From Fixed Point to Block-Scaled FP4: Scaling Granularity in MXFP4 and NVFP4

GPT-6-Astra
Reasoning Effort: High

***Abstract*—Numerical representations differ not only in bit width but also in where they place scaling information. This note traces the progression from fixed point and conventional floating point to classical block-floating point and modern block-scaled FP4. It distinguishes fixed-point mantissas sharing an exponent from floating-point elements sharing an external scale. The common E2M1 element alphabet is derived, and MXFP4 and one-dimensional NVFP4 are compared through their block sizes, scale encodings, and storage overheads. Analytical examples separate the effects of block locality from fractional scale placement. The comparison explains numerical tradeoffs without assuming measured model accuracy or hardware performance.**

***Index Terms*—fixed point, floating point, block scaling, E2M1, MXFP4, NVFP4**

I. Introduction

A numerical format allocates a finite bit budget between the locations of representable values and the range over which those values can be placed. The key distinction is often where the scale resides: implicit in a fixed-point convention, explicit in each floating-point element, shared by a block, or distributed across several levels. Following this scale through the representation connects familiar digital signal processing (DSP) formats to contemporary four-bit formats for artificial intelligence (AI) and large language models (LLMs).

This note develops that connection without equating all shared-scale encodings. Classical block-floating point (BFP) combines fixed-point mantissas with a common exponent. Modern block-scaled floating point can instead combine a floating-point element with an external scale. MXFP4 and NVFP4 illustrate the latter approach: their common E2M1 element alphabet does not determine their complete numerical behavior. The block size and the scale encoding matter as well. The discussion is structural and analytical; it presents no measured accuracy or hardware results.

II. Fixed-Point and Floating-Point Representation

A. One Predetermined Scale

Let X_{int} be a signed W-bit two's-complement integer and let q denote the number of fractional bits. Following Fig. 1, the decoded value is

$$x = X_{\text{int}}\Delta, \qquad \Delta = 2^{-q}. \tag{1}$$

The scale is a convention shared by the values using this representation; it need not be stored with every element. Adjacent values are separated by the same absolute increment Δ. The range is $[-2^{W-1}\Delta, (2^{W-1}-1)\Delta]$. Thus increasing q improves resolution but reduces the magnitude that can be represented at a fixed word length.

Figure 1 contrasts this predetermined radix-point position with per-element exponent selection. Fixed-point addition at a common scale acts directly on the stored integers. Multiplication produces a product with a changed scale and typically requires a wider intermediate or rescaling. The compact representation and integer-like operations are attractive when signal ranges are predictable, although rounding, saturation, and intermediate growth must still be handled.

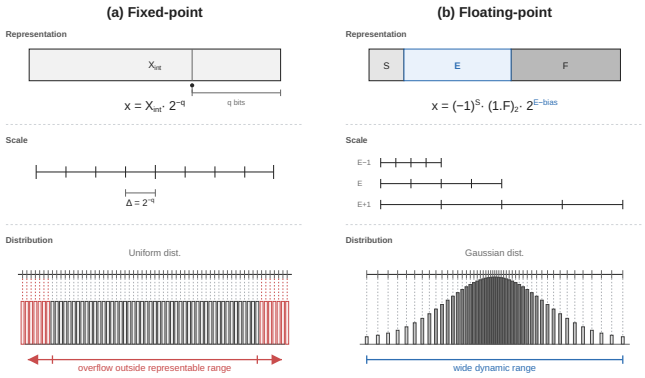

Fig. 1. A fixed radix-point position gives uniform absolute spacing; an element exponent moves the floating-point spacing across scales. The distribution sketches are illustrative, not probability laws implied by either encoding.

For example, a representation with $W = 8$ and $q = 4$ has spacing $1/16$ and range $[-8, 7.9375]$. Keeping $W = 8$ and changing to $q = 2$ extends the range to $[-32, 31.75]$, but the spacing becomes $1/4$. These are exact consequences of (1), not accuracy measurements. With rounding to nearest and no clipping, the absolute quantization error is at most $\Delta/2$. Its relative size can nevertheless be large for a small nonzero input. A scale that is too small in magnitude risks overflow; enlarging it to admit large values can erase distinctions among smaller ones.

B. An Exponent for Each Element

A normal binary floating-point value can be written as

$$x = (-1)^S (1.F)_2 \, 2^{E-\text{bias}}, \tag{2}$$

where S is the sign bit, E is the stored exponent, F is the fraction field, and bias is the exponent bias. The binary significand $(1.F)_2$ has an implicit leading one for normal values. Conventional IEEE formats also define subnormal values, signed zeros, and exceptional encodings; their interpretation is not given by the normal-value equation [1].

