This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

LogSumExp: Efficient Approximate Logarithm Acceleration for Embedded Tractable Probabilistic Reasoning

Lingyun Yao, *Student Member*, Shirui Zhao, *Student Member*, Martin Trapp, Jelin Leslin, *Student Member*, Marian Verhelst, *Fellow, IEEE*, and Martin Andraud, *Member, IEEE*

.

***Abstract*—Probabilistic models (PMs) have become an alternative to complement or replace deep learning in applications where transparency and trustworthiness are crucial. As PMs compute explicit high-resolution probabilities, ensuring numerical stability legitimates the need for logarithmic (log) computing. As exact log computation on hardware is typically costly, existing hardware accelerators stick to high-resolution linear computation with, *e.g.*, floating point (FP). From the perspective of efficient execution on edge devices, using such generic linear hardware for log operations is prone to underflow and ill-suited for operations such as log addition. Hence, the log-domain computing of PMs requires new hardware solutions, combining numerical stability and energy-efficient execution. Inspired by the Log-Sum-Exp (LSE) function used in existing PM software tools transferring data between log and linear domains to compute log additions, this work proposes an LSE Processing Element (LSE-PE). LSE-PE allows for efficient log computation, through an innovative double approximation for log addition, while ensuring numerical stability with an error compensation method using a compact error correction Look-Up Table (CLUT). Hardware synthesis results using a 16nm technology show that the proposed 24-bit LSE-PE hardware consumes 46% area and 32% power of 32-bit floating point, using only 16 LUT entries with 10 bits in each entry. Moreover, our experiments on various PM benchmarks show that LSE-PE prevents underflow even for large models, which exist in all other 32-bit number systems, with less than 0.2% accuracy loss. We also demonstrate an outlier detection task for uncertainty estimation of image classification models using the LSE-PE, for a fraction of the main model's computing cost (0.06 to 20% of representative DNN architectures for MNIST).**

***Index Terms*—approximate computing, log-sum-exp, probabilistic models, number system, hardware acceleration**

I. Introduction

THE development of explainable and trustworthy AI is essential for decision-making systems, particularly in applications such as healthcare and automotive [1]. In this context, deep neural networks (DNNs) have raised concerns about their interpretable nature and overconfidence [2], associated with increased model size and high inference costs [3], [4]. This pushes for alternatives, such as probabilistic approaches [1]. Probabilistic models (PMs) have shown their potential in areas such as speech or image recognition [5]–[7], semantic mapping [8] or neurosymbolic AI, in outlier detection [9], uncertainty estimation [10] or rule-based integration [11]. Although software PM implementations exist, compute-efficient hardware frameworks generally lag behind their DNN counterpart, making their *acceleration* challenging. Indeed, the explainable nature of PMs arises from using explicit probabilities. To capture these differences, as well as other out-of-distribution events, it is crucial to represent log-likelihood values as small as $1.2 * 10^{-4}$, (or 2^{-12000} in a linear representation) [9]. The ranges offered by floating point (FP) or custom Posit (CUSPOS) [12] are insufficient, legitimating the need for log computing. While moving to log is straightforward for multiplications (becoming additions), log addition requires more resources [13]. To maintain sufficient precision, PM trained on generic computing platforms use the *Log-Sum-Exp* (LSE) function [9], [14], [15]. LSE converts the data to the linear domain to perform a linear addition and then back to the log, avoiding underflow. Yet, in a embedded scenario, existing accelerators [12], [16]–[18], prefer to stay in the linear domain (using FP or CUSPOS) for better computational efficiency. Indeed, exact log computation is prohibitive, especially for log addition [13], while an exact LSE hardware would require more resources to transfer between domains (log and exp). To solve this issue, recent solutions explored *approximate* log computations, using either LUT combined with interpolation functions to approximate the log addition, [19], [20] or direct approximation blocks that use linear functions to approximate log and exp [21]. Yet, maintaining accuracy with a LUT approach can quickly increase hardware resources with increasing range requirements [19], while direct linear function approximation can cause 1 to 10% errors [21].

In this work, we propose a solution for a reliable and efficient hardware acceleration of probabilistic reasoning, centered around three contributions:

1 **Selecting a suitable family of *Tractable* PMs for hardware acceleration**. Probabilistic Circuits (PCs) [22] or tractable PMs (TPM), are a particularly relevant application for hardware acceleration [12], [16], [18] as they generalize multiple PMs in a single computation-oriented framework and allow for probabilistic and symbolic reasoning. PCs are compatible with DNN frameworks [9] and their combination with DNNs [7], [23] has paved the way for enhanced decision-making

Lingyun Yao, Jelin Leslin, and Martin Andraud are with the Department of Electronics and Nanoengineering, Aalto University, 02150 Espoo, Finland (e-mail: firstname.lastname@aalto.fi).

Martin Andraud is also with the ICTEAM, UCLouvain, Belgium (e-mail: firstname.lastname@uclouvain.be)

Martin Trapp is with the Department of Computer Science, Aalto University, 02150 Espoo, Finland (e-mail: firstname.lastname@aalto.fi).

Shirui Zhao, Marian Verhelst are with the are with the Department of Electrical Engineering - MICAS, KU Leuven, Belgium (e-mail: firstname.lastname@kuleuven.be).

This work has been partially funded by the Research Council of Finland's project WHISTLE (grant 332218) and the Eic pathfinder project SUSTAIN(grant 101071179). MT acknowledges funding from the Research Council of Finland (grant number 347279)

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

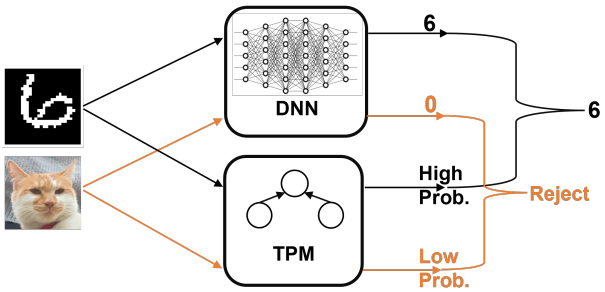

Figure 1: TPM as an outlier detector, monitoring the output of a main DNN for handwritten digit (MNIST) classification.

systems with statistical robustness and high-level symbolic reasoning.

2 **Designing a high-resolution yet efficient log computing framework for TPMs**. Inspired by software LSE [9], we introduce a novel LSE processing element (PE) LSE-PE. It is capable of handling the extensive computational range required by PM computing while being energy-efficient, requiring less power and less area than linear computation blocks. Using 24b fixed point log computation for extended range, it combines a unique double approximation technique, which can be implemented with simple hardware. To preserve the computation accuracy, an efficient error compensation block is also designed, using only 16 Correction LUT (CLUT) entries. Experiments show it can compute large PMs while having a small impact on their accuracy (max. 0.2% loss).

3 **Demonstrating how LSE-PE can be used in an outlier detection scenario**. We show how LSE-PE can enable the computation of hardware-efficient TPMs, monitoring the output of a main DNN to perform uncertainty estimation. The achieved resolution of LSE-PE allows the TPM to detect outlier training samples from the original DNN's training set, hence potentially wrong DNN outputs. In addition, the TPM runs at a fraction of the DNN's computing cost (0.4 to 20% depending on representative DNN architectures for MNIST).

