Lookup Table Refactoring: Towards Efficient Logarithmic Number System Addition for Large Language Models

Xinkuang Geng*, Siting Liu†, Hui Wang*, Jie Han‡, Honglan Jiang*
* Department of Micro-Nano Electronics, Shanghai Jiao Tong University, Shanghai, China
† School of Information Science and Technology, ShanghaiTech University, Shanghai, China
‡ Department of Electrical and Computer Engineering, University of Alberta, Alberta, Canada
xinkuang@sjtu.edu.cn, liust@shanghaitech.edu.cn, ihuiwang@sjtu.edu.cn, jhan8@ualberta.ca, honglan@sjtu.edu.cn

***Abstract*—Compared to integer quantization, logarithmic quantization aligns more effectively with the long-tailed distribution of data in large language models (LLMs), resulting in lower quantization errors. Moreover, the logarithmic number system (LNS) employs a fixed-point adder to perform multiplication, indicating a potential reduction in computational complexity for LLM accelerators that require extensive multiply-accumulate (MAC) operations. However, a key bottleneck is that LNS addition requires complex nonlinear functions, which are typically approximated using lookup tables (LUTs). This study aims to reduce the hardware resources needed for LUTs in LNS addition while maintaining high precision. Specifically, we investigate the specific nature of addition operations within LLMs; the relationship between the hardware parameters of the LUT and the computing errors is then mathematically derived. Based on these insights, we propose LUT refactoring to optimize the LUT for enhanced efficiency in LNS addition. With 10.93% and 19.78% reductions in area-delay product (ADP) and power-delay product (PDP), respectively, LUT refactoring results in an accuracy improvement of up to 33.5% in LLM benchmarks compared to the naive design. When compared to integer quantization, our method achieves higher accuracy while reducing area by 18.27% and power by 42.61%.**

***Index Terms*—logarithmic number system, lookup table, large language model, approximate computing, error analysis**

I. Introduction

With the continuous development of machine learning, large language models (LLMs) are applied to various tasks and show excellent performance [1]–[3]. Whether in servers or edge devices, optimizing the computation of large volumes of data is of great significance. Quantization is an effective technique for compressing and approximating LLMs by mapping data and computations from the floating-point domain into a number system containing fewer discrete values. A common quantization target is the integer [4]. In this case, the multiply-accumulate (MAC) operations required in LLMs can be efficiently implemented using integer MAC units. Since data in LLMs often follow a long-tailed distribution, the logarithmic number system (LNS) [5] effectively represents values around zero and is thus a better choice [6]–[8].

In the LNS, data are represented by their logarithm values (commonly base two). Thus, LNS multiplication is simplified into fixed-point addition, making it hardware-efficient. On the contrary, the logarithmic addition involves complex nonlinear functions, which are usually implemented using lookup tables (LUTs) [9].

In [10], the original LUT in LNS addition is split for fitting the nonlinear function into multiple smaller sub-LUTs. By inserting registers, only one of the sub-LUTs is activated for each operation, thereby reducing the power. However, it requires sequential logic in a single adder, which not only incurs additional area costs but also complicates the integration with existing accelerator architectures.

[11] employs Mitchell's algorithm [12] to approximate two addends as fixed-point numbers, which are then converted back to LNS format after addition and normalization. Compared to the LUT-based LNS adder, [11] offers greater hardware efficiency but introduces larger errors. [13] improves upon [11] by employing compact piecewise linear approximation to reduce the conversion error, though this inevitably results in greater hardware overhead.

The accumulation of LNS values in [8] is achieved by first converting the numbers in LNS format to their fixed-point representations and then performing accumulation in fixed-point format. Although this method is error-free in accumulation, it entails higher hardware costs. [7], [14] also employed this method to deploy logarithmic quantization for convolutional neural networks. As a more precise alternative to avoid the approximation error in LNS addition, this method presents some drawbacks: 1. Additional logic is required to facilitate the conversion between LNS and fixed-point formats. 2. High-bit-width accumulation registers are necessary, which may counteract the advantages of using LNS for representing a wide dynamic range with a reduced bit width.

In prior works, optimizations for LNS addition have focused solely on individual computations without taking into account the characteristics of the applications. Notably, in LLMs, addition operations exhibit a significant bias in their inputs. Specifically, an adder is often used as an accumulator in MAC operations, where different additions are not independent. As the accumulation progresses, the error characteristics of the addition would change from a statistical perspective. Thus, conventional metrics for evaluating the performance of adders may no longer be applicable and may lead to suboptimal

This work was supported in part by the National Natural Science Foundation of China under grant number 62374108, and in part by Alberta Innovates under grant number RES0053965.

designs for LNS addition in LLMs.

This paper aims to design an efficient LNS adder for the inference of quantized LLMs. Our major contributions are summarized as follows.

