

# ADDITION IS ALL YOU NEED FOR ENERGY-EFFICIENT LANGUAGE MODELS

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

Large neural networks spend most computation on floating point tensor multiplications. In this work, we find that a floating point multiplier can be approximated by one integer adder with high precision. We propose the linear-complexity multiplication ( $\mathcal{L}$ -Mul) algorithm that approximates floating point number multiplication with integer addition operations. The new algorithm costs significantly less computation resource than 8-bit floating point multiplication but achieves higher precision. Compared to 8-bit floating point multiplications, the proposed method achieves higher precision but consumes significantly less bit-level computation. Since multiplying floating point numbers requires substantially higher energy compared to integer addition operations, applying the  $\mathcal{L}$ -Mul operation in tensor processing hardware can potentially reduce **95%** energy cost by element-wise floating point tensor multiplications and **80%** energy cost of dot products. We calculated the theoretical error expectation of  $\mathcal{L}$ -Mul, and evaluated the algorithm on a wide range of textual, visual, and symbolic tasks, including natural language understanding, structural reasoning, mathematics, and commonsense question answering. Our numerical analysis experiments agree with the theoretical error estimation, which indicates that  $\mathcal{L}$ -Mul with 4-bit mantissa achieves comparable precision as `float8_e4m3` multiplications, and  $\mathcal{L}$ -Mul with 3-bit mantissa outperforms `float8_e5m2`. Evaluation results on popular benchmarks show that directly applying  $\mathcal{L}$ -Mul to the attention mechanism is almost lossless. We further show that replacing all floating point multiplications with 3-bit mantissa  $\mathcal{L}$ -Mul in a transformer model achieves equivalent precision as using `float8_e4m3` as accumulation precision in both fine-tuning and inference.

## 1 INTRODUCTION

Modern artificial intelligence (AI) systems are significant energy consumers. Because of the large scale computation needed for neural network inference, AI applications based on such models are consuming a considerable amount of electricity resource. Reportly, the average electricity consumption of ChatGPT service in early 2023 was 564 MWh per day, equivalent to the total daily electricity usage of 18,000 families in the United States<sup>1</sup>. It is estimated that Google’s AI service could consume as much electricity as Ireland (29.3 TWh per year) in the worst-case scenario (de Vries, 2023).

Reducing the amount of computation needed by neural networks is the key to reduce both energy consumption and inference speed for large-scale AI models. Neural networks, especially large language models (LLMs) (Radford et al., 2019; Brown, 2020; Achiam et al., 2023; Touvron et al., 2023; Team et al., 2023), contain a large number of floating point parameters involved in element-wise and matrix multiplication computations. In transformer (Vaswani, 2017) based LLMs, the attention mechanism is a major bottleneck that limits the computation efficiency. Given a input context of  $N$  tokens, the complexity of standard attention mechanism computation is  $O(N^2)$ , involving multiplying high dimensional tensors. Besides attention, there are also a large amount of element-wise multiplication and linear transformation computations. In this work, we propose the linear-complexity multiplication ( $\mathcal{L}$ -Mul) algorithm, which approximates floating point multiplication with integer addition operations. The algorithm can be integrated into existing models at various

<sup>1</sup><https://www.eia.gov/tools/faqs/faq.php?id=97>

054 levels, such as replacing the multiplication in the attention mechanism or substituting all matrix and  
 055 element-wise multiplications.

056  
 057 The proposed  $\mathcal{L}$ -Mul method will lead to a significantly reduced energy consumption for both model  
 058 training and inference. In modern computing hardware, multiplications between floating point num-  
 059 bers consumes significantly higher energy than addition operations (Horowitz, 2014). Specifically,  
 060 multiplying two 32-bit floating point numbers (`fp32`) costs four times energy as adding two `fp32`  
 061 numbers, and 37 times higher cost than adding two 32-bit integers (`int32`). The rough energy  
 062 costs for various operations are shown in Table 1. In PyTorch (Paszke et al., 2019), the default  
 063 precision for accumulating tensor multiplication results is set to `fp32`. While I/O and control op-  
 064 erations are not considered, approximating `fp32` multiplications with `int32` additions consumes  
 065 only  $1/37 \approx 2.7\%$  of the energy. When the accumulation precision is reduced to `fp16`, integer  
 066 addition consumes approximately 4.7% of the energy required for floating-point multiplication.

Operation	Integer		Floating Point	
	8-bit	32-bit	16-bit	32-bit
Addition	0.03 pJ	<b>0.1 pJ</b>	0.4 pJ	0.9 pJ
Multiplication	0.2 pJ	3.1 pJ	1.1 pJ	<b>3.7 pJ</b>

072 Table 1: Energy cost of various arithmetic operations cited from Horowitz (2014).

073  
 074  
 075 We evaluate the numerical precision of  $\mathcal{L}$ -Mul algorithm on transformer-based language models with  
 076 a wide range of language and vision tasks. Experiments with full-precision model weights show that  
 077 replacing standard multiplication operations with  $\mathcal{L}$ -Mul in the attention mechanism is almost loss-  
 078 less for transformer-based LLMs. On natural language reasoning tasks, the average performance  
 079 loss of  $\mathcal{L}$ -Mul-based attention is 0.07% across commonsense, structured reasoning, language under-  
 080 standing. On vision tasks,  $\mathcal{L}$ -Mul-based attention gained 0.12% accuracy improvement on visual  
 081 question answering, object hallucination, and free-form visual instruction tasks. The experiment re-  
 082 sults are obtained by directly adapting pretrained LLMs with the standard attention implementation  
 083 to the new  $\mathcal{L}$ -Mul-based attention mechanism without any additional training.

084 The error estimation and ablation study show that under the training-free setting,  $\mathcal{L}$ -Mul with 4-  
 085 bit mantissa can achieve comparable precision as multiplying `float8_e4m3` numbers, and  $\mathcal{L}$ -  
 086 Mul with 3-bit mantissa outperforms `float8_e5m2` multiplication. We also show that fine-tuning  
 087 can fix the performance gap between  $\mathcal{L}$ -Mul and the standard multiplication. Fine-tuning a model  
 088 where all multiplication operations in attention mechanisms, linear transformations, and element-  
 089 wise products are replaced by 3-bit-mantissa  $\mathcal{L}$ -Mul results in comparable performance to fine-  
 090 tuning a standard model with an accumulation precision of `float8_e4m3`.