This work is organized as follows: Section II provides the background on PMs and PCs; Section III introduces their computation; Section IV details the proposed LSE-PE; Section V summarizes the performance of LSE-PE; and Section VI explains a case study based on LSE-PE.

II. Background on Tractable Probabilistic Models

A. Background on tractable probabilistic models

Probabilistic Models (PMs) are general-purpose reasoning systems, in which probabilistic reasoning relates to answering specific inference queries [1]. As PMs make their way to real-world applications, *tractable* models recently raised interest, as they provide exact inference while being computationally feasible. This is the case of Probabilistic Circuits (PCs) [22], a unifying framework introduced to harmonise various advancements in designing tractable probabilistic representations (*e.g.*, SPNs [24], PSDDs [25], and CutsetNetworks [26]). Moreover, they subsume many classical probabilistic models such as mixture models and hidden Markov models (HMMs), and exhibit high expressive efficiency (representational power) while maintaining efficient and exact computation of many probabilistic inference queries. Finally, PCs have been recently vectorised for better scalability [9]. They can be trained in mainstream deep learning frameworks such as Tensorflow and PyTorch. Hence, we chose a PC as a baseline in this work.

B. Probabilistic Circuits: introduction

A PC represents a probability distribution and 'answers' to a probabilistic 'query'. For instance, if we evaluate every node in the PC bottom up for an observed state, we compute the probability of the state being true. PCs consist of a computational graph given by a directed acyclic graph comprising sum $\oplus$ and product $\otimes$ nodes. Leaves are 'simple' distributions, typically chosen to be tractable univariate probability distributions or indicator functions in case of discrete data domains. Figure 2 illustrates a PC over discrete variables (X_1, X_2, X_3), representing the joint distribution $P(X_1, X_2, X_3)$ through a computational graph consisting of weighted summations and multiplications[1]. Intuitively, product nodes represent independence assumptions between random variables (RVs) and sum nodes represent mixtures, i.e., replacing the independence assumptions of product nodes with conditional independence assumptions. In addition, each node in a PC is associated with a scope, assigned through a scope function [27], which determines the set of RVs/input dimensions a node defines a probability distribution over.

a) Benefits of PCs: An important property of PCs, which makes them stand out from classical probabilistic graphical models (*e.g.*, Bayesian networks) and modern deep learning, is that they offer a tractable representation of many probabilistic queries and guarantee that computations can be performed exactly and efficiently. Given a class of queries/inference tasks $\mathcal{Q}$ (*e.g.*, computing marginals, or maximum-a-posterior queries) and a family of models $\mathcal{M}$, we say that $\mathcal{M}$ is a tractable representation of a class of queries $\mathcal{Q}$ if we can answer every $q \in \mathcal{Q}$ for every model $m \in \mathcal{M}$ in polynomial time measured in the model complexity $\mathcal{O}(\text{poly}(|m|))$.

b) Learning PCs: Learning a PC typically involves two steps: (i) learning the structure and (ii) learning the parameters. The structure can be learned from data (e.g., [27], [28]) or chosen to be random but sufficiently large (e.g., [9]). Parameters are typically learned by employing expectation maximisation (EM) or can be obtained analytically if the PC is deterministic. Once the PC has been learned, many queries can be answered without re-training.

c) Vectorized PCs: PCs have been recently vectorized for better scalability [9], [15]. They can be trained in mainstream deep learning frameworks such as Tensorflow and PyTorch.

III. Case study: Outlier detection with EiNets

As a case study throughout this work, we propose to reproduce the uncertainty estimation experiment from [9] and port it to hardware. For this task, shown in Fig. 1, the main

[1]The reader is referred to [22] for an in-depth introduction to PCs.

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

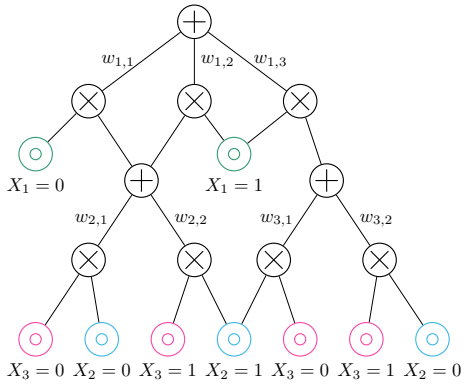

Figure 2: Example of a probabilistic circuit.

DNN model performs an image classification task (here a hand digit classification on MNIST). A TPM is added to the DNN to monitor the same input features and estimate the prediction uncertainty. Essentially, the TPM calculates the probability that the current input has similar features to the data the model was trained on, and this probability value decides whether the data can be "trusted" or not. For that, we first train the DNN and the TPM on MNIST data. For the DNN, we maximise the classification accuracy, and for the TPM, we maximise the average log-likelihood output to fit the MNIST distribution. To create outliers, we take input data from image datasets similar to MNIST but with slightly different characteristics, SEMEION and SVHN. The data is resampled to fit the same feature sizes as MNIST and fed to the TPM. In theory, if the data is different from the training, the output likelihood will drop, increasing the chance of being an outlier.

Our case study throughout this work. To embed this outlier detection task on hardware, we need three steps: ❶ select a suitable TPM (in sect.III); ❷ explain why existing computing solutions can't achieve the required resolution with a reasonable hardware cost (in sect.V); and ❸ see how the proposed LSE hardware solve this bottleneck (in sect.VI).

A. Small EiNets for outlier detection

Here, we carry on step ❶: selecting a suitable TPM. While the TPM used in our can be a PC, this approach is viable under two key conditions: the PC maintains a lightweight structure compared to the main DNN; the computational challenges associated with PCs are effectively addressed. We start by evaluating the model in this section (while its computation will be assessed after our experiments).

We consider Einsum networks (EiNets) [9] as a baseline, for its efficient vectorised architecture and compatibility with PyTorch. As a reminder, the model should learn the probability distribution of the MNIST input data to detect outliers (i.e. low log-likelihood events). The original EiNet[2] presented for our case study in [9] is a large model, comprising 107M parameters. In an embedded scenario, computing such a model would lead to a large cost, potentially higher than the main DNN. Hence, we wish to train a smaller model[3] and evaluate whether it could still detect outliers.

Table I: Comparison of the number of operations required for three DNN implementations and the PC as outlier detection

Main DNN model	**(3 implementations)**	
Model name	**Accuracy**	**# MACs**
LeNet-5	99.05%	**416,520**
MLP	99.56%	**1,346,000**
Improved VGG16	99.97%	**18,824,192**
PC (Act as outlier detector)	**Layer**	**# MACs**
Einsum	$8 \times \sum_{i=1}^{10} 2^i$	16,386
Bottom Layer	2×2^{11}	4,096
Sum Layer	1	1
Total(6b)		**20,483**
Total(24b)		**81,932**

In our experiments (see Section VIII), we choose an EiNet configuration with dimension $K = 2$ and $D = 10$, with 784 input features as binary leaves (28x28 pixel MNIST image). On MNIST samples, the average log2 likelihood on the test Data is -261.45. Note that this is lower than the original model, as our small model has less abilities to fit the complex MNIST distribution, but is significantly more compact.