- Based on the statistical characteristics of the addition operations in LLMs, we point out that the disparity between the two addends is significant in most cases.
- The relationship between the parameters of the LUT in LNS addition and the accumulation error is derived.
- According to the formula of the error, LUT refactoring is proposed to optimize the LNS addition, which consists of three techniques: precision enhancement, entry pruning, and progressive precision reduction.
- The proposed LNS addition is evaluated on a MAC unit for various LLM benchmarks. Our method enhances the average accuracy of the LNS adder in LLAMA-2-7B by 19.2% while also reducing hardware overhead.

II. Preliminaries

A. LLM Quantization

To reduce memory usage and computational complexity, quantization is commonly performed in LLMs by discretizing the high-precision floating-point data into a lower bit width. The quantization process can be regarded as approximating a real number using its nearest alternative in a given set of discrete values.

Data in LLMs often follow a long-tailed distribution. As shown in Fig. 1, we visualize the data in the first transformer block of LLAMA-2-7B [2]. It can be seen that a large number of the data are concentrated around zero. As shown by the vertical bars in Fig. 1, logarithmic quantization naturally satisfies this characteristic, enabling a more precise representation of smaller data. Prior works [6], [7] also demonstrate that LNS is more appropriate in neural network quantization.

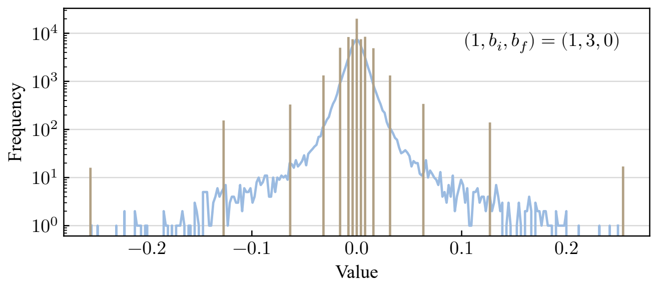


Fig. 1: Quantization for data in LLMs. The blue curve represents the distribution of the weights in the first transformer block in LLAMA-2-7B. The vertical bars represent the frequency of the corresponding logarithmic quantization results.

B. Logarithmic Number System

In LNS, a real number X is represented in signed magnitude form (x_{sign}, x), where x_{sign} is a 1-bit sign and x is the logarithm of $|X|$. x is a fixed-point number consisting of an integer part with b_i bits and a fractional part with b_f bits. We represent the format of LNS as $(1, b_i, b_f)$, which indicates the bit width of each part. The mapping relationship between a real number and its LNS encoding can be expressed as $X = (-1)^{x_{\text{sign}}} \cdot 2^x$. Typically, an additional zero flag is necessary because zero does not have a logarithm result. However, in low-bit-width quantization, each bit is precious, and a zero-flag bit would waste half of the encoding space. Therefore, in the LNS discussed in this paper, we identify $X = 0$ by setting $x = 0$. This means that the element $X = \pm 2^0$ cannot be represented. This does not affect the quantization resolution as scaling is already included in the quantization process, and 2^x can naturally be replaced with $2^{x+\text{ulp}}$ by changing the scaling factor, where $\text{ulp} = 2^{-b_f}$. In this case, the representation range of LNS is defined as

$$\mathcal{R}_{\text{LNS}} = [-2^{2^{b_i}-\text{ulp}}, -2^{\text{ulp}}] \cup \{0\} \cup [2^{\text{ulp}}, 2^{2^{b_i}-\text{ulp}}]. \quad (1)$$

When performing LNS multiplication, the cases in which either operand is zero are bypassed, and the absolute value of the logarithm of the product can be obtained simply by fixed-point addition as $p = \log_2(2^x \cdot 2^y) = x + y$. The sign of the product can be calculated by using the XOR operation.

Similarly, the addition operation in LNS can be expressed as

$$s = \log_2(2^x \pm 2^y) = \max(x, y) + \underbrace{\log_2(1 \pm 2^{-|d|})}_{\kappa_\pm(|d|)}, \quad (2)$$

where $d = y - x$. The choice of $\pm$ is determined by the sign of the original inputs X and Y, and the sign of the sum is set to be aligned with the operand with the larger absolute value. The core of implementing LNS addition is to approximate the nonlinear functions $\kappa_\pm$, as shown in Fig. 2.

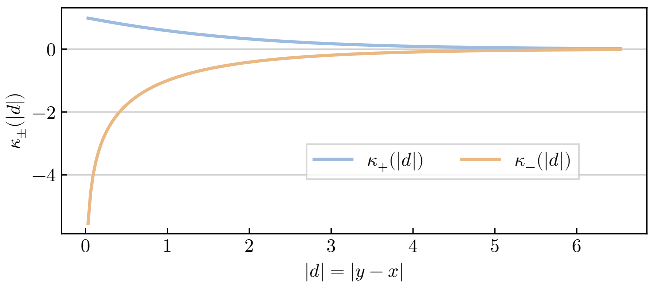


Fig. 2: Nonlinear functions required in LNS addition.