091 In the expansive landscape of AI efficiency research, our approach centers on enhancing the effi-  
 092 ciency of tensor arithmetic algorithms—a direction that is orthogonal yet complementary to prevail-  
 093 ing efforts in I/O and control optimization (Jouppi et al., 2017; Choquette et al., 2021; Abts et al.,  
 094 2022). We believe that truly energy- and compute-efficient AI computation will emerge from a  
 095 holistic integration of optimizations across I/O, control, and arithmetic operations.

## 097 2 METHOD

### 100 2.1 BACKGROUND: FLOATING-POINT NUMBERS AND TENSORS

101 Most machine learning models, including neural networks, use floating point (FP) tensors to repre-  
 102 sent their inputs, outputs, and trainable parameters. Typical choices are 32-bit and 16-bit FP tensors  
 103 (`fp32` and `fp16`) defined by the IEEE 754 standard shown in Figure 1.

104 Multiplication operations are generally more complicated than additions, and FP operation are more  
 105 costly than integers (Horowitz, 2014). Table 1 shows that multiplying two `fp32` numbers consumes  
 106 37 times higher energy than adding two 32-bit integers. While the complexity of integer addition  
 107 is  $O(n)$  where  $n$  is the number of bits used for representing the number, FP multiplication requires

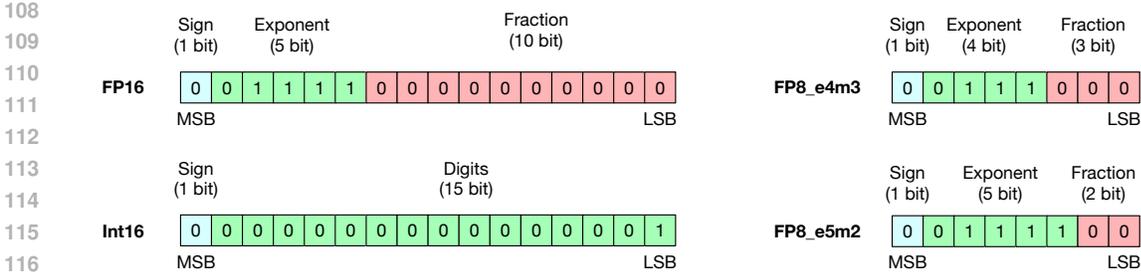


Figure 1: 16-bit, 8-bit floating point numbers defined in IEEE 754 and on various hardware for tensor computations, and the 16-bit integer. MSB stands for most significant bit and LSB stands for least significant bit.

$O(e)$  exponent addition,  $O(m^2)$  mantissa multiplication, and rounding. Here  $e$  and  $m$  stand for the number of bits used for exponent and mantissa parts of the FP numbers.

Modern LLM training and inference involves a large number of FP calculations in tensor computation. Consider calculating the element-size and dot products of two 2-D tensors:

$$Y_1 = A \circ X, Y_2 = A \cdot X^T; A, X \in R^{(N,k)}$$

Calculating  $Y_1$  involves  $N^2$  FP multiplications (Mul). If  $A$  and  $X$  are both fp32 tensors,  $A \circ X$  consumes 37 times higher energy than adding two int32 matrices of the same size. Similarly, Calculating  $Y_2$  involves  $(m \times n \times k)$  FP Mul and the same number of FP additions (Add). When  $A$  and  $X$  are fp32 tensors, each Mul-Add operation for two numbers consumes  $0.9 + 3.7 = 4.6$  (pJ) energy. If we replace the fp32 Mul with int32 Add, the energy cost becomes  $0.1 + 0.9 = 1.0$  (pJ), only 21.7% of the original cost. Similarly, if the inference is conducted in fp16, replacing fp16 Mul with int16 Add result in a  $1 - (0.05 + 0.4)/(1.1 + 0.4) = 70\%$  energy saving.

## 2.2 LINEAR-COMPLEXITY MULTIPLICATION ( $\mathcal{L}$ -MUL)

We propose  $\mathcal{L}$ -Mul, a FP multiplication algorithm with  $O(n)$  complexity, where  $n$  is the bit size of its FP operands. Consider two FP numbers  $x, y$ , whose exponents and fractions are  $x_e, y_e$  and  $x_m, y_m$  respectively, the vanilla FP Mul result is

$$Mul(x, y) = (1 + x_m) \cdot 2^{x_e} \cdot (1 + y_m) \cdot 2^{y_e} = (1 + x_m + y_m + x_m \cdot y_m) \cdot 2^{x_e+y_e}$$

plus an XOR operation ( $\oplus$ ) to decide the sign of the result. Assume  $x_m$  and  $y_m$  are mantissas of  $m$  bits. The  $O(m^2)$  mantissa multiplication operation is the complexity bottleneck of this calculation. We remove this operation and introduce a new multiplication algorithm that processes mantissas with a computational complexity of  $O(m)$ :

$$\mathcal{L}\text{-Mul}(x, y) = (1 + x_m + y_m + 2^{-l(m)}) \cdot 2^{x_e+y_e}, \quad l(m) = \begin{cases} m & \text{if } m \leq 3, \\ 3 & \text{if } m = 4, \\ 4 & \text{if } m > 4. \end{cases} \quad (1)$$

The offset exponent  $l(m)$  is defined according to the observation shown in Figure 3. In the following sections, we show that (1) the  $\mathcal{L}$ -Mul operation can be implemented by integer Adders, and (2) the algorithm is more accurate and efficient than fp8 multiplications.

The implementation of the algorithm is shown in Figure 2, where we also added the Inline PTX Assembly code we used to simulate the process on Nvidia GPUs. While Equation (1) contains 4 addition operations, the bit format design of FP numbers helps us implement the  $\mathcal{L}$ -Mul algorithm with one adder. Since the FP format handles  $1 + x_m$  implicitly, we do not have to compute the value of  $(1 + \dots)$ . The integer addition operation also automatically send the mantissa carry to the exponent. If the mantissa sum is greater than 2, a carry is automatically added to the exponent. This is different from the rounding process in traditional FP multiplier, where the fraction is manually rounded to  $1.x$  and the carry is added to the exponent as an independent step. As a result, the  $\mathcal{L}$ -Mul algorithm is more efficient than traditional FP multiplication by skipping both mantissa multiplication and rounding operations.