To compare the number of operations in MNIST classification, we chose three baseline DNN models with different structures: the basic LeNet-5 [29] with 99.05% accuracy, a relatively simple MLP-based DNN model [30] for MNIST with 99.56% accuracy, and a state-of-the-art improved VGG16 CNN structure [31] with 99.97% accuracy. Table I lists the number of operations required by each model, showing the compactness of the Einsum model, requiring only 20k MACs, compared to 416k, 1 346k and 18 824k for the three DNN models. This satisfies the first condition made at the beginning of this section (the PC has a lightweight structure compared to the DNN). Yet, a higher resolution is required to compute the PC. On a representative validation set, the minimum Log 2-Likelihood that we recorded is -567.69, which requires double precision floating point format to represent the number. For intermediate results and outlier probabilities, we need even higher precision. In turn, we require more efficient hardware to address the computational challenges of PCs. This is the goal of the remainder of this work.

IV. Computing Probabilistic Circuits

A. Computing PCs on generic processors

When PCs are trained "in software", they are actually computed through generic computing platforms, such as CPUs and GPUs. As these platforms use linear 32b or 64b floating point, and training PCs involve computing tiny probabilities, the "Log-Sum-Exp" trick is used to avoid underflow, using logarithm (log encoding). Assuming a PC encoded in the log domain, multiplications become additions and additions are computed using the "Log-Sum-Exp" (LSE) function, defined as [9]:

[2]https://github.com/cambridge-mlg/EinsumNetworks.git

[3]https://github.com/braun-steven/simple-einet.gitt

$$\mathrm{LSE}(x_1, x_2, \ldots, x_n) = c + \log\left(\sum_{i=1}^{n} e^{x_i - c}\right), \quad (1)$$

where $c = \max(x_1, x_2, \ldots, x_n)$. Notably, the LSE function introduces a constant c, which scales the exponential values to prevent an underflow. Hence, smaller terms $e^{x_i - c}$ are not neglected, leading to more accurate results. Note that the LSE function for n inputs can be written as a sequence of LSE functions over two inputs through function composition, *i.e.*, $\mathrm{LSE}(x_1, x_2, x_3, x_4) \approx \mathrm{LSE}(x_1, \mathrm{LSE}(x_2, \mathrm{LSE}(x_3, x_4)))$. The two-input LSE function can be expressed as:

$$\mathrm{LSE}(x, y) = c + \log\left(e^{x-c} + e^{y-c}\right), \quad (2)$$

with $c = \max(x, y)$. Assuming that $x \geq y$, we can write $c = x$, and $\mathrm{LSE}(x, y) = x + \log\left(e^0 + e^{y-x}\right)$.

Here, inputs x and y are log values encoded in 32b or 64b floating point, and later applied LSE function. This is different from encoding and computing them in floating-point hardware (linear values encoded in floating point) or logarithmic hardware (log values encoded in fixed point).

B. Computing PCs on Hardware

Table II: Range and Configuration of Various Number Systems

Number System	Bit Size	Exponent Range	Configuration
Float	32-bit	2^{-126} to 2^{127}	8 exponent bits
Posit	32-bit	2^{-120} to 2^{120}	es = 2
Custom Posit	32-bit	2^{-1920} to 2^{1920}	es = 6
LSE-PE	24-bit	2^{-16384} to 2^{0}	14 integer bits

The simple operations and the capabilities of PCs led to specialized hardware for their acceleration [12], [13], [16], [20], [32]. Yet, accelerating PCs faces two main challenges: (1) their sparsity, resulting in very low parallelization opportunities; and (2) their resolution, as computing a PC involves multiplying, adding and *propagating* smaller probabilities as we progress through the PC graph [32]. In that regard, Table II depicts the range covered by various number systems. As highlighted in previous work [13], [32], probabilities can become so tiny (for instance 10^{-88} for some benchmarks) that linear number systems would quickly lead to underflow. That is why existing accelerators such as [12], [16], [18], [33] are limited to relatively small-scale PCs (for instance, 500 nodes for [16] or around 1500 for [18]).

It should be noted that different applications require different computation precisions. There exist various methodologies to determine the optimal hardware precision for the PC, by analyzing the theoretical bound of a given PC graph [32], performing a data-driven precision search [16], or a combination of both [18]. In this work, we focus on finding a single precision setting, suitable for the computation of large PCs.

1) Linear computing hardware:

a) Floating point (FP): FP encodes a number v as: $v = (-1)^S \times 2^E \times 1.M$. S denotes the sign bit, M is the mantissa, and E is the exponent, which includes a bias value. In FP, the precision is determined by the number of mantissa bits, while the dynamic range is governed by exponent bits. FP introduces a rounding error in the mantissa, as it must be normalised between 0 and 1. FP32 with IEEE 754 standard covers a range of [2^{-126} to 2^{127}] [34], too limited for the computing the PC.

b) Posit (POS): The Posit number system aims to allocate the available bits more dynamically [35]. This is done through an additional regime field. POS encodes a number as: $v = (-1)^S \times \mathrm{useed}^k \times 2^E \times 1.M$, where $\mathrm{useed} = 2^{2^{es}}$,$k$ is the regime. The exponent range for a Posit number with n bits and es exponent bits is given by: $\left[2^{-(2^{es}\times(n-2))}, 2^{2^{es}\times(n-2)}\right]$ [35]. In situations where under-/overflowing may occur, the length of the mantissa can be traded off for exponent or regime bits [36]. By using a larger number of regime bits, it is possible to extend the range of 32-bit custom Posit (CUSPOS32) to [2^{-1920} to 2^{1920}] with es set as 6 in [12], still falling short for large PCs.

2) Logarithmic computing hardware:

a) Logarithmic Number System (LNS): In LNS, a positive real number is represented by its logarithm l and typically encoded as 2^l, where l is stored as a fixed-point, making its distribution exponential. Assuming n integer bits, the range of LNS is $\left[0, 2^{2^n-1}\right]$. Under the same bit-width, FP and fixed-point LNS offer comparable dynamic ranges, yet they exhibit different arithmetic characteristics [37]. Indeed, multiplication and division in LNS are implemented as fixed-point additions and subtractions, which are cheap, exact and without rounding errors [38]. In contrast, addition and subtraction in LNS are costly. For instance, assuming two LNS variables x and y, their exact addition denoted as $\mathrm{LNS}_{\mathrm{ADD}}$ (with $x \geq y$) is:

$$\mathrm{LNS}_{\mathrm{ADD}}(x, y) = \log_2(2^x + 2^y) \quad (3)$$

$$= x + \log_2(1 + 2^{(y-x)}), \quad (4)$$

which introduces rounding errors and requires complex hardware [37]. Note that $\mathrm{LNS}_{\mathrm{ADD}}(x, y)$ is equivalent to the 2 inputs LSE of (Eq. (1)) in base 2.

V. State of the art of logarithm hardware

Here, we carry on step ❷: showing why existing LNS hardware can't compute large TPMs. After making a list of recent LNS implementations, we compare various approaches in terms of performance and resolution.

A. LNS hardware implementations

A common method to perform LNS addition is to use a look-up table (LUT) [19]:

$$\mathrm{LNS}_{\mathrm{ADD}}(x, y) \approx x + f(y - x), x \geq y. \quad (5)$$

where $f(y - x) = \log_2\left(1 + 2^{(y-x)}\right)$, $x \geq y$, can be precomputed and stored as $\mathrm{LUT}(y - x)$.

The primary drawback of this approach is that for accurate LNS addition, the size of the LUT grows exponentially in the range of $y - x$. If n bits are required to represent $\max(y - x)$,

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

the LUT would need 2^n entries and the total memory size of the LUT would be $n \cdot 2^n$, assuming each entry is stored with n bits. For this, we use the name "Order-0 LUT", such LUT becomes infeasible after n=16 bits [39].