In the naive implementation of LNS addition [9], two LUTs are used to approximate κ_+ and κ_-, respectively. Fig. 3 shows the circuit for the implementation of (2).

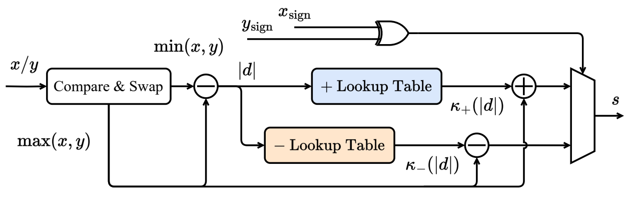


Fig. 3: LNS addition using LUTs to approximate κ_+ and κ_-.

III. Error Analysis

A. Characteristics of the Additions in LLMs

In LLMs, the dominating computations can be characterized as matrix multiplication. Each element in the output matrix is produced by a vector inner product as $Y = \sum_i A_i B_i$. For the commonly used systolic array architecture [15], each inner product is continuously completed within a single processing element, avoiding extra memory accesses required for storing and loading the partial sums that have a high bit width. In this case, the vector inner product can be considered as a continuous MAC operation as

$$Y_{n+1} = Y_n + A_n B_n = Y_n + X_n. \tag{3}$$

Since X_n is the product of the inputs A_n and B_n, it is independent of the accumulation process. However, the other addend, Y_n, is strongly correlated with the accumulation process. We consider that Y_n is obtained by summing n independent and identically distributed random variables (X_1 to X_n), which has a distribution with expected value μ and variance σ^2. As per the central limit theorem, when n is large, Y_n approximately follows the normal distribution, i.e.,

$$Y_n \sim N(n\mu, n\sigma^2). \tag{4}$$

Based on the characteristics of the folded normal distribution derived in [16], the expectation of $|Y_n|$ can be represented as

$$\mathbb{E}(|Y_n|) = \sqrt{\frac{2n}{\pi}}\sigma e^{-\frac{n}{2}\frac{\mu^2}{\sigma^2}} + n\mu\,\mathrm{erf}(\sqrt{\frac{n}{2}}\frac{\mu}{\sigma}), \tag{5}$$

which increases continuously with the increase of n. Note that X_n is the product of quantized inputs and also a bounded number, so $|Y_n|$ has a greater expected value than $|X_n|$ when the number of the accumulation operations n is large enough.

When considering the accumulation process in LNS, d in (2) can naturally be used as the relative magnitude between the two addends as

$$d = y - x = \log_2|Y| - \log_2|X| = \log_2\frac{|Y|}{|X|}. \tag{6}$$

Vectors extracted from OPT-125M [1] and LLAMA-2-7B [2] are used to conduct a statistical analysis of the distributions of d in vector inner products of different lengths. As can be seen in Fig. 4, the proportion of the addition operations with large d increases as the vector length grows, consistent with the aforementioned theoretical analysis.

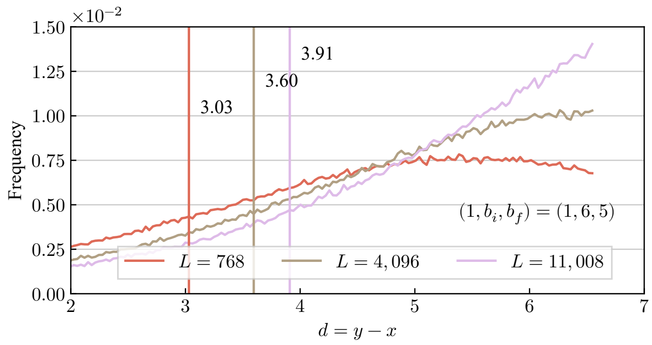


Fig. 4: The distributions of the difference between the two addends in the accumulation of the inner product vary for input vectors of different lengths L. The right of each vertical line accounts for 75% of the frequency.

Following the principle of prioritizing optimization for more frequent scenarios, in LLMs, we believe that the LNS additions with large d should be given priority consideration. Therefore, in the subsequent error analysis process, we assume

$$d = y - x \geq d_{\text{inf}}, \tag{7}$$

where d_{inf} represents the lower bound of d we consider in LNS addition. A reasonable choice for d_{inf} would be based on the 75% quantile from the statistical data, as indicated by the vertical lines in Fig. 4. Consequently, when studying the inner product of long vectors, d_{inf} can be considered positive and sufficiently large. Hence, (2) can be simplified as

$$s = y + \kappa(d). \tag{8}$$

B. Formulation of the Error for LNS Addition

When LUTs are employed to implement κ in (8), the error would inevitably arise. Ideally, the LUT can index the corresponding $\kappa(d)$ based on arbitrary d. However, since a LUT has finite capacity, it can only accommodate discretized values of d. Furthermore, each entry in the LUT has finite precision, which poses a requirement for the discretization of the function values of κ. Considering these two discretization processes, the approximation of (8) using LUTs can be expressed as

