Rigorous Error Analysis for Logarithmic Number Systems

Thanh Son Nguyen
University of Utah
thahnson@cs.utah.edu

Alexey Solovyev
University of Utah
solovyev.alexey@gmail.com

Mark G. Arnold
Lehigh University
maab@lehigh.edu

Ganesh Gopalakrishnan
University of Utah
ganesh@cs.utah.edu

***Abstract*—Theorem proving demonstrates promising potential for verifying problems beyond the capabilities of SMT-solver-based verification tools. We explore and showcase the capability of Lean, an increasingly popular theorem-proving tool, in deriving the error bounds of table-based Logarithmic Number Systems (LNS). LNS reduces the number of bits needed to represent a high dynamic range of real numbers with finite precision and efficiently performs multiplication and division. However, in LNS, addition and subtraction become non-linear functions that must be approximated—typically using precomputed look-up tables. We provide the first rigorous analysis of LNS that covers first-order Taylor approximation, cotransformation techniques inspired by European Logarithmic Microprocessor, and the errors introduced by fixed-point arithmetic involved in LNS implementations. By analyzing all error sources and deriving symbolic error bounds for each, then accumulating these to obtain the final error bound, we prove the correctness of these bounds using Lean and its Mathlib library. We empirically validate our analysis using an exhaustive Python implementation, demonstrating that our analytical interpolation bounds are tight, and our analytical cotransformation bounds overestimate between one and two bits.**

***Index Terms*—Computer Arithmetic, Theorem Proving, Logarithmic Number Systems, Error Analysis**

I. Introduction

Spurred by the ever-growing need to reduce the burden of data movement while maintaining sufficient real-number representation precision, we are witnessing heightened interest in *numerics*: number system representations, their formal properties and verification methods. Mainstream verification techniques, primarily based on SMT-solvers, have shown limitations when confronted with complex mathematical problems involving non-linearity and multiple layers of quantifiers. Ironically, many of these challenges can be overcome by humans leveraging mathematical knowledge. Theorem proving emerges as a powerful extension to SMT-solver-based verification tools, enabling the application of mathematical theorems to enhance problem-solving capabilities. While theorem proving significantly reduces the tedium and errors associated with traditional human-led proof construction and checking, its main drawback lies in the requirement for human involvement. This research investigates the capabilities and convenience of Lean 4 in proving rigorous error bounds for the Logarithmic Number System (LNS) inspired by the European Logarithm Microprocessor (ELM). This number system requires approximation of non-linear functions to perform addition and subtraction operations, making error analysis significantly more challenging compared to Floating-Point (FP). Successfully deriving rigorous error bounds for the LNS using Lean 4, with a reasonable amount of human intervention, serves as a compelling demonstration of theorem proving's power in tackling intricate mathematical problems. This achievement not only showcases the potential of theorem proving in elaborate mathematical analysis but also hints at its broad applicability in verification across various fields, pushing the boundaries of what can be formally verified in computer arithmetic. Utilizing the Lean 4 theorem prover, this research investigates rigorous bounds for the implementation errors within the Logarithmic Number System (LNS) and evaluates Lean 4's efficacy for conducting rigorous formal error analysis in numerical computing.

A. Introduction of LNS

With the increasing costs of data movement in today's HPC and ML applications [1] [2], there is significant pressure to reduce the number of bits used to represent real numbers using finite-precision representations. With fewer bits moved, memory bandwidth as well as cache memories are better utilized. Also, given the sheer number of multiplications carried out by these applications (e.g., when performing matrix and tensor products), those number representations help reduce the cost of multiplications, divisions, and roots. Logarithmic number systems (LNS) possess both these advantages [3]. They store only the logarithm of real numbers in *finite-precision fixed-point representations*[1]. Furthermore, multiplications and divisions turn into fixed-point addition and fixed-point subtraction (respectively); and in the absence of overflows, introduce no additional error. Addition and subtraction are a whole different story, turning into calculations involving non-linear functions. This requires good trade-offs between numerical accuracy and computational speed. In more detail, methods are needed to perform additions and subtractions using various lookup tables whose overall cost must be minimized.[2] However, despite the availability of these correction methods, there are no

Supported in part by NSF Awards 2403379 and 2346394 and Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under award number DE-SC0024042

[1] In virtually all LNS implementations that are surveyed later.

[2] "Tableless methods" are also popular, approximating what table-based methods provide, but are not studied in this work.

rigorous error analysis methods available that tightly bound the worst-case error. Our work in this research closes this gap by providing such a rigorous error bound. All existing error estimation techniques that we know of perform error estimations through empirical testing. It is well-known that without tight error bounds, actual hardware/software designs most likely will end up over-provisioning to accommodate for these "excess errors" that never occur. The key contribution of this research is the first such tight and rigorous bound parameterized over Taylor interpolation and Coleman's co-transformation [4]. Specifically,

- We provide the first tight parametric error estimate formulae in terms of parameters such as the machine-epsilon of fixed-point numbers as well as look-up table sizes. Such a parameterization can help precisely guide hardware and software implementations.
- We formalize the error bound proof using the Lean 4 theorem prover and develop generic tactics and theorems that can be reused in similar work. We release this code in a repository `https://github.com/soarlab/RigorousErrorLNS` and refer the interested reader to specific Lean files for details of the proofs not covered in this paper.
- We demonstrate through systematic testing using a Python implementation of LNS that our estimates are trustworthy *and tight*. This Python code is available for reuse in other applications with `pip` install of "`xlns`" followed by "`import xlnsconf.utah_tayco_ufunc`". The open-source `xlns` package [5] allows fair comparison of our technique against many others such as NVIDIA's [6].

B. Comparison of LNS and IEEE Floating-Point

For those familiar with IEEE floating-point arithmetic [7], an IEEE *normal* floating-point number is described by a triple (s, e, m), where $s \in \{-1, 1\}$ denotes the sign, $e \in \mathbb{Z}$ is the exponent and $m \in [1, 2)$ is the mantissa (also known as the significand). The value of the real number represented by this triple is $s \cdot m \cdot 2^e$. Unlike floating-point, LNS does not have the mantissa but allows rational exponent instead. Also, while a *subnormal* IEEE floating-point number has $m < 1$ (in which case its value is encoded by the value in the mantissa weighted by the smallest normal exponent), LNS seldom has the notion of subnormals: the entire representable number scale is modeled in the same manner. Specifically, an LNS number is described by a pair (s, e), where $s \in \{-1, 1\}$ denotes the sign and $e \in \mathbb{Q}$ is the exponent. This pair represents the real value $s \cdot 2^e$. The exponent part of LNS is represented by a signed fixed-point number. LNS often represents zero with an extra bit or, alternatively, by using the most negative possible value of e as a special marker.