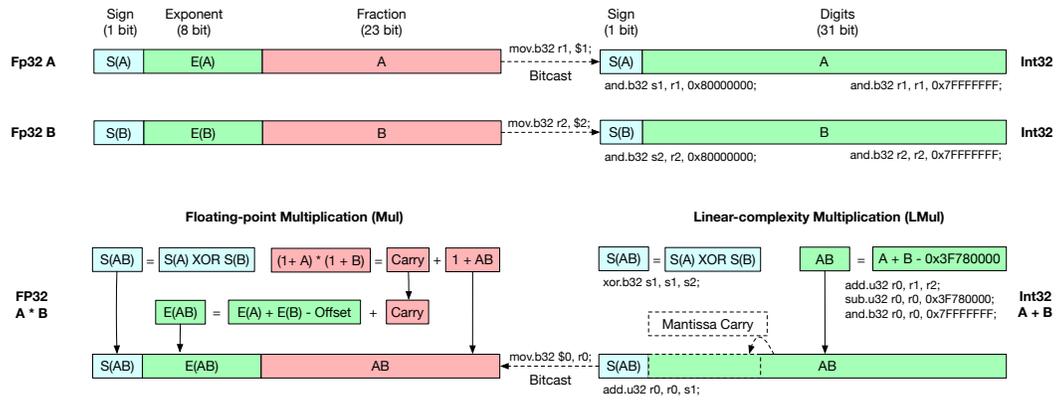


Figure 2: Comparing the process of regular floating-point multiplication and linear-complexity multiplication ( $\mathcal{L}$ -Mul) between two fp32 numbers. In the inline PTX Assembly code, \$1 and \$2 are fp32 registers storing inputs while \$0 is the fp32 register for output. s1, s2, r0, r1, r2 are unsigned int32 registers storing intermediate results. Note that the assembly program is only for numerical simulation on Nvidia GPUs. The optimal implementation is at the hardware level.

The construction of  $\mathcal{L}$ -Mul results can be expressed using the following equation, where all bit-level calculations are performed as operations between unsigned integers.

$$\begin{aligned} \mathcal{L}\text{-mul}(x, y)[0] &= x[0] \oplus y[0] \\ \mathcal{L}\text{-mul}(x, y)[1:] &= x[1:] + y[1:] - \text{offset} \end{aligned} \quad (2)$$

We further implement the attention mechanism with  $\mathcal{L}$ -Mul. In transformer models, the attention mechanism has a high computation cost because of its  $O(|C|^2)$  complexity to process the input context  $C$ . We found that  $\mathcal{L}$ -Mul can replace the complicated tensor multiplications with minimal performance loss needing no additional training. In this work we implement a more efficient attention mechanism as follows,

$$\begin{aligned} K &= H \cdot W_k, Q = H \cdot W_q, V = H \cdot W_v \\ A &= \text{softmax} \left[ \frac{\mathcal{L}\text{-matmul}(Q, K^T)}{\sqrt{d}} \right], H' = \mathcal{L}\text{-matmul}(A, V) \end{aligned} \quad (3)$$

where  $\mathcal{L}\text{-matmul}(Q, K^T)$  stands for a matrix multiplication operation where all regular FP multiplications are implemented in  $\mathcal{L}$ -Mul. By doing this, all FP multiplications are replaced with integer additions, which consumes significantly lower computation resource.

### 2.3 PRECISION AND COST ANALYSIS

In this section, we show that  $\mathcal{L}$ -Mul is more precise than fp8\_e4m3 multiplications but uses less computation resource than fp\_e5m2. To be concise, we do not consider the rounding to nearest even mode in both error analysis and complexity estimation for both Mul and  $\mathcal{L}$ -Mul.

#### 2.3.1 PRECISION ESTIMATION

The goal of the precision analysis is to find the precision of the  $\mathcal{L}$ -Mul algorithm is equivalent to rounding the fraction of a FP number to how many bits, e.g., fp8 with 2- or 3-bit mantissas (e5m2 or e4m3). Consider positive FP numbers  $x = (1 + x_m) \cdot 2^{x_e}$  and  $y = (1 + y_m) \cdot 2^{y_e}$ , they can be written in the following format if we explicitly highlight the  $k$  bits to be kept after rounding:

$$\begin{aligned} x &= (1 + x_k + x_r) \cdot 2^{x_e}, \quad x' = (1 + x_k) \cdot 2^{x_e} \\ y &= (1 + y_k + y_r) \cdot 2^{y_e}, \quad y' = (1 + y_k) \cdot 2^{y_e} \end{aligned}$$

where  $x_k, y_k$  are the first  $k$  bits of  $x_m, y_m$ , and  $x_r, y_r$  are the value of remaining bits that will be ignored after the  $k$ -bit rounding.  $x', y'$  are the rounded value of  $x, y$  by keeping the first  $k$  bits of

the mantissa. Consider  $x$  and  $y$  has  $m$ -bit mantissa in their full precision. For example, `Float16` numbers have 10-bit mantissa and `BFloat16` contain 7 bits. The error of  $Mul(x, y) = x \cdot y$  and its expectation are

$$e_{mul}^k = Mul(x, y) - Mul(x', y') = (x_k y_r + y_k x_r + x_r + y_r + x_r y_r) \cdot 2^{x_e + y_e} \quad (4)$$

$$E[e_{mul}^k] = f_1(m, k) \cdot E[2^{x_e + y_e}]$$

Comparing with a  $k$ -bit mantissa FP multiplication, the error of  $k$ -bit mantissa  $\mathcal{L}$ -Mul is

$$e_{lmul}^k = e_{mul}^k + (x_k y_k - 2^{-l(k)}) \cdot 2^{x_e + y_e} \quad (5)$$

$$E[e_{lmul}^k] = E[e_{mul}^k] + E[x_k y_k - 2^{-l(k)}] \cdot E[2^{x_e + y_e}]$$

With the equations above, we can compute the expectation of the precision gap between  $k$ -bit  $\mathcal{L}$ -Mul and FP multiplication:

$$E[e_{lmul}^k] - E[e_{mul}^k] = f_2(k) \cdot E[2^{e_x + e_y}], \quad E[e_{lmul}^k] = [f_1(m, k) + f_2(k)] \cdot E[2^{e_x + e_y}]$$

When  $x_m, y_m$  are evenly distributed, we can calculate the following expectations,

$$E[x_k] = \frac{1}{2}(1 - 2^{-k}), \quad E[x_r] = \frac{1}{2}(2^{-k} - 2^{-m})$$

By estimating  $f_1(m, k)$  and  $f_2(k)$  and further inferring  $E[e_{lmul}^k]$  and  $E[e_{mul}^k]$ , we find that  $\mathcal{L}$ -Mul is more accurate than `fp8_e5m2` with evenly distributed operands. However, the weight distribution is often biased in pretrained LLMs. Based on the combined weight distribution of five popular LLMs, we find that  $\mathcal{L}$ -Mul can achieve higher precision beyond `fp8_e4m3` with 5-bit mantissa operands in practice. We support both claims with estimated errors detailed in Appendix A.