$$\tilde{s} = y + q_{b_1}(\kappa(q_{b_2}(d))), \tag{9}$$

where q_{b_1} rounds the function value κ to a fixed-point number containing only b_1 fractional bits and q_{b_2} discretizes the index d to a fixed-point number containing only b_2 fractional bits. As shown in Fig. 2, κ goes to zero when d is large enough. Hence, the LUT only needs to handle d with finite magnitudes that make the rounded function value $q_{b_1}(\kappa)$ non-zero as

$$2 \cdot |\kappa(q_{b_2}(d))| \geq 2^{-b_1}. \tag{10}$$

The rounding function q_b is expressed as

$$q_b(x) = \lfloor x \cdot 2^b \rceil \cdot 2^{-b}, \tag{11}$$

where $\lfloor\cdot\rceil$ denotes the nearest rounding operation, producing an absolute error between 0 and 0.5. Since a continuous distribution can be considered uniform in a small range (i.e., on the scale of 2^{-b} for x), we can make an approximation as $|\lfloor t \rceil - t| \sim U(0, 0.5)$, where $U(a,b)$ denotes the uniform distribution over interval from a to b with a mean of $(a+b)/2$. Consequently, the expectation of the absolute rounding error introduced by q_b can be calculated as

$$\mathbb{E}(|q_b(x) - x|) = \mathbb{E}(|\lfloor x \cdot 2^b \rceil - x \cdot 2^b|) \cdot 2^{-b} = 2^{-2-b}. \tag{12}$$

The absolute error between the approximate sum and the exact one in LNS encoding can be represented as

$$\epsilon = |\tilde{s} - s| = |q_{b_1}(\kappa(q_{b_2}(d))) - \kappa(d)|. \tag{13}$$

Here we further scale ϵ by adding and subtracting the same term $\kappa(q_{b_2}(d))$ as

$$\begin{aligned}\epsilon &= |q_{b_1}(\kappa(q_{b_2}(d))) - \kappa(d) + \kappa(q_{b_2}(d)) - \kappa(q_{b_2}(d))| \\ &\leq \underbrace{|q_{b_1}(\kappa(q_{b_2}(d))) - \kappa(q_{b_2}(d))|}_{\epsilon_1} + \underbrace{|\kappa(q_{b_2}(d)) - \kappa(d)|}_{\epsilon_2},\end{aligned} \tag{14}$$

where ϵ_1 describes the absolute rounding error of q_{b_1}, and its expectation can be calculated as

$$\mathbb{E}(\epsilon_1) = 2^{-b_1-2}. \tag{15}$$

ϵ_2 is the absolute error incurred by using the rounded value of d to obtain the function value of κ. Subsequently, some approximations are applied to analyze the impact of q_{b_2} on ϵ_2. Suppose $\varDelta$ is the rounding error of q_{b_2}, i.e.,

$$\varDelta = q_{b_2}(d) - d. \tag{16}$$

Then ϵ_2 can be expressed as

$$\epsilon_2 = |\kappa(d+\varDelta) - \kappa(d)|. \tag{17}$$

Obviously, the rounding error $\varDelta$ must be much smaller than d itself, so it is reasonable to retain only the first-order term in the Taylor expansion of $\kappa(d+\varDelta)$, which can be approximated as $\kappa(d) + \kappa'(d)\cdot\varDelta$. Hence, ϵ_2 can be approximated as

$$\epsilon_2 \approx |\kappa'(d)\cdot\varDelta| = |(1 \pm 2^{-d})^{-1} - 1| \cdot |\varDelta|. \tag{18}$$

Since we focus on the case where d is large, 2^{-d} can be considered sufficiently small. Given that $(1+x)^{\alpha} - 1$ and $\alpha \cdot x$ are equivalent infinitesimals as x approaches 0, then by substituting $x = 2^{-d}$ and $\alpha = -1$, we can obtain

$$\epsilon_2 \approx |2^{-d}| \cdot |\varDelta|. \tag{19}$$

According to the inequality in (7), ϵ_2 can be scaled as

$$\epsilon_2 \leq 2^{-d_{\inf}} \cdot |\varDelta|. \tag{20}$$

As per (16), $|\varDelta|$ indicates the absolute rounding error of d, so its expectation is given by $\mathbb{E}(|\varDelta|) = 2^{-b_2-2}$. Therefore, the expectation of ϵ_2 satisfies

$$\mathbb{E}(\epsilon_2) \leq 2^{-d_{\inf}} \cdot 2^{-b_2-2}. \tag{21}$$

Based on the above analysis, the upper bound of the expectation of the absolute error ϵ in LNS addition is given by

$$\mathbb{E}(\epsilon) \leq \mathbb{E}(\epsilon_1) + \mathbb{E}(\epsilon_2) \leq 2^{-b_1-2} + 2^{-d_{\inf}} \cdot 2^{-b_2-2}. \tag{22}$$