To reduce the number of LUT entries, it is possible to interpolate the values. For instance, an "order-1" LUT, needs $2^{n/2}$ LUT entries, thus limited to 32 bits. Going further, it is possible to use an "order-2" LUT with $2^{n/3}$ LUT entries for slower LUT size growth, combined with polynomial helper functions for the interpolation [20]. This makes it possible for wider input ranges to be applied to probabilistic computing. This approach is the only known logarithm hardware design specifically for probabilistic computing, implemented on FPGA. The power consumption was estimated using the Xilinx Power Estimator (XPE) tool, based on hardware resource utilization from [13]. Specifically, their LNS design used 793 slices, 20 DSPs, and 1.5 BRAMs, with a estimated power consumption of 53 mW (39 mW from slices, 12 mW from DSPs, and 2 mW from BRAMs). For comparison, their 36-bit (10,26) floating-point (FP) format design used only 192 slices and 2 DSPs, consuming 10 mW (9 mW from slices and 1 mW from DSPs). Thus, the Order-2 LUT method is too costly in terms of power and area, even compared to floating-point formats.

Alternatively, a direct approximation can be used, in the form $x + \tilde{f}_{y-x}$, without relying on LUT [21]. However, this approach suffers from large approximation errors, which also increase with the dynamic range. Notably, it exhibits a maximum absolute error of 3.4×10^7, which is unacceptable for precision-critical applications.

B. Existing log hardware comparison

The previously discussed LNS implementations are summarized in Table III. In addition, we compare them with LSE implementations on generic processors including CPU (Intel® Xeon® E5-2698 v4) and GPU (NVIDIA Tesla V100-SXM2-16GB). To ensure a fair comparison, we execute a batch of 10^7 logarithmic MAC operations on both the CPU and GPU to fully utilize the computational resources. We measure the total runtime t, and compute the number of operations per second (Op/s). The GPU power is recorded using the `nvidia-smi` tool, while for the CPU, we report the operating power as the maximum power, since direct power measurement is less accessible. On both CPU and GPU, the logarithmic MAC operation is implemented using PyTorch's `logsumexp` function. The floating-point representation of logarithm values provides a sufficient computation range and accuracy in most cases, yet with a large maximum LSE error due to the large absolute quantization error in the floating-point format. It introduces complex non-linear operations on floating-point hardware, resulting in low throughput and high power consumption.

Custom hardware achieves relatively better performance overall. The order-2 LUT approach is the only one with enough range to compute medium PCs, with an FPGA implementation using a large amount of resources, which is challenging to extend the range further. The two custom approaches in [21] use less area resources under fewer bits, yet with a limited and non-scalable range (Order-0 LUT) or potentially large approximation errors (No LUT). In contrast (as detailed in Section VI), our proposed correction LUT (CLUT) method avoids an exponential LUT growth with the number of bits, maintaining a low power and area while limiting the approximation error.

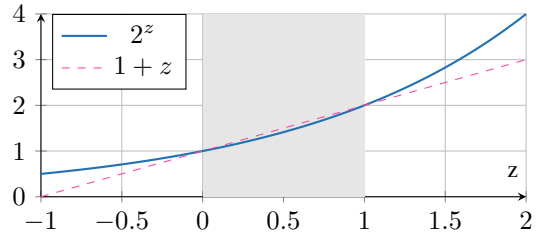

Figure 3: Illustration of $\widetilde{exp}$: $2^z \approx z + 1$

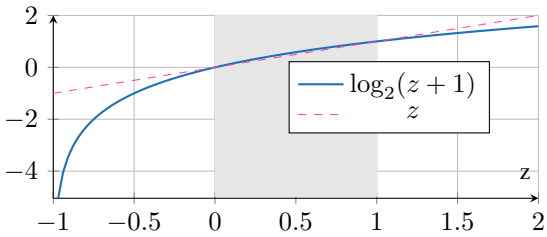

Figure 4: Illustration of $\widetilde{\log}$: $\log_2(z + 1) \approx z$

VI. Proposed LSE-PE computation

To address the challenges associated with accurate and efficient hardware log addition, we propose the LSE-PE architecture, aiming to improve the hardware efficiency of log domain addition (Eq. (5)) using: (1) a two-stage approximation method to circumvent the computationally intensive exponential and logarithmic calculations of exact LSE, and (2) an error compensation mechanism by an innovative correction lookup table (CLUT) to preserve accuracy even with small probabilities, this CLUT is just for error correction, not a full approximation of $f(y - x)$. The proposed LSE-PE is summarised in Algorithm 1 and detailed in the following subsections.

Algorithm 1 Proposed LSE-PE

function LSE-PE(x, y)
  assume $x \geq y$
  Sub. $\leftarrow y - x$
  $I_{(y-x)} \leftarrow$ Int.(Sub.), $F_{(y-x)} \leftarrow$ Frac.(Sub.)
Two-stage approximation (sec.III-B)
  $\tilde{f}_{(y-x)} \leftarrow (1 + F_{(y-x)}) \gg (-I_{(y-x)})$
Error correction (sec.III-C)
  return $x + \tilde{f}_{y-x} +$ CLUT$(\tilde{f}_{y-x})$
end function

A. Algorithm of proposed LSE

As a reminder, the task is to compute Eq. (4) accurately and efficiently. Here, the variables x and y are in the log

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

Table III: State of the art of Logarithm-computing hardware

Method	Platform	Format	Op/s	Power (μW)	Area (μm^2)	Max.Abs.Err	Range
FP encoded log	CPU (14nm)	FP32	58	$< 1.4 * 10^8$	$2.4 \cdot 10^9$	$2^{(1.01 \cdot 10^{31})}$	$2^{-2^{127}}$–$2^{2^{127}}$
FP encoded log	GPU (12nm)	FP32	76	$5.6 * 10^7$	$8.2 \cdot 10^8$	$2^{(1.01 \cdot 10^{31})}$	$2^{-2^{127}}$–$2^{2^{127}}$
Order-2 LUT [20]	FPGA	FIX(9,31)	240	53000	-	$2^{-19.5}$	2^{-512}–2^{0}
Order-0 LUT [21]	ASIC (28nm)	FIX(7,6)	500	461	594	2^{-7}	2^{-64}–2^{63}
No LUT [21]	ASIC (28nm)	FIX(7,6)	500	259	384	$3.4 * 10^{17}$	2^{-64}—2^{63}
Our CLUT	**ASIC (16nm)**	**FIX(14,10)**	**500**	**460**	**829**	**0.001**	$\mathbf{2^{-16384}}$–$\mathbf{2^{0}}$

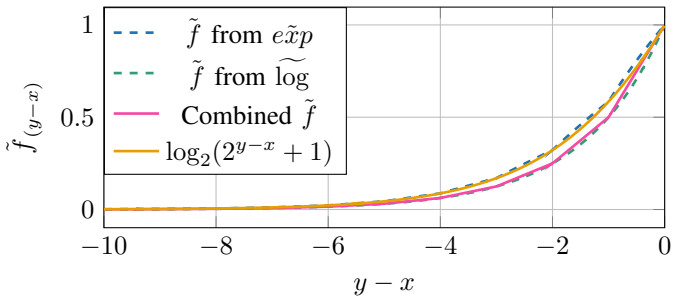

Figure 5: Comparison of $\tilde{f}_{(y-x)}$ under different approximations. Note that using only one of the two approximations ($\widetilde{\log}$ or $\widetilde{\exp}$) does not save all hardware resources as the combined approximations do.