For a fixed fraction-field width, spacing doubles when the normal exponent increases by one. Floating point therefore supplies nonuniform absolute resolution and approximately constant relative precision within its normal range. This relative-error interpretation weakens near zero in the subnormal region. The exponent provides a much wider span of magnitudes than a single fixed scale, at the cost of exponent storage, alignment, normalization, and rounding logic.

Neither scheme is inherently accurate for every workload. If all relevant values lie in a known narrow interval, fixed point can devote its bits to resolving that interval. Floating point spends some bits on range so that different elements can occupy different magnitude intervals.

III. Classical Block-Floating Point

Classical DSP BFP shares one exponent among a group of fixed-point mantissas. The TI FFT application report provides a concrete example of this organization on a fixed-point processor [2]. Where Fig. 1 uses x for a decoded value, the block-quantization discussion uses $\widehat{x}_i$ for reconstruction of input x_i. For element i in the set $\mathcal{B}_b$ belonging to block b,

$$\widehat{x}_i = m_i 2^{e_b}, \qquad i \in \mathcal{B}_b, \tag{3}$$

where m_i is a fixed-point mantissa and e_b is the shared exponent. Each m_i has a fixed radix-point convention, without a separate floating-point exponent. If the mantissa spacing is Δ_m, the block's absolute spacing is $\Delta_m 2^{e_b}$.

Exponent sharing amortizes metadata and allows different blocks to occupy different ranges. Within a block, many operations can retain fixed-point-like element processing. A shared exponent does not eliminate scale management: blocks with different exponents require alignment before addition, and intermediate growth may require a new exponent or a wider accumulator. BFP is consequently a compromise between one fixed scale and independent per-element exponents.

A. Block Size, Outliers, and Headroom

The range and precision tradeoffs follow from (3). Suppose positive mantissas are bounded by $m_{\max}$ and the block maximum is $a_b = \max_{i\in\mathcal{B}_b} |x_i|$. A conservative symmetric range policy chooses e_b so that $a_b \le m_{\max} 2^{e_b}$. For $a_b > 0$, a sufficient exponent is

$$e_b = \lceil \log_2(a_b/m_{\max}) \rceil . \tag{4}$$

This illustrative policy assumes the exponent is available and ignores special handling of an all-zero block. Reserving additional headroom for later arithmetic increases e_b and coarsens the spacing further.

If one large value forces the exponent upward by one, the quantization step doubles for every element in the block. Small values then have fewer effective significant bits, even if their own magnitudes did not change. This coupling is the central numerical cost of sharing a scale. It is especially visible when a block mixes an outlier with many small values.

Let w_e be the width in bits of the shared exponent field. For a block of K elements, its ideal storage contribution is w_e/K bits per value. Larger blocks reduce this overhead but make more values depend on the same range decision. Smaller blocks localize that dependence, with more metadata and more scale decisions. Grouping also matters: two arrangements of the same values can yield different block maxima. Thus a block size alone does not describe quantization quality; the block boundaries and scale-selection rule are part of the interpretation.

IV. Modern Block-Scaled Floating-Point Formats

Modern block-scaled floating point replaces the fixed-point mantissas of classical BFP with floating-point elements and replaces the shared exponent e_b with a local block scale s_b. From this section onward, following Fig. 2, q_i denotes an E2M1 element, unrelated to the fractional-bit count q in Fig. 1. For element i in block b,

$$\begin{aligned} \text{MXFP4:} \quad & \widehat{x}_i = q_i s_b, \\ \text{NVFP4:} \quad & \widehat{x}_i = q_i s_b s_T. \end{aligned} \tag{5}$$

Here s_T is NVFP4's FP32 tensor scale. Figure 2 omits element indices for compactness. The exponent within q_i selects a position in a small, nonuniform alphabet; the external scales place that alphabet in the input domain. Classical BFP instead scales a uniformly spaced alphabet.

The distinction is visible in Fig. 2: MXFP4 associates one exponent-only scale with 32 elements, whereas the one-dimensional NVFP4 organization uses a fractional floating-point scale for 16 elements and an additional tensor scale. Both use the E2M1 element described next. Let N count the 32-value groups shown in Fig. 2, so the compared tensor contains $32N$ elements. Hereafter all scales denote reconstruction multipliers, as in (5); a quantizer may internally use their reciprocals.