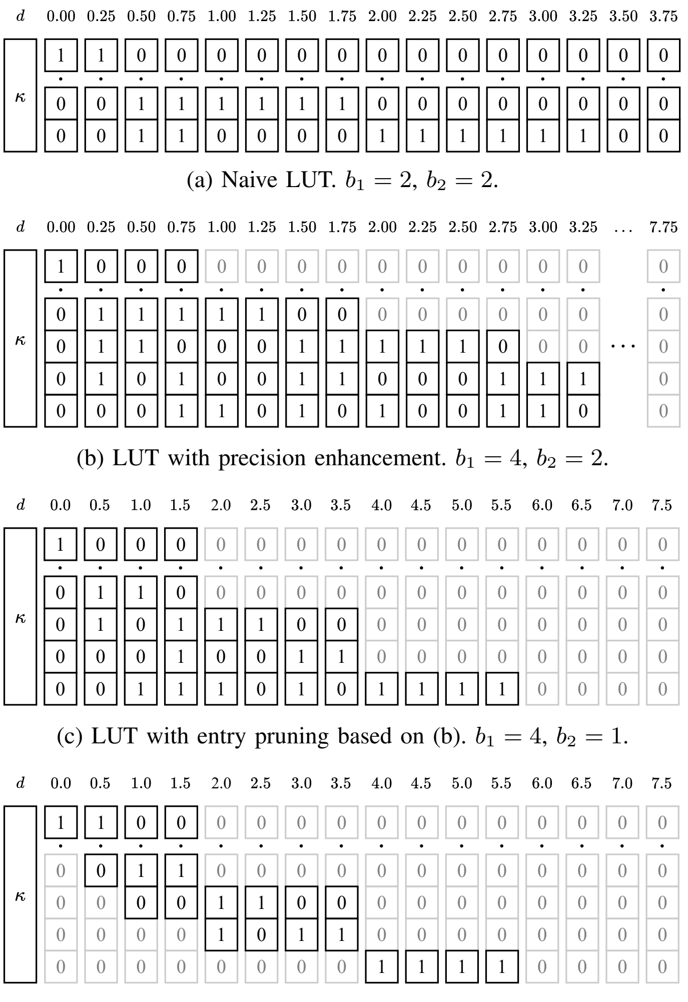
Finally, by eliminating the common constant coefficients, we obtain the upper bound of the expectation of the absolute error as

$$U_{\mathbb{E}(\epsilon)} \propto 2^{-b_1} + 2^{-d_{\inf}} \cdot 2^{-b_2}. \tag{23}$$

IV. Lookup Table Refactoring

As per the analysis of the error in LNS addition, the error is determined jointly by the precision of the LUT entry (b_1) and the granularity of the LUT index (b_2). However, in prior works [9], [10], when using the LNS adder for accumulation, it is implicitly assumed that the current accumulation result is entirely used to obtain the index d in the next cycle. That is to say, in the naive implementation, the LUT is designed with $b_1 = b_2$ by default. As shown in Fig. 5 (a), we visualize the LUT with $b_1 = b_2 = 2$, where a column corresponds to an entry. Since the entries in the LUT are all zero when $d \geq 3.50$, the LUT index d requires only 2 integer bits. Notably, only the LUT used for κ_+ is displayed as an example, i.e., the entries in Fig. 5 (a) are obtained based on (2) with $\pm$ selected as $+$.

As observed in (23), the first term introduced by q_{b_1} constitutes the dominant portion of $U_{\mathbb{E}(E)}$ given that $2^{-d_{\inf}}$ is small. Therefore, we propose to reduce this error by increasing b_1, i.e., the fractional bit width of each entry. We term this strategy **precision enhancement**. Fig. 5 (b) shows the contents of the LUT with $b_1 = 4$ and $b_2 = 2$. It can be seen that increasing b_1 not only enhances the precision of each entry in the LUT



(a) Naive LUT. $b_1 = 2$, $b_2 = 2$.

(b) LUT with precision enhancement. $b_1 = 4$, $b_2 = 2$.

(c) LUT with entry pruning based on (b). $b_1 = 4$, $b_2 = 1$.

(d) LUT with progressive precision reduction based on (c).