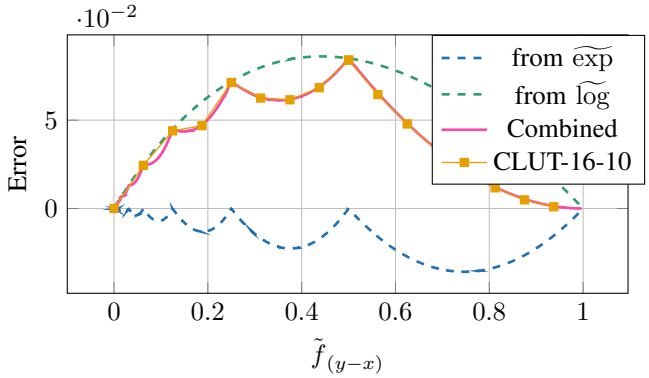

Figure 6: Error of $\tilde{f}_{(y-x)}$ under different approximations.

domain, and we assume $x \geq y$. The log values x and y are represented in fixed-point format, with an integer part I and a fraction part F. The first step of the algorithm is to perform a fixed point subtraction $y-x$ and store the result as Sub. (line 3) after which we extract I_{y-x} the integer part (a negative integer) and F_{y-x} the fractional part (in the range $[0,1)$) of Sub. (line 4). Subsequently, two approximations are used to avoid performing the $2^{(\ldots)}$ and $\log_2(\ldots)$ operations, denoted resp. $\widetilde{\exp}$ and $\widetilde{\log}$. First, $(1+F_{y-x}) \gg (-I_{y-x})$ acts as an approximation of $2^{(y-x)}$ required in Eq. (4), where $\gg$ represents the bit-shift operation. Then, $(1+F_{y-x}) \gg (-I_{y-x})$ itself allows the approximation of $\log_2\left(1+2^{(y-x)}\right)$ required in Eq. (4) (line 6). These two steps allow us to have a $\tilde{f}_{(y-x)}$ in the range of [0,1) so that we can have an small size lookup table CLUT($\tilde{f}_{y-x}$) which only covers the range [0,1) for error correction (line 8), instead of the traditional LUT($y-x$).

B. Two-stage LSE approximation

Here we detail the two consecutive approximations for hardware-efficient implementation of $f_{(y-x)} = \log_2(1+2^{(y-x)})$.

Lemma 1 (Exponential Approximation)**.**

$$\widetilde{\exp}: \quad 2^z \approx z+1. \tag{6}$$

Proof. The first order Taylor expansion of 2^z at $z=0$ gives:

$$\begin{aligned} 2^z = e^{z\ln 2} &= \sum_{n=0}^{\infty} \frac{(z\ln 2)^n}{n!} \\ &= 1 + (\ln 2)z + \frac{(\ln 2)^2}{2!}z^2 + \frac{(\ln 2)^3}{3!}z^3 + \cdots \\ &\approx 1 + (\ln 2)z \approx 1+z. \end{aligned}$$

□

Approximating "$\ln 2$" by 1 in Eq. (6) is known as Mitchell's approximation [40]. It gives a small approximation error when z is small, ignoring higher-order terms.

In our case, Eq. (6) estimates $2^{(y-x)}$. As Fig. 3 shows, the fraction part F_{y-x} of $y-x$ lies in the range $[0,1)$, where the approximation of Eq. (6) is accurate. In detail, it gives:

$$\begin{aligned} 2^{y-x} &= 2^{(I_{y-x}+F_{y-x})} = 2^{F_{y-x}} \cdot 2^{I_{y-x}} \\ &\approx (1+F_{y-x}) \cdot 2^{I_{y-x}}. \end{aligned}$$

Further, multiplying the base 2 exponential of the integer part I_{y-x} can be realised by a bit-shift operation ($\gg$). Hence, the exponential operation is approximated by:

$$2^{y-x} \approx (1+F_{y-x}) \gg (-I_{y-x}).$$

Note that we represent bit-shift operation using $\gg$ instead of $\ll$ since I_{y-x} is negative for probability values (hence we represent it as $\gg$ for positive integer values $-I_{y-x}$).

Example 1 (Exponential Approximation)**.**

Let: $x=-1.4, y=-2.0, y-x=-0.6$; *Then:*

$I_{y-x} = -1, \quad F_{y-x} = 0.4$

$2^{y-x} \approx (1+0.4) \gg 1 = \boxed{0.7}$

Exact: $2^{-0.6} \approx 0.660, \quad$ *Abs. Err.* ≈ 0.040

Lemma 2 (Log Approximation)**.**

$$\widetilde{\log}: \quad \log_2(1+z) \approx z. \tag{7}$$

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

Proof. The first order Taylor expansion of $\log_2(1+z)$ at $z=0$ gives:

$$\begin{aligned}\log_2(1+z) &= \frac{1}{\ln 2}\cdot\ln(1+z) = \frac{1}{\ln 2}\sum_{n=1}^{\infty}\frac{(-1)^{n+1}}{n}z^n \\ &= \frac{1}{\ln 2}\left(z-\frac{z^2}{2}+\frac{z^3}{3}-\frac{z^4}{4}+\cdots\right) \\ &\approx \frac{1}{\ln 2}\cdot z \approx z.\end{aligned}$$

□

Similarly, $\ln 2$ is approximated by 1, and (7) gives a small approximation error when z is small to ignore higher-order terms.

Given that I_{y-x} is negative, $(1+F_{y-x})\cdot 2^{I_{y-x}}$ also lies in the range $[0,1)$, where it can be accurately approximated using equation (7) as shown in Fig. 4. The computation becomes:

$$\begin{aligned}\log_2\left(1+2^{(y-x)}\right) &\approx \log_2\left(1+(1+F_{y-x})\cdot 2^{I_{y-x}}\right) \\ &\approx (1+F_{y-x}) \gg (-I_{y-x}).\end{aligned}$$

Example 2 (Log Approximation)**.**

From Example 1: $2^{y-x} \approx (1+F_{y-x}) \gg (-I_{y-x}) = 0.7$

$$\log_2\left(1+2^{(y-x)}\right) \approx (1+F_{y-x}) \gg (-I_{y-x}) = \boxed{0.7}$$

Exact: $\log_2(1+0.7) \approx 0.766$, *Abs. Err.* ≈ 0.066

The proposed LSE-PE combines these two approximations to approximate the LSE:

$$\tilde{f}_{(y-x)} = (1+F_{y-x}) \gg (-I_{y-x}). \tag{8}$$

Example 3 (Full Approximation)**.**

Let: $x=-1.4, y=-2.0, y-x=-0.6$; *Then:*

$I_{y-x}=-1, \quad F_{y-x}=0.4$

$\tilde{f}_{(y-x)} \approx (1+0.4) \gg 1 = \boxed{0.7}$

Exact: $f_{(y-x)} = \log_2(1+2^{-0.6}) \approx 0.731$, *Abs. Err.* ≈ 0.031

C. Error correction with a correction lookup table (CLUT)

The errors introduced by the approximations are shown in Fig. 5 (the individual contributions of $\widetilde{\exp}$ alone and $\widetilde{\log}$ are given as an additional reference). As $\tilde{f}_{(y-x)}$ lies in the range $[0,1)$, the correction only covers this range. Figure 6 details the error in $[0,1)$, also highlighting individual contributions. The maximum absolute error in the range is 0.085.