C. Rounding Modes for LNS

Two consecutive LNS fixed-point representations define an interval of real-numbered values, within which any real number can be rounded a variety of ways, such as up, down, faithfully (either up or down at implementor's discression), stochastic [8] (like faithful but unbiasedly random), to the nearest representable value (difficult with LNS), or Better Than Floating Point (BTFP; like nearest except faithful in difficult cases) [9]. The distance between two consecutive LNS fixed-point words (or half for rounding to nearest) establishes the *machine-epsilon* for LNS, denoted by ϵ, which determines the relative error bound for real values, $2^\epsilon - 1$. It is worth noting that some hardware designs may employ guard bits to improve computational accuracy [10]. In such cases, a series of arithmetic operations may be performed in extended precision before rounding back to the original precision. Because our work focuses on deriving the error bound for a single operation, in the remainder of this paper, we define the machine epsilon, ϵ, to be that associated with the fixed-point words (including any guard bits if used). This paper will assume *faithful rounding*, in which ϵ is the same as the weight of the least-significant bit of the LNS fixed-point representation.

D. Rigorous Problem Statement

We now introduce some basic notions underlying LNS that allow us to define the problem a bit more tightly. First, we describe how the four basic operations: addition, subtraction, multiplication and division are performed in LNS (operations such as square-root are not described, for brevity). In all four operations, the sign and magnitude of the result can be computed separately. Because computing the sign is straightforward, we assume that all the operands are positive. Let $2^p, 2^q$ be real numbers which can be exactly represented in LNS by the two fixed-point numbers p, q, then multiplication and division of 2^p and 2^q can be performed efficiently and exactly in LNS by fixed-point addition and subtraction:

$$\log_2(2^p \times 2^q) = \log_2 2^{p+q} = p + q$$

$$\log_2(2^p / 2^q) = \log_2 2^{p-q} = p - q.$$

However, the main drawback is that addition and subtraction are not directly realizable in LNS. Without loss of generality, let $p \geq q$ and let us use $x \leq 0$ to denote $q - p$ (this allows us to write 2^x, knowing that it will be a fraction in $(0, 1]$). Then,

$$\log_2(2^p + 2^q) = \log_2(2^p(1 + 2^x)) = p + \log_2(1 + 2^x)$$

$$\log_2(2^p - 2^q) = \log_2(2^p(1 - 2^x)) = p + \log_2(1 - 2^x).$$

Now let us introduce two non-linear functions Φ^+ and Φ^- (plotted in Figure 1, and also called *Gaussian Logarithms* [11]) defined as:

$$\Phi^+(x) = \log_2(1 + 2^x) \quad \text{and} \quad \Phi^-(x) = \log_2(1 - 2^x).$$

In Section III, we will show that Φ^+ and Φ^- will be approximated via ROM tables look-up and interpolation. Thus, to compute $\log_2(2^p + 2^q)$, we can simply perform a *fixed-point addition* of p to the result of $\Phi^+(x)$ (likewise for $\log_2(2^p - 2^q)$). In this work, we focus on deriving the bounds for the absolute-errors of the approximations of Φ^+ and Φ^-, for convenience, we call them *error bounds*.

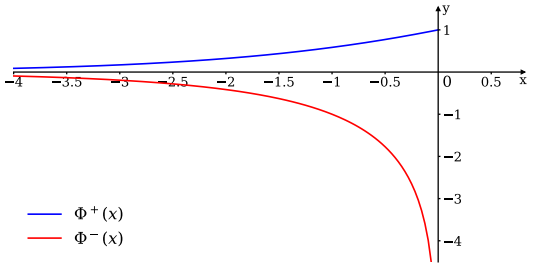

Fig. 1. Plots of Gaussian Logs, $\Phi^+(x)$ and $\Phi^-(x)$.

E. Rigorous error bound derivation

We derived the error bound of the Φ^+/Φ^- approximation for first-order Taylor approximation and the cotransformation technique proposed in European Logarithm Microprocessor [4]. To derive the rigorous error bounds, firstly, we analyze all error sources of each technique, which include the errors of the mathematical nature of the approximation method and the errors of finite-precision hardware implementation. Then, we mathematically derive the error bound of each error source by symbolic function analysis, and finally accumulate them using the triangle inequality for absolute values. Details of all error sources considered in our analysis and error bounds derivations for each source are presented in Sections III and IV.

F. Lean Proof

We construct a Lean 4 proof for the derived error bounds that leverages calculus theorems and tactics available in the Mathlib library, providing a solid foundation for our formal verification efforts. We also developed a set of lemmas and tactics which efficiently compute the symbolic derivatives of input functions and prove their monotonicity or antitonicity over specified domains, which not only facilitate the current proof but also can be reused in future verifications in similar domains. The resulting Lean proof, comprising approximately 2000 lines of code, is comparable in length to a fully detailed manually written proof.

II. Related Work

Proposed in the early 1970s, LNS is still a topic of current active interest. A bibliography [5] lists over 600 historic citations relevant to LNS research. The only previous attempt with LNS automated theorem proving was limited to direct lookup [12]. To date, no previous research has employed formal theorem proving to derive rigorous error bounds for LNS interpolation or cotransformation—a gap that this work addresses. We provide a brief, non-exhaustive survey of related work on LNS, and the current status of error analysis.

a) ***European Logarithm Microprocessor*** *[13]:* This first complete hardware realization of an LNS microprocessor uses base-2 logarithm, error-correction algorithms for the approximation of $\Phi^{\pm}(x)$, and also the cotransformation technique [14] for Φ^- when $x \to 0$. These authors evaluate area and delay, showing area equivalent to that of floating-point implementations with better delay. Although the authors compare the error of LNS with that of floating-point through empirical testing, there is no rigorous error analysis for LNS.

b) ***Convolution Neural Networks (CNN) Using Logarithmic Data Representations*** *[15]:* This approach, with logarithm bases 2 as well as $\sqrt{2}$, proposes the first CNN implemented in low-precision LNS and shows that LNS is superior to fixed-point arithmetic because of the non-uniform distribution of weights and activations. This hardware, as far as we know, has not been subject to rigorous analysis.

c) ***LogNet*** *[16]:* This approach, again with logarithm bases 2 and $\sqrt{2}$, focuses on improving the learning algorithm of CNNs using LNS, demonstrating the superiority of LNS over fixed-point schemes in terms of hardware costs, but rigorous error analysis has not been attempted.

d) ***LNS-MADAM*** *[6]:* An LNS-based implementation of weight updates in neural network training was recently proposed by NVIDIA where a hardware implementation is proposed. In this scheme, the logarithm base used is $2^{1/k}$. The highlight of their work is proposing a training algorithm for Deep Neural Networks (DNN) in low-precision LNS with an approximation of addition technique and the Multiplicative Weight Update (MWU) algorithm to replace Stochastic Gradient Descent. The authors propose their own LNS's design and hardware implementation, which is based on Mitchell's method and stochastic rounding [8]. The paper also performs symbolic error analysis to show that Multiplicative Weight Update is superior to Stochastic Gradient Descent in terms of minimizing the quantization error bound, assuming ideal LNS. However, the error bound does not provide insight to the accuracy of their LNS's design and implementation.

e) ***Dynamic LNS*** *[17]:* An FPGA-based implementation of dynamic LNS was studied for supporting large language models (LLM). This provides sufficient quantization for numbers closer to zero while also handling long-tail distribution of LLM outliers. Their work does not provide any error analysis.