### 2.3.2 GATE COMPLEXITY ESTIMATION

In this section, we make a rough estimation for the amount of gate-level computations needed by  $\mathcal{L}$ -Mul and `fp8` multiplications. Multiplying two `fpn_eimj` number require the following computation: sign prediction, exponent addition with offset, a  $j + 1$ -bit mantissa multiplication, and exponent rounding. The mantissa multiplication includes  $(j + 1)^2$  AND operations, 3 half adders and  $2j - 2$  full adders. The exponent rounding needs  $i$  half adders. In a regular circuit design, a full adder involves 2 AND, 2 XOR, and 1 OR. Each XOR has 4 NAND gates. As a result, a full adder consumes 11 gate-level computation, while a half adder (no incoming carry) consumes 5 gate-level computations (1 AND and 1 XOR).

In conclusion, the total amount of gate-level computation needed by `fp8 Mul` can be estimated as

$$N_{fp16}^\times \approx 584, \quad N_{fp8-e4m3}^\times \approx 325, \quad N_{fp8-e5m2}^\times \approx 296 \quad (6)$$

$\mathcal{L}$ -Mul consumes 1 XOR for sign prediction, 1 half adder, and  $k - 2$  full adders. The total gate count needed by 16-bit and 8-bit  $\mathcal{L}$ -Mul can be estimated as follows,

$$N_{eimj}^{\mathcal{L}\text{-mul}} = N_1^\oplus + N_{int(i+j)}^+ + N_{int8}^+ \quad (7)$$

$$N_{fp16}^{\mathcal{L}\text{-mul}} \approx 256, \quad N_{fp8}^{\mathcal{L}\text{-mul}} \approx 157$$

$\mathcal{L}$ -Mul with `fp8_e4m3` and `fp8_e5m2` operands have similar complexity since exponent offsets are typically implemented by 8-bit unsigned integer adders. As estimated, `fp16`  $\mathcal{L}$ -Mul requires less gates than `fp8` multiplications, and `fp8`  $\mathcal{L}$ -Mul is significantly more efficient.

To summarize the error and complexity analysis,  $\mathcal{L}$ -Mul is both more efficient and more accurate than `fp8` multiplication.

## 3 EXPERIMENTS

To prove the theoretical precision estimation and find out how  $\mathcal{L}$ -Mul-based LLMs perform on real tasks, we conducted experiments on various benchmarks with different transformer-based large language models. We evaluated `Llama-3.1-8b-Instruct` (Dubey et al., 2024),

mistral-7b-v0.3-Instruct (Jiang et al., 2023), Gemma2-2b-It (Team et al., 2024), and Llava-v1.5-7b (Liu et al., 2024) models, and found that the proposed method can replace different modules in transformer layers under fine-tuning or training-free settings. In this section, we first introduce the benchmarks and tasks used for evaluation, then compare the numerical error of the  $\mathcal{L}$ -Mul algorithm against models with fp8 parameters. We also report the benchmarking results of LLMs under different precision settings.

### 3.1 TASKS

**Massive Multitask Language Understanding (MMLU)** (Hendrycks et al., 2020) contains 57 multi-choice natural language understanding tasks covering various high-school and college subjects. With 5 few-shot examples, the LLMs for evaluation are required to find the most appropriate answer option to each question. The benchmark focuses on evaluating the language understanding and knowledge abilities related to given subjects.

**BigBench-Hard (BBH)** (Srivastava et al., 2023) contains a set of complex symbolic tasks to evaluate the structural and logic reasoning abilities of language models. In this work, we select a subset of 17 multi-choice tasks to evaluate Llama and Mistral LLMs. We evaluate language models under the few-shot prompting setting for all BBH tasks.

**Common Sense.** We put together a set of 5 question answering tasks to evaluate the commonsense knowledge reasoning ability of LLMs. The set of task includes ARC-Challenge (Clark et al., 2018), CSQA (Saha et al., 2018), OBQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and SIQA (Sap et al., 2019), covering different aspects of factual and social knowledge.

**Visual Question Answering.** We select a set of multi-choice question answering tasks based on images for evaluating both vision and language understanding abilities of visual language models. The tasks include VQAv2 (Goyal et al., 2017), VizWiz (Gurari et al., 2018), and TextVQA (Singh et al., 2019), containing both unanswerable and answerable questions with different types of answers.

**Visual Instruction following.** We test the instruction following ability of Llava-v1.5-7b model with the Llava-bench task (Liu et al., 2024) by generating free-form responses given images and corresponding instructions. Following the official evaluation guide, we evaluate the instruction following quality with GPT4o and compare the relative performance.

**Object Hallucination.** We explore if conducting inference with lower precision infects the truthfulness of the Llava model using the POPE benchmark (Li et al., 2023), which prompt visual language models with a sequence of yes/no questions about positive and negative objects.

**GSM8k** (Cobbe et al., 2021) consists of 8.5k human-crafted grade school math problems, with a test split of 1,000 problems designed to evaluate the arithmetic capabilities of language models. We conduct experiments on GSM8k in two different settings. In the training-free setting, we assess LLMs with few-shot, chain-of-thought prompting (Wei et al., 2022). Additionally, we fine-tune the Gemma2-2b-It model on the training split and evaluate its performance in a zero-shot setting.

### 3.2 PRECISION ANALYSIS

**Selection of  $l(k)$ .** We first visualize the mean square errors obtained by different  $l(k)$  selections with different models on the GSM8k dataset in Figure 3. In the plot, we highlight the  $l(k)$  configurations that leads to lower average error than float8\_e4m3 multiplications in model inference in red, and the  $k, l(k)$  combinations leading to an error between e4m3 and e5m2 are underlined. In both models,  $\mathcal{L}$ -Mul with 3-bit mantissas is more accurate than fp8\_e5m2 and  $\mathcal{L}$ -Mul with 4-bit mantissas achieves comparable or lower error than fp8\_e4m3.

**Mantissa size.** In section 2.3.1, we argued that the error expectation of  $\mathcal{L}$ -Mul can be lower than multiplying fp8\_e4m3 multiplication while using less computation resource than multiplying fp8\_e5m2 numbers. We hereby confirm the correctness of our theoretical precision estimates for the  $\mathcal{L}$ -Mul algorithm with experimental analysis. The average errors of Llama and Gemma models are illustrated in Figure 4.

