Efficient Floating Point Representations for Accumulators in Matrix Multiplication

Martin Wiesner, Joren Dumoulin, Marian Verhelst
MICAS-ESAT, KU Leuven, Belgium

***Abstract*—Quantization is widely used to reduce the memory footprint, model size, and computational demands of neural networks during inference. While INT8 quantization is commonly explored in research and supported by hardware, FP8 has gained attention for offering a better trade-off between dynamic range and precision. However, FP8 formats lack the precision required for accurate accumulation during matrix multiplications. This work investigates the use of custom accumulator formats alongside three FP8 variants (E5M2, E4M3, and E3M4) to improve inference performance. We evaluate the numerical accuracy of matrix multiplications under both Gaussian and heavy-tailed Student's t-distributions. The analysis focuses on the influence of exponent and mantissa precision on Mean Squared Error (MSE), using a 64-bit floating-point result as reference. Results indicate that significantly less than the commonly used 32 bits are required for the accumulator. The type of distribution does not significantly impact the numerical accuracy. These findings demonstrate that with appropriate accumulator design, FP8 quantization can be a competitive alternative to INT8 in specific inference scenarios.**

***Index Terms*—Matrix Multiplication, Floating Point, Neural Networks**

I. Introduction

To reduce model size, memory usage and computational cost during inference, quantization of neural network weights is sometimes applied. This process reduces numerical precision, usually from 32-bit floating point representation (FP32) to either FP8 or INT8. According to [1], FP8 outperforms INT8 as representation of weights during inference. However, FP8 lacks sufficient precision to handle the accumulation of the sum of all multiplications. While FP32 is a commonly used accumulation format, the cost of an FP32 addition outweighs the cost of an FP8 multiplication significantly, possibly eliminating the benefits of using an FP8 format. Therefore, an alternative floating-point format is needed for accumulation.

II. Related Works

In their study [2] comparing FP8 and INT8 for efficient deep learning inference, Qualcomm AI Research conclude that INT8 generally outperforms FP8 in terms of efficiency and accuracy during inference. Their work highlights the practical advantages of INT8 quantization, especially in well-behaved layers, and points to the added hardware complexity of FP8 accumulation. However, their evaluation primarily focuses on standard accumulator configurations (FP32) and does not explore the impact of alternative accumulator precisions on FP8 performance. Additionally, INT8 quantization often requires scaling, which introduces additional overhead and is sensitive to outliers in activation distributions. Thus, FP8 could outperform INT8 in various inference scenarios.

Another important study on precision and accumulation comes from DeepSeek in their technical report on DeepSeek-V3 [3]. They report that NVIDIA Hopper Tensor Cores truncate FP8×FP8 multiplications to retain only the top 14 bits of the mantissa during accumulation. While this design is efficient, DeepSeek finds that it introduces unacceptable errors during training, where precise gradient accumulation is essential. However, their study is limited to training workloads and does not explore inference scenarios, the focus of this paper.

III. Methodology

To evaluate the effect of accumulator precision on different floating-point formats, a Java-based simulator is developed allowing for bit-precise simulation of different hardware implementations. The program generates two random matrices of variable size N (16, 64, 256 or 1024), represented in one of the three main FP8 formats: E5M2, E4M3, or E3M4, where E5M2, for example, denotes a floating point format with 5 exponent bits and 2 mantissa bits. One matrix represents the output of the previous neural network layer and the other the weights of the current layer. Both are initialized with a standard deviation s which equals the power of 2 closest to the maximum value representable by the FP8 format divided by 7. The standard deviation of the matrix resulting from the multiplication of these two matrices will then equal $s^2 \cdot \sqrt{N}$ The values in this matrix can now be divided by $s \cdot \sqrt{N}$ to have the same standard deviation s as the original matrices. By choosing s to be a power of 2 and N to be the square of a power of 2, the divisor $s \cdot \sqrt{N}$ is guaranteed to be a power of 2, making this division easily implementable.

To evaluate performance under different data distributions, matrices are generated with a mean of 0 using both a Gaussian distribution and a Student's t-distribution with $\nu = 3$, which introduces more outliers and models heavy-tailed behaviour more realistically.

To assess the numerical error introduced by limited-precision accumulators, the Mean Squared Error (MSE) is computed by comparing the output matrix to a reference result obtained using a 64-bit accumulator. As a benchmark for acceptable error, the quantization error, which is caused by

rounding from FP32 to FP8 is used. This provides a reference point for understanding whether accumulator-induced errors are within reasonable bounds relative to quantization noise. Thus, the FP representation with the least number of bits and of which the MSE is smaller than the quantization error is determined to be optimal. Relative Mean Square Error will be used to refer to the MSE divided by the quantization error.