III. Rigorous error bound via First-Order Taylor approximation

a) Notation: The computation of LNS addition and subtraction involves fixed-point addition/subtraction and interpolations with tables that discretize the Φ^+ and Φ^- functions, suitably limiting the number of precomputed and stored table values. Let Φ generically stand for either Φ^+ or Φ^-.

When it is necessary to make it clear that we are indexing tables, we will use the notation Φ_T. We will sample Φ at a spacing of Δ and store these values in Φ_T. Now, given an arbitrary x, define $\mathbf{i}$ as $\left\lceil \frac{x}{\Delta} \right\rceil \Delta$ and $\mathbf{r}$ as $\mathbf{i} - x$. Recall that x is negative. Thus, $\mathbf{i}$ is the discrete index *after* x and we have $x \le \mathbf{i} \le 0$, $0 \le \mathbf{r} < \Delta$, and $x = \mathbf{i} - \mathbf{r}$. It must be clear that $\Phi_T(\mathbf{i}) = \Phi(\mathbf{i})$ and $\Phi_T(x)$ (viewed as a tabular function) is undefined at other x than at these $\mathbf{i}$. For x not in Φ_T, we can apply interpolation techniques [9], [13], [18].

b) First-order Taylor approximation: We can now define an approximation to $\Phi(x)$ defined with the help of two tables Φ_T and $\Phi_T^{'}$ that are defined at the $\mathbf{i}$ points:

$$\hat{\Phi}_T(x) = \Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i}) = \Phi_T(\mathbf{i}) - \mathbf{r}\Phi'_T(\mathbf{i}) \qquad (1)$$

Equation 1 shows how to calculate Φ^+ and Φ^- using the first-order Taylor approximation, the error of which is illustrated in Figure 2. However, because LNS is implemented in hardware

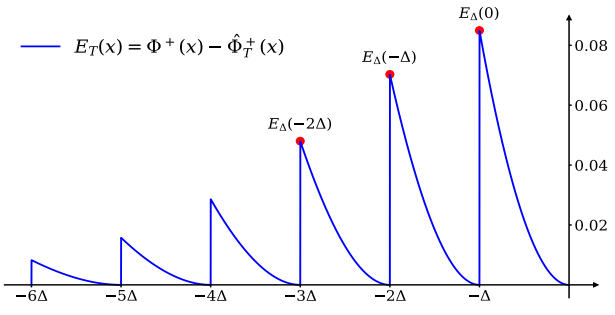

Fig. 2. The error of the first-order Taylor approximation of $\Phi^+(x)$.

using fixed-point numbers, there are two more sources of error due to the fact that the look-up tables' values of $\Phi(\mathbf{i})$ and $\Phi'(\mathbf{i})$ must be rounded to the LNS fixed-point representation, and the multiplication $\mathbf{r}\Phi'(\mathbf{i})$ is performed in fixed-point arithmetic (with rounding). Taking into account the implementation using fixed-point, we refine the first-order Taylor approximation presented in Equation 1 as:

$$\widetilde{\Phi}_T(x) = \overline{\Phi}(\mathbf{i}) - \text{rnd}(\mathbf{r}\overline{\Phi'}(\mathbf{i})) \tag{2}$$

where $\overline{\Phi}$ and $\overline{\Phi'}$ are the fixed-point rounded look-up tables for Φ and Φ'.

We define the notations for the three sources of error:

- Interp-err: the mathematical error of interpolating Φ^+ and Φ^- via first-order Taylor approximation, which is $|\Phi(x) - \hat{\Phi}_T(x)|$.
- Tab-err: the rounding error of the precomputed values in the look-up tables, which are $|\Phi(\mathbf{i}) - \overline{\Phi}(\mathbf{i})|$ and $|\Phi'(\mathbf{i}) - \overline{\Phi'}(\mathbf{i})|$.
- Mul-err: the error of fixed-point arithmetic multiplication, which is $|\mathbf{r}\overline{\Phi'}(\mathbf{i}) - \text{rnd}(\mathbf{r}\overline{\Phi'}(\mathbf{i}))|$.

The error of interpolating Φ using first-order Taylor approximation is $|\Phi(x) - \widetilde{\Phi}_T(x)|$, where $\widetilde{\Phi}_T(x)$ is defined in Equation 2. The rigorous error bound is derived as follows.

c) Error bound of Interp-err derivation: Lemma 1 and Lemma 2 derive the error bound of the Interp-err for Φ^+ and Φ^-, without considering Tab-err and Mul-err. Intuitively, from the shape of Interp-err of Φ^+ (illustrated in Figure 2), we observe that:

- For each Δ-segment, the error is greater when x is further away from 0.
- The further a Δ-segment is away from 0, the smaller its supremum error is.

Lemma 1 formally proves these observations by analyzing the error function $|\Phi^+(x) - \hat{\Phi}_T^+(x)|$, concluding that the supremum of the error over the whole domain $x < 0$ is reached as $x \to -\Delta$. This supremum is denoted by $E_\Delta^+(0)$. Similarly, Lemma 2 proves that the error bound of Φ^- is achieved as $x \to -1 - \Delta$, denoted by $E_\Delta^-(-1)$.

d) Total error bound derivation: The error bound of Tab-err and Mul-err is simply a constant ϵ, which is the maximum absolute rounding error of the fixed-point representation of LNS. The total error bound is derived in Theorem 1 by accumulating (using the triangle inequality for absolute values) the error bound of Interp-err with that of Tab-err and Mul-err.

Lemma 1 derives the error bound of Interp-err of Φ^+ in the range $(-\infty, 0]$. Recall that the Interp-err is $|\Phi^+(x) - \hat{\Phi}_T^+(x)|$ and $E_\Delta(\mathbf{i}) = \Phi(\mathbf{i} - \Delta) - \Phi(\mathbf{i}) + \Delta\Phi'(\mathbf{i})$.