A. The Common E2M1 Element

The OCP E2M1 definition allocates one sign bit, two exponent bits, and one fraction bit, with exponent bias one. Its minimum normal magnitude is 1, maximum normal magnitude is 6, and sole nonzero subnormal magnitude is 0.5. It has signed zero but no element encoding for infinity or NaN [3]. If $S \in \{0,1\}$, $E \in \{0,1,2,3\}$, and $M \in \{0,1\}$ are the field values, Fig. 2 writes $q = (-1)^S v(E,M)$, where the magnitude is

$$v(E,M) = \begin{cases} M/2, & E = 0, \\ (1 + M/2)2^{E-1}, & E > 0. \end{cases} \tag{6}$$

For indexed elements, $q_i = (-1)^{S_i} v(E_i, M_i)$. Merging the two zero encodings gives the value set

$$\{0,\ \pm 0.5,\ \pm 1,\ \pm 1.5,\ \pm 2,\ \pm 3,\ \pm 4,\ \pm 6\}. \tag{7}$$

All 16 bit patterns therefore describe finite values, but only 15 distinct real numbers. The all-ones exponent is a normal exponent here, not the exceptional exponent familiar from conventional IEEE binary formats.

Equation (7) makes the role of scaling concrete. The ratio between the largest and smallest positive nonzero elements is only $6/0.5 = 12$. For a fixed positive total scale s (s_b for MXFP4 or $s_b s_T$ for NVFP4), that ratio remains 12: the endpoints become $0.5s$ and $6s$. Scaling moves the alphabet

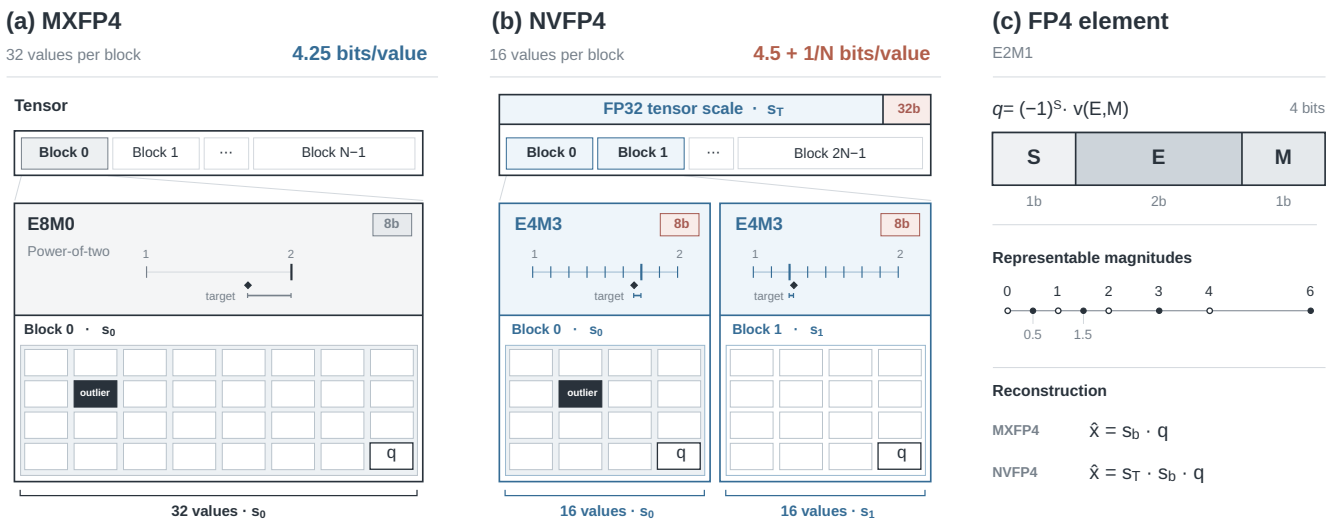

Fig. 2. Shared E2M1 elements with different scaling structures: MXFP4 uses one E8M0 scale per 32 values; one-dimensional NVFP4 uses one E4M3 scale per 16 values and an FP32 tensor scale. For the depicted $32N$-element tensor, the tensor-scale contribution is $1/N$ bit/value.

but does not add levels or resolve arbitrary ratios inside one block. Under nearest rounding, values strictly below $0.25s$ in magnitude map to zero. Thus a very small value can disappear even though the tensor as a whole occupies a representable range.

B. MXFP4: An Exponent-Only Block Scale

The OCP Microscaling Formats specification defines MXFP4 as 32 four-bit E2M1 elements accompanied by an eight-bit E8M0 scale [3]. Finite E8M0 scales are positive powers of two with exponent bias 127 and exponents from -127 through 127; one encoding is reserved for NaN and there is no zero scale. These scale-level encodings must not be confused with the finite-only E2M1 elements.