The error correction is implemented using a CLUT. This CLUT is only applied to the $\tilde{f}_{(y-x)}$ in the range $[0,1)$. This means we have a fixed range 1 for all lookup table entries to cover instead of covering the range of inputs $(y-x)$. Further, the value for all keys has the range $[0, 0.085]$, which is easy to represent with low-bit.

To enhance accuracy with a reduced number of CLUT entries, we employ linear interpolation. Given two consecutive entries in the CLUT, denoted as $f(x_i)$ and $f(x_{i+1})$, the interpolated value $f(x)$ for an input x that falls between x_i and x_{i+1} can be calculated using:

$$f(x) \approx f(x_i) + \frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}\cdot(x-x_i). \tag{9}$$

The different correction factors are stored in the CLUT, denoted CLUT($\tilde{f}_{y-x}$). Fig. 6 shows an example CLUT with 16 entries, 10 fraction bits and interpolation, closely following the combined error. Including the error correction, CLUT($\tilde{f}_{y-x}$), the final LSE-PE(x,y) function is:

$$\text{LSE-PE}(x,y) \approx x + \tilde{f}_{y-x} + \text{CLUT}(\tilde{f}_{y-x}). \tag{10}$$

D. LSE-PE Hardware Implementation

The computational simplicity of the previous approximations and error correction allows for the implementation of LSE-PE with minimal hardware, as shown in Fig. 7. It is used to build log Multiply-Accumulate (MAC) units, where each unit consists of one multiplier shown as MULT (realised through a fixed-point adder) and an LSE-PE log adder shown as LSE-PE in Fig. 7. In this design, multiple units share access to the same CLUT, but each LSE-PE unit is equipped with its own multiplexer (MUX) to select 10 bits for error correction.

a) Operating principles: The example LSE-PE configuration includes 16 CLUT entries, each with 14 integer bits, and 10 fractional bits. Two 24-bit fixed-point inputs x and y are processed through a 24-bit comparator. The comparator identifies the largest of the two inputs, denoted as $\max(x,y)$, and the smallest input, denoted as $\min(x,y)$. These values are then passed to a subtractor for computing $\min(x,y)-\max(x,y)$. Then, the approximation is performed, computing $(1+\text{Frac.}) \gg (-\text{Int.})$ (adding 1 to the fraction part of $\min(x,y)-\max(x,y)$ and shifting the results by Int. bits), where Int. and Frac. represent the integer and fraction part of $\min(x,y)-\max(x,y)$. $(1+\text{Frac.}) \gg (-\text{Int.})$ is contained in the range $[0,1)$ and uses 10 fractional bits. Finally, the 24-bit $\max(x,y)$, 10-bit $(1+\text{Frac.}) \gg (-\text{Int.})$ and 10-bit error correction from CLUT are added together to form the output of LSE-PE.

b) Error correction: the number of CLUT entries, n, can be adjusted to achieve the desired level of accuracy, allowing for a trade-off between hardware complexity and computational precision. Each entry has the same F bits as fraction bits so the CLUT size is $n \cdot F$. The first $\log_2(n)$ bits of $(1+\text{Frac.}) \gg (-\text{Int.})$ are used to index the CLUT entries, so $\log_2(n)$ must be smaller than F. The remaining bits of F are used as $(x-x_i)$ in Eq. (9) for interpolation between the entries. $f(x_{i+1})$ and $f(x_i)$ are the next value and the current value in CLUT, $(x_{i+1}-x_i)$ is a fixed distance $1/16$ between each index. The interpolated value can be calculated in the circuit by multiplying F by $f(x_{i+1})-f(x_i)$, followed by a bit shift operation. Together with the number of CLUT entries n, the precision of each CLUT entry (the same as the number of fraction bits) F is another approximation parameter that will be further explored in Section VII. The number of integer bits is kept at 14 to offer enough representation range (see Table II), all 14 bits reserved for negative values (as $\log_2$ of probabilities are always smaller than 0).

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

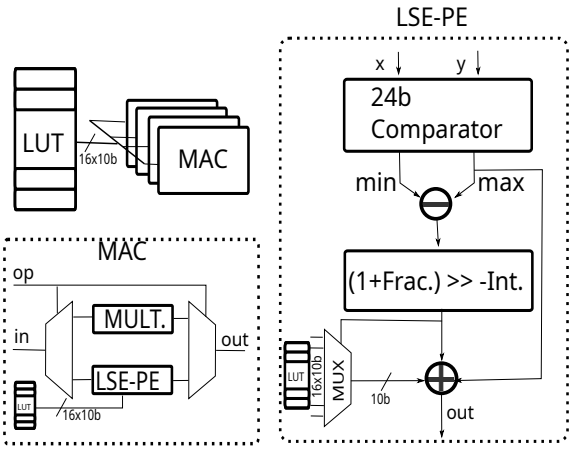

Figure 7: Implementation of 64 parallel MAC units using LSE-PE as logarithmic adders (LSE-PE), fixed-point adder as logarithmic multipliers (MULT.).

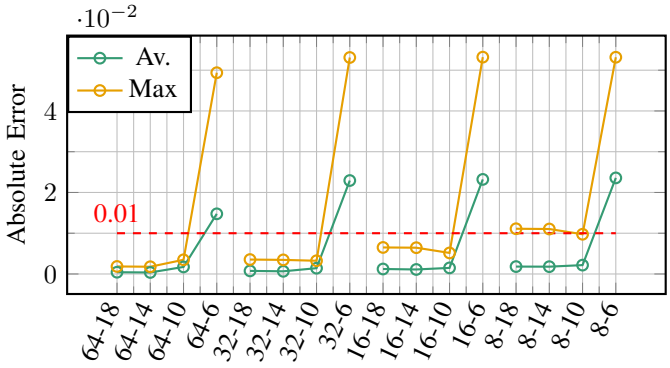

Figure 8: Maximum (Max) and average (Av.) absolute errors on $10,000$ randomly generated inputs x, y in $(0, -10]$ by various CLUT Entries-Fraction Bits configurations.

VII. RESULTS

A. Accuracy of single LSE function under different CLUT configurations

The computation range of the proposed LSE-PE, shown in Table II shows that the LSE-PE configuration effectively addresses the range limitations of other linear number systems (as shown later in Fig. 11). Yet, this range extension should not be done at the expense of accuracy loss. Hence, to evaluate the computational accuracy of our LSE function, we varied the number of CLUT entries and the number of fractional bits. Note that here the LSE-PE is prototyped with software models in Python which are verified by RTL function for accuracy test. The error is calculated using 10,000 randomly generated inputs in the log domain, ranging from $[-10, 0)$, where $\tilde{f}_{(y-x)}$ exhibits mostly non-zero values (see Fig. 5). Maximum and average absolute errors for various configurations are summarised in Fig. 8. The results indicate that, although the error increases when reducing the number of CLUT entries or fraction bits, most configurations achieve less than 0.01 maximum error.