Lemma 1. *For all* $x \in (-\infty, 0]$,

$$|\Phi^+(x) - \hat{\Phi}_T^+(x)| \le E_\Delta^+(0) = \frac{\ln 2}{8}\Delta^2 + O(\Delta^4).$$

Proof. Let $E(\mathbf{i}, \mathbf{r}) = \Phi^+(x) - \hat{\Phi}_T^+(x)$. From the definition of $\hat{\Phi}_T^+(x)$ in Equation 1:

$$E(\mathbf{i}, \mathbf{r}) = \Phi^+(\mathbf{i} - \mathbf{r}) - (\Phi^+(\mathbf{i}) - \mathbf{r}(\Phi^+)'(\mathbf{i})).$$

A novel approach which has not been used in LNS analysis before is to note that despite the definition: $\mathbf{i} = \left\lceil \frac{x}{\Delta} \right\rceil \Delta$, it is safe to consider the domain of $\mathbf{i}$ to be $\mathbb{R}_{\le 0}$ instead of $\Delta\mathbb{Z}_{\le 0}$ when deriving error bound, because:

$$\max_{\mathbf{i} \in \Delta\mathbb{Z}_{\le 0}, 0 \le \mathbf{r} < \Delta} |E(\mathbf{i}, \mathbf{r})| \le \max_{\mathbf{i} \in \mathbb{R}_{\le 0}, 0 \le \mathbf{r} < \Delta} |E(\mathbf{i}, \mathbf{r})|.$$

This allows to prove the lemma using calculus techniques alone, without induction otherwise needed for a discrete $\mathbf{i}$.

The lemma is proved by:

1) Proving that $\forall \mathbf{i} \le 0, 0 \le \mathbf{r} < \Delta : E(\mathbf{i}, \mathbf{r}) \ge 0$, so $E(\mathbf{i}, \mathbf{r}) = |E(\mathbf{i}, \mathbf{r})| = |\Phi^+(x) - \hat{\Phi}_T^+(x)|$.
2) Proving that both partial derivatives of E w.r.t $\mathbf{r}$ and $\mathbf{i}$ are non-negative, so E approaches its supremum when $\mathbf{i} = 0$ and $\mathbf{r} \to \Delta$. Formally, we have to prove that $\forall \mathbf{i} \le 0, 0 \le \mathbf{r} < \Delta : \frac{\partial E}{\partial \mathbf{r}}(\mathbf{i}, 0) \ge 0$ and $\frac{\partial E}{\partial \mathbf{i}}(\mathbf{i}, 0) \ge 0$, so $\max_{\mathbf{i} \in \mathbb{R}_{\le 0}, 0 \le \mathbf{r} < \Delta} E(\mathbf{i}, \mathbf{r}) \le E(0, \Delta) = E_\Delta^+(0)$.

Take the first and second derivatives of $E(\mathbf{i}, \mathbf{r})$ w.r.t $\mathbf{r}$:

$$\frac{\partial E}{\partial \mathbf{r}}(\mathbf{i}, \mathbf{r}) = \frac{2^{\mathbf{i}}}{2^{\mathbf{i}} + 1} - \frac{2^{\mathbf{i}-\mathbf{r}}}{2^{\mathbf{i}-\mathbf{r}} + 1},$$
$$\frac{\partial^2 E}{\partial \mathbf{r}^2}(\mathbf{i}, \mathbf{r}) = \frac{2^{\mathbf{i}-\mathbf{r}} \ln 2}{(2^{\mathbf{i}-\mathbf{r}} + 1)^2}.$$

From $\frac{\partial^2 E}{\partial \mathbf{r}^2}(\mathbf{i}, \mathbf{r}) > 0$ and $\frac{\partial E}{\partial \mathbf{r}}(\mathbf{i}, 0) = 0$, we conclude that $\forall \mathbf{i} \le 0, 0 \le \mathbf{r} < \Delta : \frac{\partial E}{\partial \mathbf{r}}(\mathbf{i}, \mathbf{r}) \ge 0$.
Then, because $E(\mathbf{i}, 0) = 0$, we conclude that $\forall \mathbf{i} \le 0, 0 \le \mathbf{r} < \Delta : E(\mathbf{i}, \mathbf{r}) \ge 0$.

We complete the proof by proving that $\forall \mathbf{i} \le 0, 0 \le \mathbf{r} < \Delta : \frac{\partial E}{\partial \mathbf{i}}(\mathbf{i}, \mathbf{r}) \ge 0$. Let $a = \mathbf{r} \ln 2$, then

$$\frac{\partial E}{\partial \mathbf{i}}(\mathbf{i}, \mathbf{r}) = \frac{2^{\mathbf{i}}}{(2^{\mathbf{i}} + 1)^2 (2^{\mathbf{i}-\mathbf{r}} + 1)} \left(2^{\mathbf{i}} f(a) + g(a)\right)$$

with $f(a) = ae^{-a} + e^{-a} - 1$ and $g(a) = e^{-a} + a - 1$.

Since $\frac{2^{\mathbf{i}}}{(2^{\mathbf{i}}+1)^2(2^{\mathbf{i}-\mathbf{r}}+1)} > 0$, the sign of $\frac{\partial E}{\partial \mathbf{i}}(\mathbf{i}, \mathbf{r})$ is the same as that of $N(\mathbf{i}) = 2^{\mathbf{i}} f(a) + g(a)$.

Because $a \ge 0$, from $f(0) = 0$ and $f'(a) = -ae^{-a} \le 0$, we conclude that $f(a) \le 0$, so

$$N'(\mathbf{i}) = 2^{\mathbf{i}} (\ln 2) f(a) \le 0.$$

Let $h(a) = N(0) = (a+2)e^{-a} + a - 2$, then $h(0) = 0$ and $h'(a) = -f(a) \geq 0$, we conclude that: $h(a) = N(0) \geq 0$.

From $N(0) \geq 0$ and $N'(\mathbf{i}) \leq 0$, we conclude that for all $\mathbf{i} \leq 0$, $N(\mathbf{i}) \geq N(0) \geq 0$. Hence, $\forall \mathbf{i} \leq 0,\ 0 \leq \mathbf{r} < \Delta$: $\frac{\partial E}{\partial \mathbf{i}}(\mathbf{i}, \mathbf{r}) \geq 0$. □

Lean 4 proof: Lemma 1 corresponds to the `Ep_bound` lemma in the file `BasicErrTaylor.lean`. The proof primarily employs the following techniques:

1) Applying Mathlib's `strictMonoOn_of_deriv_pos` theorem to establish the strict monotonicity of E.
2) Using the custom `get_deriv` tactic to compute the derivatives.
3) Using the `positivity` tactic to establish the positivity of the expressions.
4) Utilizing the `fun_prop` tactic to verify the continuity and differentiability of the functions.

Lemma 2 derives the error bound of Interp-err of Φ^- in the range $(-\infty, -1]$. Note that for the range $(-1, 0)$, Φ^- is calculated by the cotransformation technique, the error bound of which is derived in Section IV.