Lastly, since the weight matrix is determined once and then used many times during inference, its elements can be strategically ordered. Simulations were run where the weight matrix was ordered in different ways before the matrix multiplication. This process is investigated as a simple method to mitigate numerical accumulation errors inherent in floating-point arithmetic.

To run the simulations servers provided by ESAT were used.

IV. Results and Discussion

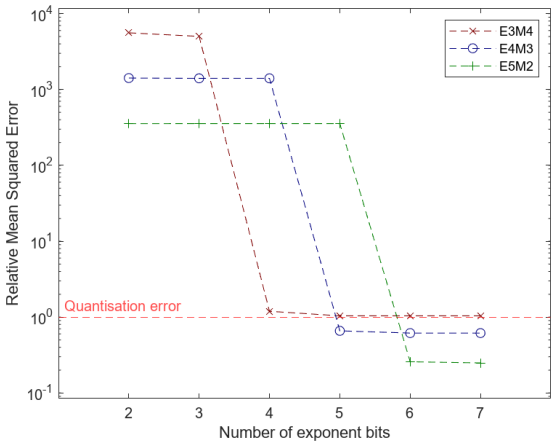

Fig. 1. Relative Mean Squared Error (MSE / Quantization Error) as a function of accumulator exponent bit width on a logarithmic chart for a 64x64 matrix when the accumulator mantissa is fixed at 9 bits.

1) Exponent bits: The program logged when overflow happened during multiplication. When this occurred the MSE was extremely high and as soon as the exponent bit width was increased enough to prevent overflow the MSE dropped significantly. After this point adding more exponent bits had a negligible effect. In Figure 1 the relative mean square error demonstrates this behaviour for a 64x64 matrix using a Gaussian distribution with 9 bits for the accumulator mantissa. This effect was independent of distribution, original FP representation or matrix size. However, the number of exponent bits required to avoid overflow was dependent on the number of exponent bits in the original distribution.

2) Mantissa bits: Increasing the mantissa bit width decreases the MSE exponentially, as can be seen in Figure 2. The number of required mantissa bits depends on the FP8 representation and matrix size but not on the distribution (Figure 3).

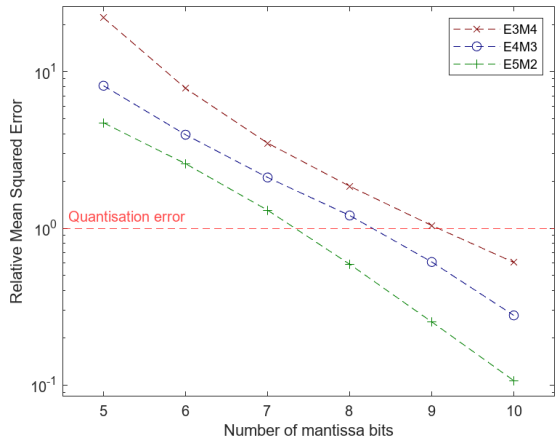

Fig. 2. Relative Mean Squared Error (MSE / Quantization Error) as a function of accumulator mantissa bit width on a logarithmic chart for a 64x64 matrix when the accumulator exponent is fixed at 6 bits.

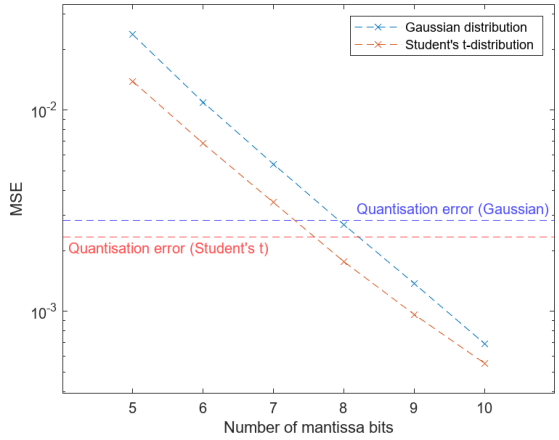

Fig. 3. Comparison of two distributions of the weights of a 16x16 matrix using E3M4 with an accumulator with 4 exponent bits and varying number of mantissa bits

3) Optimal FP representation: The quantization errors of a Gaussian distribution and a Student t-distributions are not exactly equal, but the optimal FP representation for their accumulator is (Figure 3). The number of required exponent and mantissa bits are always the starting FP8 representation's plus a certain number dependent on the size of the matrix. The optimal FP representation can be determined using Table I, e.g. for E4M3 the optimal FP representation of the accumulator in a 64x64 matrix is E5M9.