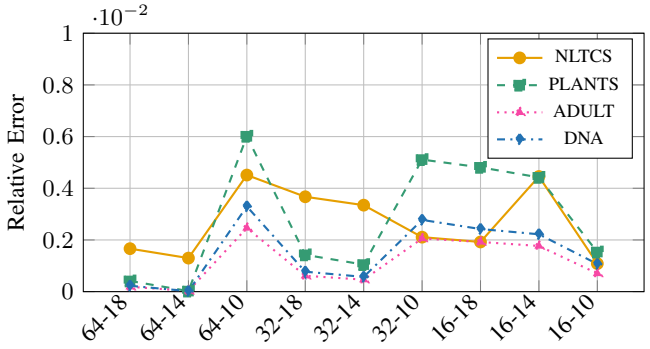

Figure 9: Design Space Exploration for Relative Error of $\log_2$ likelihood on PCs benchmarks by various CLUT Entries-Fraction Bits configurations, Int bits=14.

B. Accuracy Experiments on PC Benchmarks

The second analysis assesses the proposed LSE-PE implementation under representative probabilistic inference tasks, to assess whether LSE-PE can handle a wide range of inference scenarios while maintaining accuracy. The optimal LSE-PE configuration is determined through a design space exploration on four common PC benchmarks [41]. To keep the maximum absolute error within the 0.01 limit, configurations using either 8 CLUT entries and/or 6 fractional bits are excluded. The results shown in Fig. 9 indicate that all combinations of parameters achieved an average relative error of final $\log_2$ likelihood lower than 1%. To minimise hardware resources and keep the maximum absolute error low, we choose a CLUT configuration with 16 CLUT entries and 10 fraction bits.

This configuration is evaluated for ten PC benchmarks, ranging in size from the smallest ('nltcs') to the largest ('ad') [41]. The PCs are generated using Einsum networks [9]. We compare the relative accuracy of the proposed LSE-PE on these benchmarks with the achievable accuracy using SotA number systems used in existing PC accelerators: 32-bit floating point (FP32) and 32-bit Posit (POS32) [13], as well as 32-bit custom Posit CUSPOS32 [12]). For each, we record the average $\log_2$ likelihood and the average relative error of $\log_2$ likelihood. The results are reported in Table IV. Analysing these results, only LNS-PE does not show underflow for the larger benchmarks, which the alternatives encounter due to their limited representation range (denoted Inf in Table IV). Note that even when the average $\log_2$ likelihood is within the representable range, underflow can still occur for intermediate probability values, causing underflow to propagate to the top node. The proposed LSE-PE shows a relative error below 0.2% for all benchmarks. Most importantly, the relationship between $\log_2$ likelihood and error demonstrates that the error does not grow with the size of the model, legitimating LSE-PE for larger PCs.

C. Fitting quality of different LUT configurations

We noted that the maximum error tends to decrease slightly between 14 and 10 fraction bits for CLUT with 8 and 16 entries in Fig. 8 and for CLUT with 16 entries in Fig. 9. Also, 16 entries CLUT do not always result in higher error than 64

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

Table IV: Average relative error of $\log_2$ likelihood on PCs benchmarks (LSE-PE with 16 CLUT entries and 10 frac. bits).

Benchmark	**Features**	**Av. $\log_2$ likelihood**	**Relative Error**			
			FP32	**POS32**	**CUSPOS32**	**LSE-PE**
nltcs	16	-9.85	4.51e-03	2.88e-06	2.90e-06	1.08e-03
msnbc	17	-9.64	3.05e-03	2.89e-06	2.88e-06	4.23e-04
plants	69	-51.39	5.56e-07	1.52e-06	4.77e-07	1.53e-03
jester	100	-98.14	2.76e-07	1.04e-03	2.52e-07	8.98e-04
adult	123	-124.68	2.31e-07	3.78e-02	1.93e-07	7.02e-04
tretail	135	-74.88	3.91e-07	2.17e-05	3.38e-07	1.94e-03
dna	180	-186.63	inf	inf	inf	1.06e-03
mnist	**784**	**-261.45**	**inf**	**inf**	**inf**	**8.87e-05**
bbc	1058	-1031.16	inf	inf	inf	1.61e-03
ad	1556	-1512.51	inf	inf	inf	9.93e-04

entries CLUT in Fig. 9. This indicates that more CLUT entries and more fraction bits do not guarantee higher accuracy. To verify this, we checked the fit quality of some CLUT configurations by calculating the difference of CLUT($\tilde{f}_{y-x}$) and the original error of the $\tilde{f}_{(y-x)}$(Fig. 6), as seen in Fig. 10.

We observe that the error curve in Fig. 6 consists of multiple convex shape parts. For a convex function, any linear interpolation between two points lies above the curve. However, by carefully adjusting the interpolation values to be slightly lower and choosing an appropriate number of interpolation points, we can construct a fitting function that oscillates across the curve, crossing it both above and below.

Due to the quantisation effect, the 10-bit quantised values tend to be slightly lower than the exact values in each CLUT entry. However, when combined with a higher number of CLUT entries, the 64-10 configuration results in consistent directional errors, with most values falling below the true function. Similarly, with fewer CLUT entries (e.g., 16) and higher precision (more fraction bits), the 16-18 configuration tends to produce consistent directional errors in the opposite direction, where most values are higher than the true function. In contrast, the 16-10 configuration causes the fitting function to cross the true curve both above and below. Although this 16-10 configuration introduces larger local errors compared to the 64-18 case, it avoids systematic bias, making it better suited than 64-10 and 16-18 for computations. This explains that more hardware resources do not guarantee higher accuracy, which makes the 16-10 configuration become competitive in both hardware cost and accuracy.

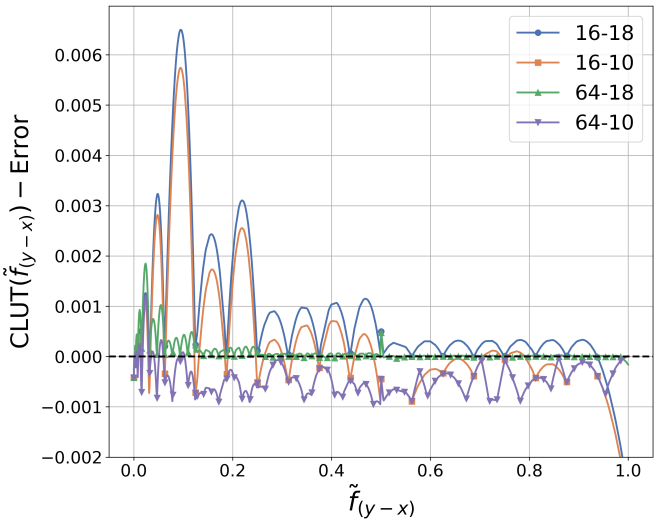

Figure 10: The difference of CLUT($\tilde{f}_{y-x}$) and the original error of the $\tilde{f}_{(y-x)}$ in 4 different CLUT configurations. The LUT with 16-10 configuration shows better performance compared to configurations with more entries (64-10) or higher precision (16-18), due to its error fluctuating around zero rather than being consistently biased in one direction.

D. Comparison with existing hardware

Comparison with PC accelerators. As a last experiment, we evaluate the hardware performance of the LSE-PE for PC acceleration, against several state-of-the-art implementations, as reported in Table V. For a fair comparison, all designs are re-implemented from the original works, using the architecture of DPU [12] and synthesised in a CMOS 16nm technology with Cadence Genus and Verilog. The results demonstrate that our LSE-PE MAC units achieve the lowest power consumption and the smallest area among all designs, notably consuming 46% area and 32% power of FP32.