Lemma 2. *Suppose that $\frac{1}{\Delta} \in \mathbb{N}$ (e.g., $\Delta = 2^{-k}$ for some natural number k). Then for all $x \in (-\infty, -1]$,*

$$|\Phi^-(x) - \hat{\Phi}_T^-(x)| \leq -E_\Delta^-(-1) = (\ln 2)\Delta^2 + O(\Delta^3).$$

Proof. Define $E(\mathbf{i}, \mathbf{r}) = -\big(\Phi^-(\mathbf{i}-\mathbf{r}) - (\Phi^-(\mathbf{i}) - \mathbf{r}(\Phi^-)'(\mathbf{i}))\big)$. Using the same proof techniques as in the proof of Lemma 1, we can show that for all $\mathbf{i} < 0$ and $0 \leq \mathbf{r} < \Delta$, partial derivatives of E w.r.t. to $\mathbf{i}$ and $\mathbf{r}$ are non-negative. It follows that

$$|E(\mathbf{i}, \mathbf{r})| = E(\mathbf{i}, \mathbf{r}) \leq E(\mathbf{i}^*, \Delta)$$

for all $\mathbf{i} \leq \mathbf{i}^* < 0$ and $0 \leq \mathbf{r} < \Delta$.

To prove the lemma, we need to take $\mathbf{i}^*$ corresponding to the largest value of $x \leq -1$. Since $\frac{1}{\Delta} \in \mathbb{N}$, we obtain $\mathbf{i}^* = \left\lceil \frac{-1}{\Delta} \right\rceil \Delta = -1$ and the error bound is $E(-1, \Delta) = -E_\Delta^-(-1)$. □

Lean 4 Proof: Lemma 2 corresponds to the `Em_bound` lemma in the file `BasicErrTaylor.lean` and its formal proof is similar to the formal proof of Lemma 1.

Theorem 1 derives the total error bound of computing Φ^+ and Φ^- using first-order Taylor approximation.

Theorem 1. *Let ϵ be the machine-epsilon of the fixed-point representation of the LNS under consideration. Let $E_M^+ = E_\Delta^+(0)$, and $E_M^- = -E_\Delta^-(-1)$. Then*

$$|\Phi(x) - \tilde{\Phi}_T(x)| < E_M + (2 + \Delta)\epsilon.$$

Proof. We have the following inequality

$$\begin{aligned} |\Phi(x) - \tilde{\Phi}_T(x)| &\leq |\Phi(x) - \hat{\Phi}_T(x)| + |\hat{\Phi}_T(x) - \tilde{\Phi}_T(x)| \\ &\leq E_M + |\hat{\Phi}_T(x) - \tilde{\Phi}_T(x)|. \end{aligned}$$

From Equation 1 and Equation 2, $\hat{\Phi}_T(x) - \tilde{\Phi}_T(x)$ can be re-written as $a_1 + a_2 + a_3$, where

$$a_1 = \Phi(\mathbf{i}) - \overline{\Phi}(\mathbf{i})$$

$$a_2 = \mathbf{r}(\Phi'(\mathbf{i}) - \overline{\Phi'}(\mathbf{i}))$$

$$a_3 = \mathbf{r}\overline{\Phi'}(\mathbf{i}) - \text{rnd}(\mathbf{r}\overline{\Phi'}(\mathbf{i})).$$

Apply the triangle inequality for absolute values again:

$$|\hat{\Phi}_T(x) - \tilde{\Phi}_T(x)| \leq |a_1| + |a_2| + |a_3|.$$

From $|a_1|, |a_3|, |\Phi'(\mathbf{i}) - \overline{\Phi'}(\mathbf{i})| \leq \epsilon$ (as they are errors of fixed-point rounding) and $0 \leq \mathbf{r} < \Delta$:

$$|\hat{\Phi}_T(x) - \tilde{\Phi}_T(x)| < (2 + \Delta)\epsilon.$$

Hence, the final error bound of first-order Taylor approximation is

$$|\Phi(x) - \tilde{\Phi}_T(x)| < E_M + (2 + \Delta)\epsilon.$$

□

Lean 4 proof: The proof of Theorem 1 is stored in the file `ErrTaylor.lean`. The proof involves the following steps:

1) Defining the error bound of fixed-point rounding as an axiom (defined in the structure `FixedPoint`):

$$\texttt{hrnd} : \forall x,\ |x - \texttt{rnd}(x)| \leq \varepsilon.$$

2) Using the `abs_add` theorem to prove the total rounding error.

IV. Rigorous error bound via Cotransformation technique

One of the more difficult cases of error control in LNS is when computing $\Phi^-(x)$ for values of x close to 0. The trouble arises because of the nature of this function: the nth derivative of $\Phi^-(x)$ for all n tends to $-\infty$ as x approaches 0, i.e. $\Phi^-(x)$ has a singularity at 0 (see Figure 1). To avoid computing $\Phi^-(x)$ in this range, addition/subtraction can be split across different intervals and computed by the so-called *cotransformation* techniques [14], [19]–[22]. Cotransformation has the advantage of using less memory than other accurate alternatives [9], [18], [23]. A cotransformation technique is proposed by European Logarithmic Microprocessor [14], which suggests 3-way interval split defined by design-specific constants Δ_a and Δ_b. The general idea is to maintain three extra look-up tables T_a, T_b and T_c of Φ^- inside the range $(-1, 0)$, then transform $\Phi^-(x)$ such that it can be computed by indexing those look-up tables together with interpolating Φ^- outside of the range $(-1, 0)$.

Let Δ_a and Δ_b be two positive fixed-point numbers such that Δ_a is very close to 0, Δ_b is a multiple of Δ_a and $\frac{1}{\Delta_b} \in \mathbb{N}$:

- The table T_a covers all fixed-point numbers in a very small range $[-\Delta_a, 0)$,
- The table T_b covers all multiples of Δ_a in the range $[-\Delta_b, -\Delta_a)$.
- The table T_c covers all multiples of Δ_b in the range $x \in (-1, -\Delta_b)$.

Let $x \in (-1, 0)$. Compute $\Phi^-(x)$ as follows:

Case 1: $x \in [-\Delta_a, 0)$. $\Phi^-(x)$ is indexed directly from table T_a.

Case 2: $x \in [-\Delta_b, -\Delta_a)$.

Let $r_b = \left(\left\lceil \frac{x}{\Delta_a} \right\rceil - 1\right)\Delta_a$ (i.e. r_b is the index value of T_b which is smaller than and closest to x) and $r_a = r_b - x$. Let $k = x - \Phi^-(r_b) + \Phi^-(r_a)$. It can be shown that $k \leq -1$ (Lemma 4) and

$$\Phi^-(x) = \Phi^-(r_b) + \Phi^-(k). \tag{3}$$

$\Phi^-(r_a)$ and $\Phi^-(r_b)$ are indexed directly from tables T_a (since $-\Delta_a \leq r_a < 0$) and T_b respectively, and $\Phi^-(k)$ is computed with first-order Taylor approximation for $k \in (-\infty, -1]$.

Case 3: $x \in (-1, -\Delta_b)$.