The experiments demonstrated that across various sizes of LLMs, the  $\mathcal{L}$ -Mul algorithm using 6-bit mantissa FP operands approximates the lowest average error, significantly outperforming both fp8

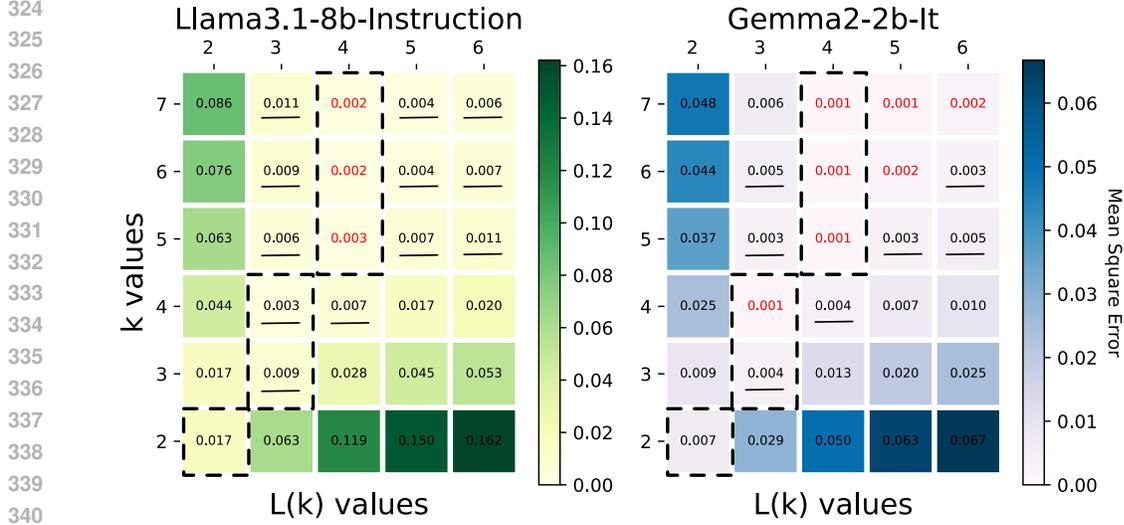


Figure 3: Mean square errors obtained by different  $l(k)$  selections on Llama and Gemma models. The combinations achieving higher precision than  $\text{fp8\_e4m3}$  are highlighted in read, and those outperforming  $\text{fp8\_e5m2}$  are underlined. When  $k = 4$  and  $l(k) = 3$ , the average error of the llama model is slightly lower but very close to  $\text{fp8\_e4m3}$ .

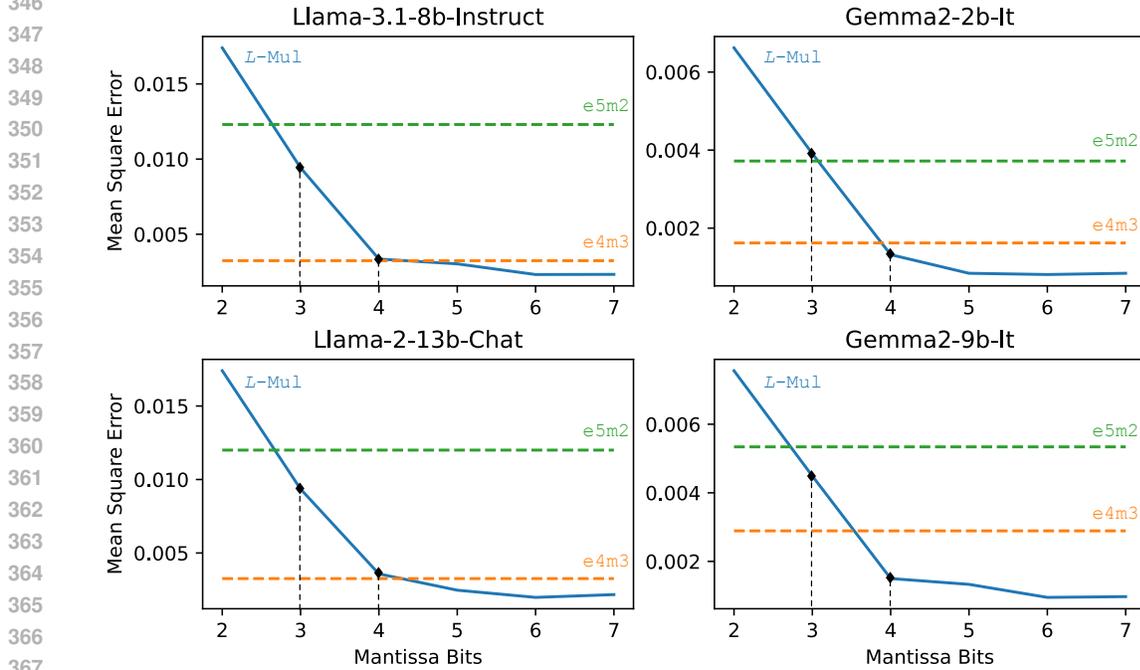


Figure 4: Comparing the error levels of linear-complexity multiplication ( $\mathcal{L}$ -Mul) against the number of mantissa bits comparing with 8-bit FP multiplication operation in different formats.

formats. Additionally, the 3- and 4-bit mantissa  $\mathcal{L}$ -Mul achieved accuracy on par with or exceeding that of  $\text{fp8\_e5m2}$  and  $\text{fp8\_e4m3}$  multiplication operations, respectively.

In the IEEE 754 format (with a 1-bit sign and a 5-bit exponent), using a 6-bit mantissa is equivalent to rounding  $\text{fp16}$  numbers down to  $\text{fp12}$ . By applying the complexity estimation method outlined in Equation (7), we can compute the gate count for 12-bit  $\mathcal{L}$ -Mul operations as follows:

$$N_{12}^{\mathcal{L}\text{-mul}} \approx 201 < N_{\text{fp8}}^{\times} \approx 300 \quad (8)$$

The experimental results further confirm that  $\mathcal{L}$ -Mul is more efficient and accurate than  $\text{fp8}$  multiplication. Although we estimated gate counts as an indicator of computational complexity, the actual difference in energy cost is greater than the complexity gap suggests. Based on the energy consumption reported in Horowitz (2014), an  $\text{fp8}$  multiplication consumes approximately 0.25 pJ to 0.4 pJ, while a 16-bit  $\mathcal{L}$ -Mul uses around 0.06 pJ of energy.

