bit Q32 accumulators. Start/last controls delimit variable-K outer-product streams; completed outputs drain as four 256-bit rows. Both tiles therefore accept 16 MACs per cycle. SRAM, NoC, scale generation and storage, scheduling, larger tiling, result buffering, and BF16/FP16 boundary conversion are excluded from both designs.

CurveFP packs sign in bit 7 and magnitude rank m in bits 6:0, with $m = 0$ reserved for zero. For nonzero operands, $q = m - 64$ and the product coordinate is $q_p = m_a + m_b - 128$. E4C3 splits $q_p = 8e + k$; the implementation selects one Q16 coefficient $\mathrm{round}(2^{k/8} 2^{16})$, shifts it into Q32, applies the product sign, and accumulates. FP8 performs E4M3FN field decoding, subnormal handling, significand multiplication, sign application, and binary alignment into the same Q32 accumulator. Neither design rounds intermediate products or partial sums.

P0 places no register between product formation and accumulation; P1 adds one product/control register. The comparison independently selects the shallowest variant that closes the common 2.0 ns constraints rather than forcing equal latency. Table 7 records all four routed candidates.

Table 7: Complete Nangate45 pipeline sweep. WS and TNS are in ns; "–" denotes an unmeasured operating point.

Tile	Reg.	Area (μm^2)	Cells	Setup WS	Setup TNS	Hold WS	pJ/MAC
CurveFP8 P0	0	49,282.9	37,999	$+0.064$	0	$+0.104$	4.78
CurveFP8 P1	1	49,540.1	33,868	$+0.426$	0	$+0.104$	4.78
FP8 P0	0	53,245.5	41,077	-0.045	-1.223	$+0.105$	–
FP8 P1	1	51,646.8	35,716	$+0.302$	0	$+0.106$	4.75

C.2 Verification, tools, and power method

All 256 E4M3FN byte patterns were checked against PyTorch 2.6, and all 16,384 CurveFP E4 magnitude pairs were checked exhaustively. Selected RTL and final routed netlists pass 64 seeded variable-K matrices spanning 1,010 compute cycles and 1,024 checked outputs per datatype. Both selected routes have zero setup/hold TNS, final detailed-route DRC errors, antenna violations, and final flow errors.

The reproducible flow uses OSS CAD Suite 2026-08-11 (Yosys 0.68, Verilator 5.051, Icarus 14, and Slang 11), OpenROAD Flow Scripts revision `56496f3`, OpenROAD revision `ab6fd26`, and the public Nangate45 library.

Power uses a matched 4,096-cycle, $K = 64$ stream at 500 MHz with identical finite input bytes, matrix controls, and row-drain timing. OpenSTA reads each final ODB, SDC, extracted SPEF, and gate-level VCD; all routed pins are annotated. The functional cell simulation is zero-delay and omits timing-dependent glitches, and only one physical seed is measured. Consequently, the paper claims area and product-stage flexibility only; frequency scaling and application throughput require system-level study.