Let $r_c = \left(\left\lceil \frac{x}{\Delta_b} \right\rceil - 1\right)\Delta_b$, (i.e. r_c is the index value of T_c which is smaller than and closest to x)

and $r_{ab} = r_c - x$,

and $r_b = \left(\left\lceil \frac{r_{ab}}{\Delta_a} \right\rceil - 1\right)\Delta_a$, (i.e. r_b is the index value of T_b which is smaller than and closest to r_{ab})

and $r_a = r_b - r_{ab}$,

and $k_1 = r_{ab} - \Phi^-(r_b) + \Phi^-(r_a)$,

and $k_2 = x - \Phi^-(r_c) + \Phi^-(r_{ab}) = x - \Phi^-(r_c) + \Phi^-(r_b) + \Phi^-(k_1)$.

Then $k_1 \leq -1$, [3] $k_2 \leq -1$ and

$$\Phi^-(x) = \Phi^-(r_c) + \Phi^-(k_2) \tag{4}$$

where $\Phi^-(r_a)$, $\Phi^-(r_b)$, and $\Phi^-(r_c)$ are indexed directly from tables T_a, T_b, and T_c respectively, and $\Phi^-(k_1)$ and $\Phi^-(k_2)$ are computed with first-order Taylor approximation.

Now we derive and prove the error bound of the cotransformation technique described above.

Lemma 3 derives the error bound of computing $\Phi^-(x)$ given the error bound of x (assuming that computing $\Phi^-(x)$ is error-free).

Lemma 3. *For all $x, x^* \leq -1$ and $|x - x^*| \leq m$*

$$|\Phi^-(x) - \Phi^-(x^*)| \leq \Phi^-(-1-m) - \Phi^-(-1).$$

Proof. Note that both the first and second derivatives of Φ^- are always negative:

$$\forall x, (\Phi^-)'(x) = \frac{-2^x}{1-2^x} < 0 \text{ and } (\Phi^-)''(x) = \frac{-2^x \ln 2}{(1-2^x)^2} < 0.$$

Without loss of generality, suppose that $x \geq x^*$, then $|\Phi^-(x) - \Phi^-(x^*)| = \Phi^-(x^*) - \Phi^-(x)$.

Let $t = x - x^*$ and $F(x,t) = \Phi^-(x-t) - \Phi^-(x)$, then $0 \leq t \leq m$ and

$$F(x,t) = |\Phi^-(x) - \Phi^-(x^*)| \leq \max_{0\leq t\leq m, x\leq -1} F(x,t).$$

We will prove that $\max_{0\leq t\leq m, x\leq -1} F(x,t) = F(-1,m)$, which is true if both partial derivatives of F are non-negative. Since $x \geq x - t$ and $(\Phi^-)''(x) < 0$:

$$\frac{\partial F}{\partial x}(x,t) = (\Phi^-)'(x-t) - (\Phi^-)'(x) \geq 0.$$

Since $(\Phi^-)'(x) < 0$:

$$\frac{\partial F}{\partial t}(x,t) = -(\Phi^-)'(x-t) \geq 0$$

Therefore, $\max_{0\leq t\leq m, x\leq -1} F(x,t) = F(-1,m)$. □

Lemma 4 establishes a bound of k (and also k_1, k_2).

Lemma 4. *Suppose that $x \leq -d < 0$. Let $i = \left(\left\lceil \frac{x}{d} \right\rceil - 1\right)d$ and $r = i - x$. Then*

$$x - \Phi^-(i) + \Phi^-(r) \leq -d - \Phi^+(-d) \leq -1 - \frac{d}{2}.$$

Proof. Let $f(x) = x - \Phi^-(x)$. We have $x = i - r$. Rewrite the left hand side of the goal as follows:

$$i - r - \Phi^-(i) + \Phi^-(r) = f(i) - f(r).$$

We can prove that $f(x)$ is strictly increasing (for $x < 0$) by taking its derivative and showing that it is positive. It is easy to prove that $i \leq -2d$ (since $x \leq -d$) and $-d \leq r$. Hence

$$f(i) - f(r) \leq f(-2d) - f(-d) = -d - \big(\Phi^-(-2d) - \Phi^-(-d)\big).$$

It is not difficult to show that $\Phi^-(-2d) - \Phi^-(-d) = \Phi^+(-d)$. The inequality $-d - \Phi^+(-d) \leq -1 - \frac{d}{2}$ can be shown by moving everything to the right hand side and showing that the derivative of the right hand side is non-negative. □

Lean 4 Proofs: Lemma 3 and Lemma 4 correspond to lemmas `cotrans_lemma` and `k_bound` in the file `BasicCotrans.lean`. All proof steps use similar proof-construction techniques as used in previous lemmas and theorems.

Theorem 2 performs step-by-step error bound derivation for each of the three cases of the cotransformation technique's computation. The total error bound is the maximum of three error bounds.

Theorem 2. *Let ϵ be the machine-epsilon of the fixed-point representation of the LNS under consideration and E_{Φ^-} be the error bound of interpolating of Φ^- in the range $(-\infty, 1]$. Assume also that $\Delta_a \geq 4\epsilon$ and $\Delta_b \geq 8\epsilon + 2E_{\Phi^-}$. The error bound of computing $\Phi^-(x)$ when $x \in (-1, 0)$ using the cotransformation technique is:*

$$\Phi^-(-1 - E_{k_2}) - \Phi^-(-1) + E_{\Phi^-} + \epsilon$$

where

$$E_{k_2} = \Phi^-(-1 - 2\epsilon) - \Phi^-(-1) + E_{\Phi^-} + 2\epsilon.$$

Proof. **Case 1:** $x \in [-\Delta_a, 0)$. $\Phi^-(x)$ is indexed directly from T_a, so the error bound is ϵ.

Case 2: $x \in [-\Delta_b, -\Delta_a)$. r_b is error-free because it is an integer multiple of a fixed-point number Δ_a. $r_a = r_b - x$ is error-free as a difference of two fixed-point numbers. We derive the error bound of computing k, $\Phi^-(k)$, and $\Phi^-(x)$. Let $\tilde{k}$ and $\tilde{\Phi}^-$ be the results of the computations of k and Φ^- with fixed-point rounding and interpolation errors. From the formula $k = x - \Phi^-(r_b) + \Phi^-(r_a)$, the error of computing k consists of the two fixed-point rounding errors of $\Phi^-(r_a)$

[3] If $r_{ab} \in [-\Delta_a, 0)$, it may happen that $k_1 > -1$. But $r_{ab} \geq -\Delta_a$ implies that the computations from Case 2 can be applied to this case. More specifically, replace r_b and r_a (in Case 2) with r_c and r_{ab} (from Case 3) and follow the computation steps of Case 2: $\Phi^-(r_c)$ is indexed from T_c and $\Phi^-(r_{ab})$ is indexed from T_a since $r_{ab} \in [-\Delta_a, 0)$.