### 3.3 BENCHMARKING

In this section, we demonstrate that  $\mathcal{L}$ -Mul can replace tensor multiplications in the attention mechanism without any loss of performance, whereas using  $\text{fp8}$  multiplications for the same purpose degrades inference accuracy. This indicates that we can achieve the same model inference performance while reducing the energy cost of attention computations by 80%. Additionally, we present the full-model fine-tuning performance when all tensor multiplication operations are replaced with  $\mathcal{L}$ -Mul on the GSM8k benchmark.

**Textual tasks.** Table 2 presents the evaluation results of the Llama and Mistral models on various natural language benchmarks, including MMLU, BBH, ARC-C, CSQA, PIQA, OBQA, and SIQA. In these experiments, the matrix multiplications in the attention layers, both before and after the softmax operation, were replaced with 8-bit tensor computations in different formats or  $\mathcal{L}$ -Matmul following the implementation we discussed in Equation (3).

Precision	BBH	MMLU	ARC-R	CSQA	OBQA	PIQA	SIQA	Avg.
Mistral-7b-Instruction-v0.3								
BFloat16	55.85	62.20	75.94	71.42	76.20	80.74	44.83	69.83
Float8_e4m3	55.16	62.18	75.39	71.25	76.00	80.47	44.63	69.55
Float8_e5m2	53.20	61.75	74.91	71.25	74.40	79.76	44.52	68.97
$\mathcal{L}$ -Mul	55.87	62.19	76.11	71.09	76.60	80.52	45.34	69.93
Llama-3.1-8B-Instruct								
BFloat16	70.79	68.86	82.51	74.53	84.20	84.00	45.96	74.24
Float8_e4m3	69.91	68.16	81.66	74.28	82.20	83.51	45.34	73.40
Float8_e5m2	62.94	66.61	80.12	73.30	79.40	81.07	45.39	71.86
$\mathcal{L}$ -Mul	70.78	68.54	82.17	74.28	84.20	83.30	46.06	74.00

Table 2: Comparing the attention mechanism implemented with 16- and 8-bit tensor multiplication operations and  $\mathcal{L}$ -Mul approximations. Note that the  $\mathcal{L}$ -Mul computations cost significantly less resource than  $\text{fp8}$  tensors.

The results indicate that  $\mathcal{L}$ -Mul not only requires significantly fewer computational resources but also delivers higher precision than  $\text{float8-e4m3}$  tensors in 12 out of 14 experiments using Mistral and Llama models. This leads to a minimal performance gap when compared to  $\text{bf16}$  inference. On average, across the two models, the performance difference between  $\text{bf16}$  and  $\mathcal{L}$ -Mul is just 0.07%. These findings suggest that matrix multiplication operations in the attention mechanism can be seamlessly replaced with the  $\mathcal{L}$ -Mul algorithm without any loss of accuracy or the need for additional training.

**GSM8k.** We evaluated the performance of three language models—Mistral-7b-Instruct-v0.3, Llama3.1-7b-Instruct, and Gemma2-2b-It—on the GSM8k dataset using few-shot prompting and  $\mathcal{L}$ -Mul-based attention. The models were tested under different numerical precision formats:  $\text{bf16}$ ,  $\text{fp8-e4m3}$ ,  $\text{fp8-e5m2}$ , and the  $\mathcal{L}$ -Mul method. The results are summarized in Table 3.

Notably, the  $\mathcal{L}$ -Mul-based attention mechanism slightly improved the average performance compared to the  $\text{bf16}$  baseline. Mistral-7b-Instruct-v0.3 and Gemma2-2b-It both exhibited improved accuracies with  $\mathcal{L}$ -Mul, achieving 52.92% and 47.01% respectively. Llama3.1-7b-Instruct’s accuracy with  $\mathcal{L}$ -Mul was slightly lower than its  $\text{bf16}$  performance but still higher than with  $\text{fp8-e4m3}$  and  $\text{fp8-e5m2}$ . On contrary, rounding the tensors in the attention computation to  $\text{fp8-e5m2}$  leads to a significant performance drop although it’s more complicated than  $\mathcal{L}$ -Mul.

Model	Bfloat16	Float8_e4m3	Float8_e5m2	$\mathcal{L}$ -Mul
Mistral-7b-Instruct-v0.3	52.54	52.39	50.19	52.92
Llama3.1-7b-Instruct	76.12	75.44	71.80	75.63
Gemma2-2b-It	45.87	45.94	44.43	47.01
<b>Average</b>	58.17	57.92	55.47	58.52

Table 3: GSM8k accuracy with Mistral, Llama, and Gemma models with few-shot prompting and attention mechanism implemented in different precision levels.

**Vision-language tasks.** The performance of the Llava-v1.5-7b model on VQA, object hallucination, and instruction following tasks are shown in Table 4. Similar to the experiments on language tasks, the attention computation is conducted with different precision/methods while other linear transformation layers are unchanged. Except for TextVQA where the accuracy gap is 0.5%, the performance of  $\mathcal{L}$ -Mul and BFloat16 attentions are comparable. The VQA tasks are evaluated with the official evaluation scripts and the Llava-Bench results are generated by GPT-4o.

Task	POPE				comp.	Llava-Bench			TextVQA	
	Split	rand.	adv.	pop.		all	conv.	detail.		all
BFloat16		86.20	83.17	85.13	84.83	66.80	57.60	41.40	57.50	57.90
$\mathcal{L}$ -Mul		86.57	83.19	85.34	85.03	64.90	58.70	43.30	57.50	57.41

Task	Split	VQAv2			yes/no	VizWiz			all	
		yes/no	num.	other		all	num.	unans.		other
BFloat16		91.88	59.04	70.56	78.03	77.19	45.24	71.75	38.19	49.31
$\mathcal{L}$ -Mul		91.78	58.93	70.73	78.06	78.54	50.48	73.78	38.41	50.16

Table 4: Evaluating the performance of different attention implementation on the Llava-v1.5-7b model. VQAv2, VizWiz, and TextVQA are visual question answering tasks, POPE evaluates object hallucination, and Llava-Bench assesses the instruction following ability scored by GPT-4o.