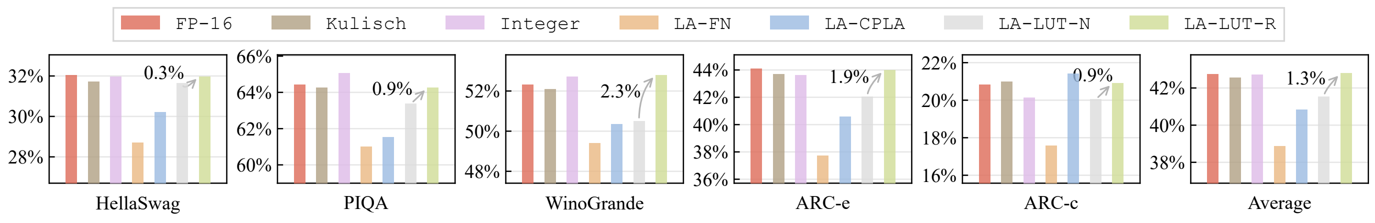
Fig. 5: Bit-level perspectives of the LUTs utilizing various optimization techniques.

but also increases the number of required entries due to the improved minimum resolution. In this case, the entries in the LUT are all zero when $d \geq 7.75$, the LUT index d requires 3 integer bits.

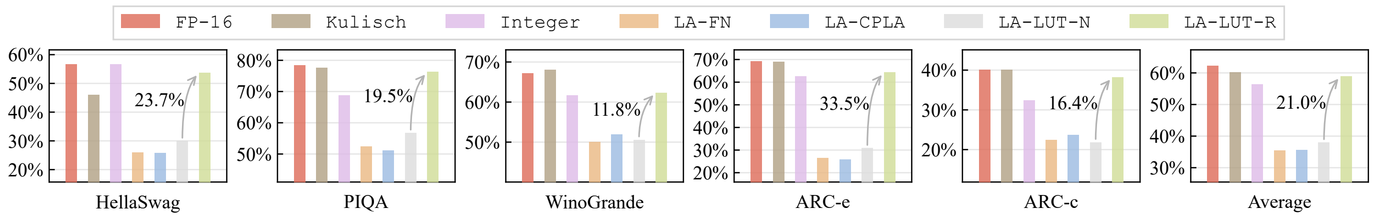
Additionally, it is noteworthy that the entries at higher LUT indices have smaller values, leading to consecutive zeros, as seen in the gray bits at the upper right corner in Fig. 5 (b). For the LUTs implemented using combinational logic, these zero bits consume minimal resources; hence, we ignore their impact on hardware overhead in our analysis.

We further simplify the hardware by lowering b_2 to reduce the number of the entries stored in the LUT, which we refer to as **entry pruning**. According to (23), lowering b_2 increases the error introduced by q_{b_2}. However, given that this error is scaled by the factor $2^{-d_{\inf}}$ and the dominant error has already been significantly mitigated by increasing b_1, the overall error is still reduced compared to the naive design. Fig. 5 (c) shows the contents of the LUT with $b_1 = 4$ and $b_2 = 1$. The index granularity changes from 2^{-2} to 2^{-1}, meaning that half of the entries are pruned in Fig. 5 (b).

By applying these two techniques, the hardware overhead of the LUT first increases and then decreases, ultimately leading to minimal overall variation; however, the error has indeed been



(a) OPT-350M



(b) LLAMA-2-7B

Fig. 6: Zero-shot accuracy of 5 common sense reasoning tasks using 7 computing schemes on 2 LLMs.

reduced. Specifically, given that $d_{\text{inf}} > 1$, increasing b_1 by 2 and decreasing b_2 by 1 result in a reduction of $U_{\mathbb{E}(\epsilon)}$, as defined in (23). Moreover, the error reduction becomes pronounced when d_{inf} is large, and this condition can be easily satisfied in continuous accumulation processes.

As shown in Fig. 4, in the accumulation of long sequences, the frequency of addition operations with smaller d is significantly lower than those with larger d. We argue that providing high precision for these less frequent cases is inefficient. Therefore, we propose to reduce the precision for the entries with small indices (d is small) in the LUT to further lower the hardware overhead, which is termed **progressive precision reduction**. Specifically, the first half of the LUT entries retain at most $b_1 - 1$ fractional bits, the first quarter retains at most $b_1 - 2$ fractional bits, and so on. Following this strategy, the gray bits at the bottom left of the LUT are truncated, as shown in Fig. 5 (d). This technique further simplifies the LUTs implemented based on combinational logic.

These three techniques all involve adjusting the entries of the LUT in LNS addition, which we collectively refer to as LUT refactoring. Precision enhancement reduces the primary error in (23), although it increases the hardware overhead. In contrast, entry pruning and progressive precision reduction relax the secondary error in (23) and the minor error in the initial stage of accumulation, respectively. However, they effectively reduce the hardware overhead.

V. Experiments

The LNS MAC unit employing the proposed LUT refactoring technique to implement the adder (referred to as `LA-LUT-R`), is modeled using Verilog and C++ to evaluate its hardware costs and accuracy in the context of 8-bit quantized LLMs.