TABLE I
Optimal FP representation

Size of matrix	Exponent	Mantissa
16x16	+1	+4
64x64	+1	+6
256x256	+2	+7
1024x1024	+2	+9

4) Matrix Size: Figure 4 illustrates the correlation between matrix size and the required mantissa bit width. As matrix

dimensions increase, so does the minimum mantissa precision needed to maintain accuracy. Specifically, for the worst-case FP8 representation at a matrix size of 1024, 13 mantissa bits are required.

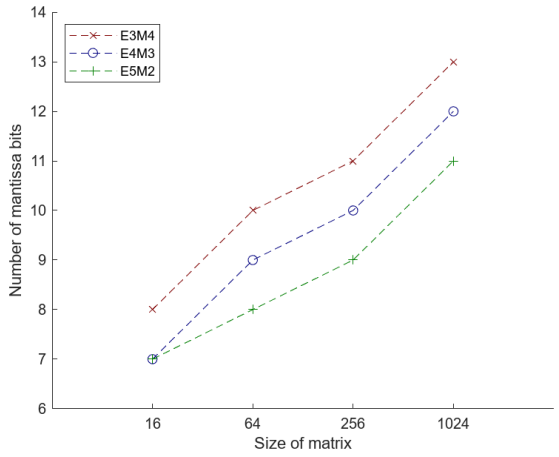

Fig. 4. Required mantissa bit width required to reach quantization error for increasing matrix size

5) Ordering weights matrix: As expected, ordering the weights in ascending order of absolute values significantly reduces the MSE (Table II). By adding smaller values to the accumulator first, the intermediate sums maintain a smaller exponent. This process helps to preserve the precision of the subsequent small values being added, preventing them from being completely rounded away when added to a much larger accumulated sum. This strategic ordering minimizes the accumulation of rounding errors, thereby improving overall numerical accuracy.

TABLE II
EFFECT OF ORDERING WEIGHTS ON MSE

	E3M4	E4M3	E5M2
Random	1.7688E-6	4.9726E-6	2.5285E-5
Ascending Absolute Values	2.5354E-7	2.1432E-6	8.5580-6
Improvement	18.5x	3.5x	4.8x

V. CONCLUSION

The work successfully demonstrated that significant bit width savings can be achieved whilst maintaining sufficient numerical accuracy. By comparing results against a 64-bit reference, the simulations confirmed that an accumulator requires substantially fewer than 32 bits to effectively manage the accumulation of FP8 multiplications.

Firstly, the required exponent bit width is primarily dictated by the need to prevent overflow, with minimal additional benefit beyond this threshold. Secondly, the necessary mantissa bit width scales predictably with the size of the matrix, demanding up to 13 bits for the largest 1024×1024 matrices. This finding concurs with DeepSeek's report [3] that NVIDIA Hopper Tensor Cores utilize 14 mantissa bits for FP8 accumulation. Importantly, the numerical performance proved largely independent of the input data's distribution (Gaussian vs. Student's t), simplifying hardware design considerations. Finally, strategic ordering of the weight matrix in ascending absolute value provided a substantial reduction in Mean Squared Error, highlighting a simple algorithmic approach to mitigating accumulation error.

These results support the viability of FP8 quantization as an alternative to INT8 for specific deep learning inference hardware implementations.

REFERENCES

[1] H. Shen, N. Mellempudi, X. He, Q. Gao, C. Wang, and M. Wang, "Efficient post-training quantization with fp8 formats," *arXiv preprint arXiv:2309.14592*, 2023. [Online]. Available: https://arxiv.org/abs/2309.14592

[2] R. Banner, N. Mellempudi, A. Kumar, J. Yan, B. J.-H. Tseng *et al.*, "Fp8 versus int8 for efficient deep learning inference," *arXiv preprint arXiv:2303.17951*, 2023. [Online]. Available: https://arxiv.org/abs/2303.17951

[3] D.-V. Team, "Deepseek-v3: Towards deeper language modeling with vision and external memory," https://github.com/deepseek-ai/DeepSeek-V3, 2024, accessed: 2025-05-19.