**$\mathcal{L}$ -Mul with fewer bits.** In this section, we explore how  $\mathcal{L}$ -Mul-based attention precision influences the overall model performance using the MMLU benchmark with Mistral and Llama models. We implement the attention mechanism with  $\mathcal{L}$ -Mul and only keep the first  $k$  bits of the operand tensors. The results of  $\mathcal{L}$ -Mul attention with different precision are listed in Table 5. As expected, using  $\mathcal{L}$ -Mul with a 4-bit mantissa achieves performance comparable to or slightly better than that of bfloat16 and fp8\_e4m3. However, performance drops proportionally to the estimated error depicted in Figure 4. When  $k = 3$ , both models significantly outperform their fp8\_e5m2 counterparts, with the Llama model’s performance remaining close to that of fp8\_e4m3. When  $k = 2$ , the Llama model’s performance is comparable to that of fp8\_e5m2 rounding. This suggests that with the Llama model, we can perform  $\mathcal{L}$ -Mul directly on fp8 models without compromising performance.

Model	e4m3		e5m2		$k = 4$			$k = 3$			$k = 2$		
	8bit	Acc.	e4m3	e5m2	$\mathcal{L}$ -Mul								
Mistral	62.18	61.75	62.16	62.06	61.08								
Llama	68.16	66.61	68.43	68.12	66.67								

Table 5: The performance of Mistral models with attention mechanism implemented with  $k$ -bit tensor  $\mathcal{L}$ -Mul.

Table 6: Zero-shot fine-tuned Gemma2-2b models with 8-bit accumulation precision.  $\mathcal{L}$ -Mul uses fp8\_e4m3 inputs.

**Full-model fine-tuning.** To further explore the impact of the  $\mathcal{L}$ -Mul algorithm, we go beyond implementing attention layers with  $\mathcal{L}$ -Mul by replacing all multiplication operations—including matrix multiplications in linear transformations, element-wise multiplications, and those within attention layers—with fp8\_e4m3  $\mathcal{L}$ -Mul for the Gemma2-2b-It model. We then fine-tune the updated model on the training set of the GSM8k corpus and evaluate both the fine-tuned fp8 and  $\mathcal{L}$ -Mul

486 models under a zero-shot setting on the GSM8k test set. Note that the  $\mathcal{L}$ -Mul operations in this ex-  
 487 periment takes operands with 3-bit mantissas ( $k = 3$ ) and the accumulation precision is  $\text{fp8\_e4m3}$   
 488 to explore an extremely efficient setting.

489 The experimental results demonstrate that a fine-tuned  $\text{fp8\_e4m3}$   $\mathcal{L}$ -Mul model achieves perfor-  
 490 mance comparable to a standard fine-tuned  $\text{fp8\_e4m3}$  model under  $\text{fp8}$  accumulation precision.  
 491 This suggests that  $\mathcal{L}$ -Mul can enhance training efficiency without compromising the fine-tuned  
 492 model’s performance. Moreover, it reveals the potential of training  $\mathcal{L}$ -Mul native LLMs for accurate  
 493 and energy-efficient model hosting.

## 494 4 RELATED WORK

495 Reducing the computation needed by neural networks while maintain the performance is an impor-  
 496 tant problem which entailed multiple research directions. Typical methods include neural network  
 497 pruning, quantization, and improved tensor I/O implementations.

501 **Pruning.** Neural network pruning focuses on improving the inference efficiency by reducing the  
 502 number of connections among layers (Han et al., 2015a;b; Wang et al., 2020). Neural network prun-  
 503 ing methods usually involves training. After important weights are identified, the neural networks  
 504 are re-trained to further update the selected weights for specific tasks. Different from model pruning,  
 505 the method we proposed is designed for general tasks, requiring no task-specific re-training.

506 **Optimizing tensor I/O.** On regular GPUs, moving tensors between GPU SRAM and high-  
 507 bandwidth memory (HBM) is the main bottleneck of time and energy consumption. Reducing the  
 508 I/O operations in transformer models and making the best use of the HBM can significantly improve  
 509 the efficiency of AI training and inference (Dao et al., 2022; Dao; Kwon et al., 2023). Our method,  
 510 which focuses on optimizing arithmetic operations, is orthogonal to this direction.

511 **Rounding and quantization.** Standard neural network weights are stored as 32-bit or 16-bit FP  
 512 tensors. However, the full-sized weights takes a considerable amount of GPU memory. To improve  
 513 the storage efficiency, both weights storage and computation can be conducted in a lower precision,  
 514 for example, using 16-bit, 8-bit, or 4-bit FP and Int ( $\text{fp16}$ ,  $\text{bf16}$  (Kalamkar et al., 2019),  $\text{fp8\_e4m3}$ ,  
 515  $\text{fp8\_e5m2}$  (Micikevicius et al., 2023),  $\text{int8}$  (Dettmers et al., 2022),  $\text{fp4}$ , and  $\text{int4}$  (Dettmers et al.,  
 516 2024)) tensors to represent model weights. Inference with lower-bit parameters usually hurts the  
 517 computation accuracy and impacts the performance of pretrained models, and Integer-based quan-  
 518 tization methods spend significant time to handle outlier weights. comparing to the quantization  
 519 methods, our method requires less computation but achieves higher accuracy.

## 520 5 FUTURE WORK

521 To unlock the full potential of our proposed method, we will implement the  $\mathcal{L}$ -Mul and  $\mathcal{L}$ -Matmul  
 522 kernel algorithms on hardware level and develop programming APIs for high-level model design.  
 523 Furthermore, we will train textual, symbolic, and multi-modal generative AI models optimized for  
 524 deployment on  $\mathcal{L}$ -Mul native hardware. This will deliver high-speed and energy-efficient AI host-  
 525 ing solutions, reducing the energy cost for data centers, robotics, and a wide spectrum of edge-  
 526 computing devices.

## 527 6 CONCLUSION

528 In this paper, we introduced  $\mathcal{L}$ -Mul, an efficient algorithm that approximates floating-point multi-  
 529 plication using integer addition. We first demonstrated that the algorithm exhibits linear complexity  
 530 relative to the bit size of its floating-point operands. We then showed that the expected accuracy of  
 531  $\mathcal{L}$ -Mul surpasses that of  $\text{fp8}$  multiplications while requiring significantly less computational power.  
 532 To assess the practical impact of  $\mathcal{L}$ -Mul, we evaluated it on natural language, vision, and mathe-  
 533 matics benchmarks using popular language models. Our experiments indicate that  $\mathcal{L}$ -Mul outper-  
 534 forms 8-bit transformers with lower computational consumption and achieves lossless performance  
 535 when applied to computation-intensive attention layers without additional training. Based on this  
 536 evidence, we argue that tensor multiplications in language models can be effectively implemented  
 537 using  $\mathcal{L}$ -Mul to preserve performance while enabling energy-efficient model deployment.