Three approximate LNS adders are considered for comparison: `LA-FN`, based on Mitchell's algorithm and fractional normalization [11]; `LA-CPLA`, based on a compact piecewise linear approximation [13]; `LA-LUT-N`, based on the naive LUTs [5], [9]. For these LNS MAC units, the LNS formats $(1, b_i, b_f)$ are all set to $(1, 4, 3)$ for quantized inputs and $(1, 6, 5)$ for the intermediate accumulation results.

As references, two computing methods that introduce no error in accumulation are also considered: `Integer`, based on the $8/32$ MAC designed for integer quantization; `Kulisch`, based on the MAC unit performing accumulation by converting LNS products to their corresponding fixed-point representations [7], [8], [14], following the configurations in [8].

A. LLM Evaluation

To evaluate the accuracy of the considered MAC designs in LLMs, we replace the matrix multiplications (after 8-bit quantization) in all transformer blocks with the simulation code consistent with the hardware behavior, and then test the performance of the models with approximations on 5 common sense reasoning tasks. Two LLMs with different architectures and sizes are evaluated, including OPT-350M [1] and LLAMA-2-7B [2]. Tensor-wise quantization is adopted to ensure the efficiency of hardware deployment, i.e., all the elements within activation or a weight tensor share the same quantization scale factor.

Fig. 6 shows the accuracy of the two LLMs on 5 benchmarks using the aforementioned MAC designs. For `Integer`, we also tried to use SmoothQuant [17], which makes quantized LLMs perform better in token-wise and channel-wise quantization. However, SmoothQuant does not improve the accuracy for LLAMA-2-7B under 8-bit tensor-wise quantization.

For `LA-FN` and `LA-CPLA`, due to the high-level approximation, their accuracy is unacceptable in most of the tasks. Although `LA-LUT-N` exhibits low accuracy degradation compared to the original model (`FP-16`) on OPT-350M, it results in unacceptable performance in the larger model LLAMA-2-7B. In contrast, our design demonstrates a significant advantage over `LA-LUT-N`, offering an average accuracy improvement of 21.0% in LLAMA-2-7B. The accuracy of `LA-LUT-R` is comparable to that of the original model and `Kulisch`, which can be considered to have no accumulation error. Furthermore, our design outperforms `Integer`, benefiting from the reduced quantization error from logarithmic quantization and the reduced addition error from LUT refactoring.

To further assess the effectiveness of `LA-LUT-R`, the language modeling benchmark (perplexity) using the considered MAC designs are evaluated for OPT-125M, OPT-350M [1], LLAMA-2-7B [2] and MISTRAL-7B [3], as shown in Table I.

TABLE I: Comparison of the perplexity scores (lower is better) on WikiText-2 using 7 computing schemes.

Method	OPT-125M	OPT-350M	LLAMA-2-7B	MISTRAL-7B
`FP-16`	27.7	22.0	5.5	5.3
`Kulisch`	28.0	22.1	5.6	5.4
`Integer`	30.5	22.4	10.0	84.3
`LA-FN`	71.6	111.0	1,326,346.3	7,324.0
`LA-CPLA`	35.0	46.5	77,180.8	114,803.5
`LA-LUT-N`	30.1	24.3	192.9	25.6
`LA-LUT-R`	**28.7**	**22.7**	**6.2**	**5.9**

It can be seen that `LA-LUT-R` achieves the lowest perplexity compared to the other three MAC designs using approximate LNS adders. Although `Kulisch` with accurate accumulation has slightly lower perplexity, our design offers a greater advantage in hardware overhead. Compared to `Integer` method, our design exhibits comparable perplexity in small models (with 125M and 350M parameters); when considering large models (with 7B parameters), `LA-LUT-R` demonstrates a more pronounced advantage.

B. Hardware Evaluation

All the considered MAC designs are synthesized using Design Compiler based on a 28 nm CMOS technology to obtain their area and delay results. The power is measured using PrimeTime PX by the time-based analysis; the vectors randomly extracted from LLAMA-2-7B are used as inputs. All circuits are evaluated based on the same constraints with a 500 MHz clock and a 0.9 V supply voltage. The integer multiplier and adder are implemented directly by the IPs in DesignWare to ensure a fair comparison. The circuit measurement results of the considered MAC designs are shown Table II.

TABLE II: Circuit measurements of 6 MAC designs.

Method	Area	Power	Delay	ADP	PDP
	μm^2	μW	ns	μm^2·ns	fJ
`Kulisch`	386.2	363.1	1.524	588.5	553.3
`Integer`	355.2	327.4	1.161	412.5	380.2
`LA-FN`	220.9	168.0	1.145	252.9	192.4
`LA-CPLA`	311.2	262.0	1.568	488.1	410.9
`LA-LUT-N`	303.0	217.8	1.395	422.7	303.8
`LA-LUT-R`	**290.3**	**187.9**	**1.297**	**376.5**	**243.7**

It can be seen that the proposed `LA-LUT-R` shows much lower hardware overhead than the two accurate MAC designs, `Integer` and `Kulisch`. Although `LA-FN` consistently exhibits the lowest hardware overhead, it yields the worst performance in our accuracy evaluation experiments, making it unacceptable for practical applications. Compared to `LA-LUT-N`, our design achieves reductions of 10.93% in area-delay product (ADP) and 19.78% in power-delay product (PDP), while delivering average accuracy improvements of 1.3% and 21.0% in OPT-350M and LLAMA-2-7B, respectively. Specifically, compared to the commonly used `Integer`, which is designed for integer quantization, `LA-LUT-R` achieves an average accuracy improvement of 2.6% and a perplexity reduction of 4.8 in LLAMA-2-7B, while reducing area by 18.27% and power by 42.61%.