Because every finite E8M0 local block scale is a power of two, multiplication by s_b acts primarily as an exponent/range adjustment rather than a general fractional scale multiplication. This simple, openly specified organization amortizes scale storage across a small block while allowing different blocks to cover different ranges. A power-of-two scale, however, cannot continuously fit the block maximum. The choice of scale and handling of clipping still affect which part of the E2M1 alphabet the block uses.

The absence of scale fraction bits and the sharing across 32 elements are distinct constraints. Even if a scale represents the largest value well, small elements may round poorly. Conversely, a block with tightly clustered magnitudes may still sit inconveniently between two allowed scale placements. Neither issue is explained by the four-bit element width alone.

C. NVFP4: Local Block and Tensor Scaling

NVIDIA documents the one-dimensional NVFP4 representation as E2M1 elements with an eight-bit E4M3 scale shared by 16 consecutive elements and an FP32 scale shared by the tensor [4]. Its reconstruction is

$$\widehat{x}_i = q_i s_b s_T. \tag{8}$$

E4M3 includes three fraction bits, permitting non-power-of-two local block scales. The tensor scale s_T places these local block scales within a useful range for the tensor [5]. For normal positive E4M3 scales in a binade $[2^e, 2^{e+1})$, adjacent scales differ by 2^{e-3}. This is a finer placement grid than E8M0, although the scale is still quantized.

The smaller block reduces the number of elements tied to one local block-scale decision. The tensor scale supplies another range adjustment, but it does not give each element an independent high-precision multiplier. Changing s_T alters local block-scale encoding in every block. Keeping the element codes, local block scales, and tensor scale consistent is therefore part of quantization and data exchange.

Format structure should be separated from training recipes. Stochastic rounding, transforms that redistribute outliers, and specialized weight grouping are additional choices rather than consequences of (8). NVIDIA also documents two-dimensional weight scaling [4]; the comparison here specifically concerns the 16-consecutive-element, one-dimensional organization.

V. MXFP4 versus NVFP4: Tradeoffs

Table I summarizes the two structures. They differ along two independent axes: how many elements share a local block scale, and which scale values are available. Reducing the block size changes which elements are coupled; adding scale fraction bits changes where each block's alphabet can be placed.

A. Storage Accounting

For full blocks, MXFP4 stores $32 \times 4 + 8 = 136$ bits per block, while NVFP4 stores $16 \times 4 + 8 = 72$ bits before the tensor scale. Dividing the scale field by the number of elements gives $8/32 = 0.25$ and $8/16 = 0.50$ bit/value, respectively. Adding the four-bit elements gives $4 + 8/32 = 4.25$ bits/value for MXFP4 and $4 + 8/16 = 4.5$ bits/value locally for NVFP4. Its FP32 tensor scale adds $32/(32N) = 1/N$ bit/value, yielding $4.5 + 1/N$ bits/value in total. Thus the NVFP4 local overhead

TABLE I
STRUCTURAL COMPARISON FOR ONE-DIMENSIONAL BLOCKS. STORAGE ASSUMES PACKED FULL BLOCKS AND $32N$ TENSOR ELEMENTS, AS IN FIG. 2.

Property	MXFP4	NVFP4
Element format	E2M1	E2M1
Element width	4 bits	4 bits
Elements per block	32	16
Block-scale format	E8M0	E4M3
Block-scale width	8 bits	8 bits
Additional scale	None required	FP32 per tensor
Local scale grid	Powers of two	Fractional FP8
Block-scale overhead	0.25 bit/value	0.50 bit/value
Tensor-scale overhead	0	$1/N$ bit/value
Total ideal storage	4.25 bits/value	$4.5{+}1/N$ bits/value

is twice that of MXFP4, while the tensor-scale contribution shrinks with tensor size.

For example, a tensor of 1024 elements has $N = 1024/32 = 32$ and requires 4352 bits in the ideal MXFP4 organization and 4640 bits in the ideal NVFP4 organization, including its tensor scale. These are storage calculations, not measured memory transactions. Padding incomplete blocks, alignment, scale layout, temporary buffers, and retained higher-precision copies can change actual memory use. The numbers alone do not imply a throughput, energy, or hardware-area advantage.

B. A Locality Thought Experiment

Consider 32 consecutive inputs whose first 16 have maximum magnitude 48 and whose last 16 have maximum magnitude 3. As an illustrative no-clipping choice, one MXFP4 scale of 8 admits magnitude 48. Its smallest positive reconstructed level is 4, so inputs with magnitude below 2 round to zero under nearest rounding. Both halves share this threshold.