## REFERENCES

- 540  
541  
542 Dennis Abts, Garrin Kimmell, Andrew Ling, John Kim, Matt Boyd, Andrew Bitar, Sahil Parmar,  
543 Ibrahim Ahmed, Roberto DiCecco, David Han, et al. A software-defined tensor streaming mul-  
544 tiprocessor for large-scale machine learning. In *Proceedings of the 49th Annual International*  
545 *Symposium on Computer Architecture*, pp. 567–580, 2022.
- 546 Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Ale-  
547 man, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical  
548 report. *arXiv preprint arXiv:2303.08774*, 2023.
- 549  
550 Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical com-  
551 monsense in natural language. In *Proceedings of the AAI conference on artificial intelligence*,  
552 volume 34, pp. 7432–7439, 2020.
- 553 Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- 554  
555 Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100  
556 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.
- 557  
558 Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and  
559 Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge.  
560 *arXiv preprint arXiv:1803.05457*, 2018.
- 561  
562 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,  
563 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to  
564 solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- 565  
566 Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The*  
567 *Twelfth International Conference on Learning Representations*.
- 568  
569 Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-  
570 efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*,  
571 35:16344–16359, 2022.
- 572  
573 Alex de Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194,  
574 2023.
- 575  
576 Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix  
577 multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:  
578 30318–30332, 2022.
- 579  
580 Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning  
581 of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.
- 582  
583 Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha  
584 Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models.  
585 *arXiv preprint arXiv:2407.21783*, 2024.
- 586  
587 Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa  
588 matter: Elevating the role of image understanding in visual question answering. In *Proceedings*  
589 *of the IEEE conference on computer vision and pattern recognition*, pp. 6904–6913, 2017.
- 590  
591 Danna Gurari, Qing Li, Abigale J Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and  
592 Jeffrey P Bigham. Vizwiz grand challenge: Answering visual questions from blind people. In  
593 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3608–3617,  
2018.
- 594  
595 Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks  
596 with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.
- 597  
598 Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for  
599 efficient neural network. *Advances in neural information processing systems*, 28, 2015b.

- 594 Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob  
595 Steinhardt. Measuring massive multitask language understanding. In *International Conference*  
596 *on Learning Representations*, 2020.
- 597 Mark Horowitz. 1.1 computing’s energy problem (and what we can do about it). In *2014 IEEE*  
598 *international solid-state circuits conference digest of technical papers (ISSCC)*, pp. 10–14. IEEE,  
599 2014.
- 600 Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot,  
601 Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al.  
602 Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- 603 Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa,  
604 Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of  
605 a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer*  
606 *architecture*, pp. 1–12, 2017.
- 607 Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee,  
608 Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen,  
609 et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.
- 610 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph  
611 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model  
612 serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Prin-*  
613 *ciples*, pp. 611–626, 2023.
- 614 Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Xin Zhao, and Ji-Rong Wen. Evaluating object hallu-  
615 cination in large vision-language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.),  
616 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp.  
617 292–305, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/  
618 v1/2023.emnlp-main.20. URL <https://aclanthology.org/2023.emnlp-main.20>.
- 619 Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction  
620 tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recogni-*  
621 *tion*, pp. 26296–26306, 2024.
- 622 Paulius Micikevicius, Stuart Oberman, Pradeep Dubey, Marius Cornea, Andres Rodriguez, Ian Bratt,  
623 Richard Grisenthwaite, Norm Jouppi, Chiachen Chou, Amber Huffman, et al. Ocp 8-bit floating  
624 point specification (ofp8). *Open Compute Project*, 2023.
- 625 Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct elec-  
626 tricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference*  
627 *on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.
- 628 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor  
629 Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-  
630 performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- 631 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language  
632 models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- 633 Amrita Saha, Vardaan Pahuja, Mitesh Khapra, Karthik Sankaranarayanan, and Sarath Chandar.  
634 Complex sequential question answering: Towards learning to converse over linked question an-  
635 swer pairs with a knowledge graph. In *Proceedings of the AAAI conference on artificial intelli-*  
636 *gence*, volume 32, 2018.
- 637 Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Common-  
638 sense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical*  
639 *Methods in Natural Language Processing and the 9th International Joint Conference on Natural*  
640 *Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.
- 641 Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh,  
642 and Marcus Rohrbach. Towards vqa models that can read. In *Proceedings of the IEEE/CVF*  
643 *conference on computer vision and pattern recognition*, pp. 8317–8326, 2019.

- 648 Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam  
649 Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the  
650 imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions*  
651 *on Machine Learning Research*, 2023.
- 652 Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu,  
653 Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly  
654 capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- 655 Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhu-  
656 patiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma  
657 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.
- 658 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée  
659 Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and  
660 efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- 661 A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- 662 Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In  
663 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*  
664 *(EMNLP)*, pp. 6151–6162, 2020.
- 665 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny  
666 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in*  
667 *neural information processing systems*, 35:24824–24837, 2022.

## 672 A ERROR ESTIMATION

673 We calculate the error expectations with different  $(n, k)$  combinations as follows in Table 7. The  
674 values are calculated with the actual parameters of Mistral, Llama, and Gemma models. For even  
675 distribution, we use the expectations introduced in Section 2.3.1. For real distribution, we estimate  
676 the average value of possible operands using the parameters of five popular pretrained LLMs.

679 <b>K values</b>		<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
680 Even	$abs[f_1(n = 7, k)]$	0.68	0.35	0.17	0.081	0.035	0.012
681 Distribution	$abs[f_1(n = 7, k) + f_2(k)]$	0.68	0.43	0.30	0.24	0.20	0.19
682 Real	$abs[f_1(n = 7, k)]$	0.61	0.33	0.16	0.077	0.033	0.011
683 Distribution	$abs[f_1(n = 7, k) + f_2(k)]$	0.16	0.18	0.18	0.12	0.15	0.14

684 Table 7: Average error expectation with five different language models on floating point multiplica-  
685 tion and  $\mathcal{L}$ -Mul with different rounding representations when the full precision is `BFloat16`.  $K$   
686 stands for the bit number of the operand mantissa.

687 We find that when the operands are distributed evenly,  $\mathcal{L}$ -Mul is more accurate than  
688 `float8_e5m2` multiplications. However with real models,  $\mathcal{L}$ -Mul can achieve higher precision  
689 than `float8_e4m3` calculations.