and $\Phi^-(r_b)$, so its error bound is 2ϵ, (i.e. $|k - \tilde{k}| \leq 2\epsilon$). From the error bound of computing k and Lemma 3, we get $|\Phi^-(k) - \Phi^-(\tilde{k})| \leq \Phi^-(-1-2\epsilon) - \Phi^-(-1)$. Next, because $\tilde{\Phi}^-(\tilde{k})$ is computed by interpolation, $|\Phi^-(\tilde{k}) - \tilde{\Phi}^-(\tilde{k})| \leq E_{\Phi^-}$, we derive the error bound of computing $\Phi^-(k)$:

$$\begin{aligned}|\Phi^-(k) - \tilde{\Phi}^-(\tilde{k})| &\leq |\Phi^-(k) - \Phi^-(\tilde{k})| + |\Phi^-(\tilde{k}) - \tilde{\Phi}^-(\tilde{k})| \\ &\leq \Phi^-(-1-2\epsilon) - \Phi^-(-1) + E_{\Phi^-}.\end{aligned}$$

We also need to show that $\tilde{k} \leq -1$ (because the interpolation error E_{Φ^-} is valid in $(-\infty, -1]$ only). Using the bound from Lemma 4 with $d = \Delta_a$ and the error bound of $\tilde{k}$, we get that $\Delta_a \geq 4\epsilon$ implies $\tilde{k} \leq -1$.

Finally, from Equation 3, we accumulate the error bound of computing $\Phi^-(k)$ with the fixed-point rounding error bound of $\Phi^-(r_b)$, which is ϵ, to get the error bound of computing $\Phi^-(x)$ in Case 2:

$$\Phi^-(-1-2\epsilon) - \Phi^-(-1) + E_{\Phi^-} + \epsilon.$$

Case 3: $x \in (-1, -\Delta_b)$. Similar to Case 2, all the terms r_c, r_{ab}, r_b and r_a are error-free. We derive the error bound of computing k_1, $\Phi^-(k_1)$, k_2, $\Phi^-(k_2)$, and finally $\Phi^-(x)$. Let $\tilde{k_1}$, $\tilde{k_2}$ and $\tilde{\Phi}^-$ be the actual results of computation of k_1, k_2 and Φ^-. Because of the similarity in the formula of k_1 and that of k in Case 2, the error bound of computing k_1 and $\Phi^-(k_1)$ is the same as that of computing k and $\Phi^-(k)$ in Case 2, which are 2ϵ and $\Phi^-(-1-2\epsilon) - \Phi^-(-1) + E_{\Phi^-}$. From the formula $k_2 = x + \Phi^-(r_b) + \Phi^-(k_1) - \Phi^-(r_c)$, we derive the error bound of computing k_2 by accumulating the error bound of computing $\Phi^-(k_1)$ with the fixed-point rounding error bounds of $\Phi^-(r_b)$ and $\Phi^-(r_c)$ (both are ϵ). Let E_{k_2} be the error bound of computing k_2, then

$$E_{k_2} = \Phi^-(-1-2\epsilon) - \Phi^-(-1) + E_{\Phi^-} + 2\epsilon.$$

Similar to how we derive the error bound of computing $\Phi^-(k_1)$, the error bound of computing $\Phi^-(k_2)$ is:

$$|\Phi^-(k_2) - \tilde{\Phi}^-(\tilde{k_2})| \leq \Phi^-(-1 - E_{k_2}) - \Phi^-(-1) + E_{\Phi^-}.$$

We can also show that $\Delta_b \geq 8\epsilon + 2E_{\Phi^-}$ implies $\tilde{k_2} \leq -1$ (the proof is similar to the corresponding inequality for $\tilde{k}$ in Case 2). Finally, from Equation 4, we accumulate the error bound of computing $\Phi^-(k_2)$ with the fixed-point rounding error bound of $\Phi^-(r_c)$, which is ϵ, to get the error bound of computing $\Phi^-(x)$ in Case 3:

$$\Phi^-(-1 - E_{k_2}) - \Phi^-(-1) + E_{\Phi^-} + \epsilon.$$

The error bound of computing $\Phi^-(x)$ when $x \in (-1, 0)$ with cotransformation technique is the maximum of the error bounds of Cases 1, 2, and 3, which is the error bound of Case 3 (it follows from the fact that Φ^- is decreasing and $-E_{k_2} \leq -2\epsilon$). □

Lean 4 Proof: The proof of Theorem 2 is located in the file `Cotransformation.lean`, with the error bounds for Case 2 and Case 3 established in lemmas `bound_case2` and `bound_case3`.

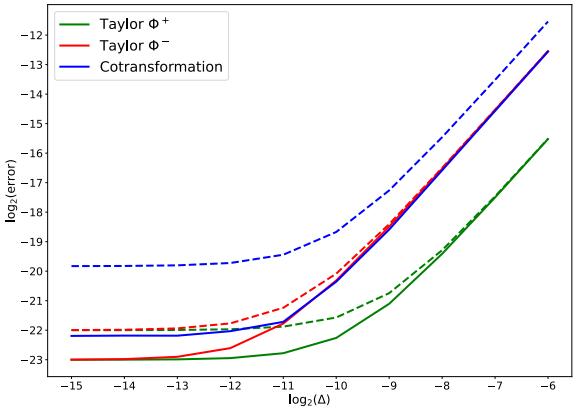

Fig. 3. Plots of actual worst-case errors for $\epsilon = 2^{-23}$ obtained via exhaustive testing (solid lines) and theoretical error bounds (dashed lines). Cotransformation errors are computed with $\Delta_a = 2^{-20}$ and $\Delta_b = 2^{-10}$.

V. Numerical Validation of Error Bounds

Fig. 3 shows a log-log figure of the actual errors (solid lines) observed in exhaustive simulation ($\epsilon = 2^{-23}$, similar to IEEE-754 single FP) versus our error bounds (dashed lines) for the three methods analyzed here plotted against $\log_2(\Delta)$. All the log-log plots are "hockey sticks" (left flat line; middle transition curve; and right upward diagonal). Except in the transition, these are straight lines. Until Δ becomes small enough that ϵ quantization plays a role, $\log_2(\text{error})$ is proportional to $2\log_2(\Delta) = \log_2(\Delta^2)$ plus a constant, making the "hockey sticks" mostly parallel to each other, consistent with the well-known quadratic error from linear interpolation [24], [25]. The constant is the same for Φ^- interpolation and cotransformation because Coleman's cotransformation method exclusively uses Φ^- interpolation in its implementation. On the other hand, Φ^+ interpolation (both actual and bounds) is two bits more accurate than Φ^- interpolation for a given Δ.