C. Ablation Study

The proposed LNS adder is enhanced by using three techniques, precision enhancement, entry pruning, and progressive precision reduction. To further analyze the impact of these techniques on the performance of LNS addition, ablation studies are conducted on each of the three technologies. Specifically, starting from a naive LNS adder, we incrementally add each technique and evaluate the perplexity of LLAMA-2-7B and the hardware cost of the MAC unit. The experimental results are presented in Table III.

TABLE III: The perplexity scores (*PPL*) and circuit measurements of the MAC units with the LNS adder applied different techniques. *PE*, *EP*, and *PPR* represent precision enhancement, entry pruning, and progressive precision reduction, respectively.

PE	*EP*	*PPR*	*PPL*	Area	Power	Delay
				μm^2	μW	ns
×	×	×	192.87	303.0	217.8	1.395
√	×	×	5.97	429.4	299.9	1.412
√	√	×	6.00	347.8	228.1	1.345
√	√	√	6.19	290.3	187.9	1.297

It can be observed that precision enhancement significantly decreases the perplexity, resulting in a very close performance to that of the original model. However, this improvement comes at the cost of increased hardware, due to the higher entry precision and increased number of entries in the LUTs. After applying entry pruning, the perplexity is increased by only 0.03, because only the secondary error in (23) is affected. At the same time, ADP and PDP are reduced by 22.89% and 27.58%, respectively, due to the pruning of the half LUT entries. Although progressive precision reduction leads to a slight increase (less than 0.2) in perplexity, it effectively simplifies the LUT, resulting in reductions of 19.48% and 20.54% in ADP and PDP, respectively.

VI. CONCLUSION

In this paper, we initially analyze the characteristics of the addition operations in LLMs. Consequently, based on the observation that the difference between the two addends is large in most cases, we mathematically derive the relationship between the hardware parameters and the error of LUT-based LNS addition. To simultaneously optimize the hardware and the accuracy, we propose LUT refactoring, involving three optimization techniques, i.e., precision enhancement, entry pruning, and progressive precision reduction. The experimental results show that, compared to existing approximate logarithmic adders, our design demonstrates significantly better performance in quantized LLMs. Compared to integer quantization, Applying LUT refactoring to the LUT-based LNS addition yields an improved average accuracy of 2.6% and a reduced perplexity of 4.8 in LLAMA-2-7B, with significant reductions in area (18.27%) and power (42.61%).

References

[1] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[3] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[4] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[5] E. E. Swartzlander and A. G. Alexopoulos, "The sign/logarithm number system," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1238–1242, 1975.

[6] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "LogNet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5900–5904.

[7] S. Vogel, M. Liang, A. Guntoro, W. Stechele, and G. Ascheid, "Efficient hardware acceleration of cnns using logarithmic data representation with arbitrary log-base," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[8] J. Johnson, "Rethinking floating point for deep learning," *arXiv preprint arXiv:1811.01721*, 2018.

[9] B. Parhami, "Computing with logarithmic number system arithmetic: Implementation methods and performance benefits," *Computers & Electrical Engineering*, vol. 87, p. 106800, 2020.

[10] I. Kouretas, C. Basetas, and V. Paliouras, "Low-power logarithmic number system addition/subtraction and their impact on digital filters," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2196–2209, 2012.

[11] G. Tsiaras and V. Paliouras, "Logarithmic number system addition-subtraction using fractional normalization," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[12] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[13] W. Zhang, X. Geng, Q. Wang, J. Han, and H. Jiang, "A low-power and high-accuracy approximate adder for logarithmic number system," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 125–131.

[14] Y. Wang, D. Deng, L. Liu, S. Wei, and S. Yin, "PL-NPU: An energy-efficient edge-device dnn training processor with posit-based logarithm-domain computing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 10, pp. 4042–4055, 2022.

[15] C.-N. Liu, Y.-A. Lai, C.-H. Kuo, and S.-A. Zhan, "Design of 2d systolic array accelerator for quantized convolutional neural networks," in *2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 2021, pp. 1–4.

[16] M. Tsagris, C. Beneki, and H. Hassani, "On the folded normal distribution," *Mathematics*, vol. 2, no. 1, pp. 12–28, 2014.

[17] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.