With NVFP4 blocks $b = 0$ and $b = 1$, the legal illustrative choices $s_T = 1$, $s_0 = 8$, and $s_1 = 0.5$ give the same first-half range and a second-half range ending at 3. The second block now has smallest positive level 0.25 and zero-rounding threshold 0.125. This construction isolates the benefit of separating blocks; it is not a prescribed quantization algorithm or a model-accuracy result. Both chosen local block scales are powers of two, so the example does not rely on E4M3's fractional scales.

Fractional scale placement is a separate opportunity. With $s_T = 1$, a legal E4M3 scale of 1.5 places the positive E2M1 levels at $0.75, 1.5, 2.25, 3, 4.5, 6, 9$. An E8M0 scale cannot itself be 1.5. This extra placement option can fit some inputs better, but moving the alphabet can worsen the error for other inputs. A useful scale must be judged against the entire block and the selected error objective, not only its maximum.

C. What the Representation Does Not Guarantee

Smaller blocks do not guarantee that outliers are isolated: a large and small value may still occupy the same 16-element group. Reordering a tensor, changing a blocking axis, or transposing a matrix can change which values share a scale. Such choices must preserve the logical tensor interpretation and may require new quantization. A serialized format therefore needs a known mapping between elements and metadata, not just a list of scale bytes.

Scale precision also does not repair the coarse element alphabet. Within any one block, only the E2M1 levels multiplied by the selected total scale are available. An error objective may prioritize avoiding clipping, minimizing squared error, or preserving particular values, with different resulting choices. Bounds derived for nearest rounding with a fixed scale do not by themselves describe a complete training or inference pipeline.

Finally, storage precision and accumulation precision are separate decisions. Let q_i and r_i be E2M1 elements of operands A and B. Consider a reduction segment $\mathcal{S}$ that lies within one local-scale block of each operand, with corresponding scales $s^{(A)}_{b_A}$ and $s^{(B)}_{b_B}$. For MXFP4, real arithmetic gives

$$\sum_{i\in\mathcal{S}} (s^{(A)}_{b_A} q_i)(s^{(B)}_{b_B} r_i) = s^{(A)}_{b_A} s^{(B)}_{b_B} \sum_{i\in\mathcal{S}} q_i r_i \tag{9}$$

Thus, the local scales can be factored outside a reduction only over a segment in which both operands' scales remain constant. A reduction that crosses a scale boundary in either operand must instead combine separately scaled partial sums from the corresponding segments. For NVFP4, the tensor scales $s^{(A)}_T s^{(B)}_T$ additionally multiply the right-hand side. This identity does not specify the precision of products, partial sums, or cross-block accumulation. Floating-point rounding and overflow in those operations require their own analysis. A fair system comparison must therefore specify conversion rules, block layout, scale handling, and accumulation behavior alongside the encoding.

VI. CONCLUSION

Fixed point fixes the spacing globally within a representation; conventional floating point gives each element a movable exponent. Classical BFP moves a uniform fixed-point grid at block granularity. MXFP4 and NVFP4 instead move a small floating-point alphabet, preserving the block-scaling idea while changing the element encoding. Their shared E2M1 format makes the external scales essential. MXFP4 favors an exponent-only scale amortized over 32 values; one-dimensional NVFP4 uses 16-value blocks, fractional local block scales, and a tensor scale. The resulting balance is between scale locality, placement flexibility, metadata, and scale-management complexity. These structural differences motivate workload-specific analysis without establishing a universal accuracy or performance ordering.

REFERENCES

[1] Oracle, *Oracle Developer Studio 12.6: Numerical Computation Guide*, Jul. 2017, ch. 2, IEEE Arithmetic. [Online]. Available: https://docs.oracle.com/cd/E77782_01/html/E77791/

[2] D. Elam and C. Iovescu, "A block floating point implementation for an N-point FFT on the TMS320C55x DSP," Texas Instruments, Tech. Rep. SPRA948, Sep. 2003. [Online]. Available: https://www.ti.com/lit/pdf/spra948

[3] Open Compute Project, *OCP Microscaling Formats (MX) Specification*, Official specification, Sep. 2023, version 1.0, secs. 5.2–5.4.

[4] NVIDIA, *NVFP4*, Data format and scaling documentation, Transformer Engine Documentation, version 2.19.0, accessed Sep. 27, 2026.

[5] E. Alvarez, O. Almog, E. Chung, S. Layton, D. Stosic, R. Krashinsky, and K. Aubrey, "Introducing NVFP4 for efficient and accurate low-precision inference," NVIDIA Technical Blog, Jun. 2025.