The designer of an LNS circuit typically wants to achieve below a certain error using minimum memory. Since Δ is inversely proportional to memory size, achieving the same accuracy for Φ^- interpolation as for Φ^+ interpolation requires twice as much memory. The desired values of $\log_2(\Delta)$ are on the right of Fig. 3, where the formally-proved interpolation bounds given here are asymptotic to the exhaustive-simulation results. Where the bounds diverge pessimistically on the left are for over-provisioned $\Delta \approx \epsilon$ unlikely to be used in an actual implementation. LNS designers can trust our Lean-validated interpolation bounds are both valid and tight for practical use.

On the other hand, the bound derived here for Coleman's cotransformation is one to two bits more pessimistic than the actual result because it assumes the worst-case error bound for Φ^- interpolation in all computations used with cotransformation. Another reason is the cotransformation version [14] used here has a third case that interpolates Φ^- twice. The original Coleman cotransformation [4] only has two cases and one Φ^- interpolation, which would allow the bound to be tighter.

VI. Conclusion and Future Work

In summary, this work demonstrates the effectiveness of theorem proving with Lean in addressing verification problems that exceed the capabilities of SMT-based tools. We performed a comprehensive error analysis of table-based Logarithmic Number Systems (LNS), focusing on challenges arising from non-linear approximations for addition and subtraction. Through detailed examination of errors from first-order Taylor approximation and cotransformation, we derived symbolic error bounds and formally verified their correctness using Lean and Mathlib. Empirical validation through an exhaustive Python implementation confirmed the accuracy of our interpolation bounds and highlighted the conservativeness of our cotransformation bounds. This study not only deepens the theoretical understanding of LNS but also underscores Lean's value as a robust tool for formal error analysis in numerical methods.

Although hardware for LNS via linear Taylor interpolation is well-known [4], [24], [25], this is the first rigorous machine-verified bound on such hardware. We are currently considering bounds for Coleman's Error Correction (EC) [4] and other quadratic methods [9] of interpolation. Future work includes deriving tighter bounds for cotransformation, including other cotransformations [19], [22], which unlike Coleman's cotransformation [4], do not use the less accurate Φ^- interpolation to implement cotransformation. Although we did not attempt to model bases other than two [16], guard bits [10], varying Δs [4], [9], etc., these would be straightforward extensions to our Lean code.

References

[1] S. John, "The future of computing beyond moore's law," 2020, philosophical Transactions of the Royal Society, http://doi.org/10.1098/rsta.2019.0061.

[2] D. Reed, D. Gannon, and J. Dongarra, "Hpc forecast: Cloudy and uncertain," *Commun. ACM*, vol. 66, no. 2, p. 82–90, jan 2023. [Online]. Available: https://doi.org/10.1145/3552309

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

[4] J. N. Coleman, E. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the european logarithmic microprocessor," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 702–715, 2000.

[5] "XLNS Research," website: http://www.xlnsresearch.com, repository: https://github.com/xlnsresearch.

[6] J. Zhao, S. Dai, R. Venkatesan, B. Zimmer, M. Ali, M.-Y. Liu, B. Khailany, W. J. Dally, and A. Anandkumar, "Lns-madam: Low-precision training in logarithmic number system using multiplicative weight update," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3179–3190, 2022.

[7] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod *et al.*, *Handbook of Floating-Point Arithmetic*, 2nd ed. Birkhauser, 2018, softcover reprint of the original edition.

[8] M. Croci, M. Fasi, N. J. Higham, T. Mary, and M. Mikaitis, "Stochastic rounding: implementation, error analysis and applications," *Royal Society Open Science*, vol. 9, no. 3, p. 211631, 2022.

[9] H. Fu, O. Mencer, and W. Luk, "Fpga designs with optimized logarithmic arithmetic," *IEEE Transactions on Computers*, vol. 59, no. 7, pp. 1000–1006, 2010.

[10] M. Arnold, E. Chester, and J. Cowles, "Guarding the guards: Enhancing lns performance for common applications," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2016, pp. 123–130.

[11] Gaussian logarithm, "Gaussian logarithm — Wikipedia, the free encyclopedia," 2021, [Online; 16 July 2023]. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian_logarithm

[12] M. G. Arnold, T. A. Bailey, and J. A. Cowles, "Towards automated verification of logarithmic arithmetic," *arXiv preprint arXiv:2411.12923*, 1994.

[13] J. N. Coleman, C. I. Softley, J. Kadlec, R. Matousek, M. Tichy, Z. Pohl, A. Hermanek, and N. F. Benschop, "The european logarithmic microprocesor," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 532–546, 2008.

[14] J. N. Coleman and R. C. Ismail, "Lns with co-transformation competes with floating-point," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 136–146, 2015.

[15] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *arXiv preprint arXiv:1603.01025*, 2016.

[16] 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.

[17] P. Haghi, C. Wu, Z. Azad, Y. Li, A. Gui, Y. Hao, A. Li, and T. T. Geng, " Bridging the Gap Between LLMs and LNS with Dynamic Data Format and Architecture Codesign ," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2024, pp. 1617–1631. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MICRO61859.2024.00118

[18] N. Belanger and Y. Savaria, "On the design of a double precision logarithmic number system arithmetic unit," in *2006 IEEE North-East Workshop on Circuits and Systems*. IEEE, 2006, pp. 241–244.

[19] M. G. Arnold, T. A. Bailey, J. R. Cowles, and M. D. Winkel, "Arithmetic co-transformations in the real and complex logarithmic number systems," *IEEE Transactions on Computers*, vol. 47, no. 7, pp. 777–786, 1998.

[20] M. Basir, R. Ismail, and S. Naziri, "An investigation of extended co-transformation using second-degree interpolation for logarithmic number system," in *2020 FORTEI-International Conference on Electrical Engineering (FORTEI-ICEE)*. IEEE, 2020, pp. 59–63.

[21] M. Basir, R. Ismail, and M. Isa, "A novel double co-transformation for a simple and memory efficient logarithmic number system," in *2020 IEEE International Conference on Semiconductor Electronics (ICSE)*. IEEE, 2020, pp. 25–28.

[22] Y. Popoff, F. Scheidegger, M. Schaffner, M. Gautschi, F. K. Gürkaynak, and L. Benini, "High-efficiency logarithmic number unit design based on an improved cotransformation scheme," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1387–1392.

[23] I. Orginos, V. Paliouras, and T. Stouraitis, "A novel algorithm for multi-operand logarithmic number system addition and subtraction using polynomial approximation," in *Proceedings of ISCAS'95-International Symposium on Circuits and Systems*, vol. 3. IEEE, 1995, pp. 1992–1995.

[24] D. M. Lewis, "An architecture for addition and subtraction of long word length numbers in the logarithmic number system," *IEEE Transactions on Computers*, vol. 39, no. 11, pp. 1325–1336, 1990.

[25] M. G. Arnold, T. A. Bailey, and J. R. Cowles, "Comments on 'an architecture for addition and subtraction of long word length numbers in the logarithmic number system'," *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 786–788, 1992.