Comparison with logarithmic hardware. These results can be put in perspective with previous LNS implementations, as reported in Table III. Compared to the Order-2 LUT method, our CLUT significantly reduced the power and area consumption (Order-2 LUT energy cost is 5×FP while our CLUT method costs 0.32×FP). Compared to the Order-0 LUT, it supports wider input ranges without increasing LUT size and with even less power. Compared to the No LUT methods, our design offers larger input range, higher accuracy, and competitive power and area even under double the bit width. In the end, the accumulated error in all benchmarks is lower than 0.2%.

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

Table V: Area and Power of Various Number Systems for 64 parallel MAC units, operate @500 Mhz.

Number System	Bit Size	Area(um^2)	Power(mW)
Fixed32	32-bit	69482	63.91
FP32	32-bit	115911	91.68
Float-add		372	
Float-mult		1259	
POS32 [42]	32-bit	395161	488.98
Posit-add		2739	
Posit-mult		3266	
CUSPOS32 [42]	32-bit	356703	433.27
Posit-add		2519	
Posit-mult		2963	
LSE	24-bit	53049	29.69
LSE-add		696	
LSE-mult		37	

[1] Note that there is no implementation of other LNS blocks for direct comparison, since previous designs are not suitable for PCs as shown in Table III.

E. Scalability of proposed LSE

To compute the original LogSumExp (LSE) expression $x+\log_2\left(1+2^{(y-x)}\right)$, our proposed CLUT-based LSE approximation takes the form $x+\tilde{f}_{y-x}+\text{CLUT}(\tilde{f}_{y-x})$. The error is determined by the error correction term CLUT$(\tilde{f}_{y-x})$, where $\tilde{f}_{y-x} \in [0,1]$. Recall that x and y represent probability values propagating through a probabilistic model and are encoded in fixed-point logarithmic form, $x, y < 0$, assume $y < x$. As the model depth increases, $y, x \to -\infty$, and thus $y - x \to -\infty$. Consequently, $\tilde{f}_{y-x} \to 0$ as shown in Fig. 5. The error term CLUT$(\tilde{f}_{y-x}) = 0$ when $\tilde{f}_{y-x} = 0$, see from Fig. 6. The error introduced by our approximation vanishes in deep propagation chains. This key property helps prevent cumulative error growth and supports scalability. Additionally, Fig. 10 shows that the error introduced by our CLUT approximation fluctuates around zero rather than exhibiting a consistent bias. This behavior further mitigates the accumulation of error over multiple stages of computation. This analysis is consistent with the experimental results in Table IV, where the error does not increase with the size of the probabilistic model.

VIII. Experiments on the Outlier Detection

Here, we carry on step ❸: showing that our LSE hardware achieves the required computation resolution and accuracy in real applications. For that, we implement the EiNet derived in Sect.III to compute the outlier detection experiment from [9]. As a reminder, the TPM to detect outlier inputs to evaluate if we can trust the output of the main DNN model.

A. Experimental setup

We first train both the main DNN and the TPM on MNIST. To evaluate the outlier detection performance, we include two other datasets that serve as outliers: SEMEION and SVNH. The SEMEION dataset includes images of handwritten digits like MNIST, with a size of 16 pixels by 16 pixels, in a binary format. We resampled SEMEION into 28 pixels by 28 pixels, SEMEION raw images. The SVHN dataset consists of real-world images of house numbers extracted from Google Street View images with colored images and varying resolutions, with the size of 32 pixels by 32 pixels on 3 dimensions (Red Blue Green). Here, we converted them to binary images and resampled them to 28 pixels by 28 pixels.

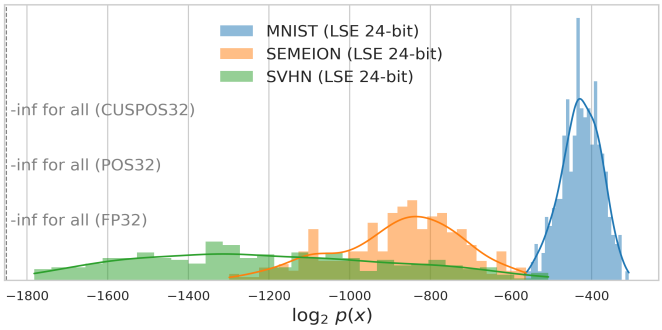

Figure 11: Histograms of log-probabilities of the 300 test sets of MNIST, SEMEION and SVHN using 16 CLUT entries and 10 fraction bits LSE (LSE 24-bit), 32-bit floating point (FP32), 32-bit Posit (POS32) and 32-bit customised Posit (CUSPOS32) under EiNet trained on MNIST training data.

In our experiments, we performed inference on 300 test data from each dataset. We also compare 4 arithmetic formats: 32-bit floating point (FP32) and 32-bit Posit (POS32) [13], as well as 32-bit custom posit (CUSPOS32) [12], and our 24-bit LSE. All arithmetic formats are prototyped software models in Python; the LSE Python model is verified with a System Verilog implementation for producing the same result.

B. Simulation results

We performed in total 900 inference on MNIST, SEMEION, SVHN using FP32, POS32, CUSPOS32 and our LSE. As seen in Table IV, all inference in FP32, POS32 and CUSPOS32 lead to underflow ($-inf$ in log). LSE did not lead to any underflow, and the average relative error due to the approximation of the LSE hardware on MNIST is $8.87.10^{-5}$. To evaluate the possibility of outlier detection, Fig. 11 shows the density distribution of the log2 likelihood results at the PC's outputs. We can observe that even with the compact PC structure, the model can tell SEMEION and SVHN datasets apart from MNIST. Specifically, SEMEION doesn't overlap with MNIST at all. Hence, LSE-PE offers enough resolution to detect out-of-distribution events.

To evaluate the computational differences between the main DNN and the PC, we performed a computational cost comparison. Since our PC is computed by 24-bit LSE, we scale 24-bit MAC operation numbers to 6-bit equivalent MAC operation numbers with the assumption that DNN models operate at 6-bit log for fair comparison as seen in Table I. Compared to the main DNNs, an outlier detection mechanism using our PC requires only 20% of the operations of the simple CNN, 6% of the simple MLP, and only 0.4% of the improved VGG.

Therefore, LSE can enable PCs play an important role to improve the reliability of AI models.

IX. Discussion on other probabilistic applications

This work is part of a growing literature about probabilistic reasoning, which finds a variety of applications in addressing the overconfidence issue of deep learning models [43]. Yet, there exist multiple manners to implement such probabilistic reasoning frameworks. One of them is to directly modify DNNs to make them explicitly probabilistic, leading to Bayesian Neural Networks (BNNs) [44]. Another manner is to implement dedicated probabilistic models, for example, TPMs, in this work.

From a hardware perspective, probabilistic reasoning can be categorized into exact inference and approximate inference. In exact inference, inferences are answered by calculating propagated probabilities, for example, TPMs and exact inference algorithms of Hidden Markov Models (HMMs) [45]. As discussed earlier in this work, TPMs demands high computational resolution. Similar for HMMs, a previous HMMs accelerator has employed log-addition to replace high-precision floating-point units [46]. Such exact inference guarantees reliable output, yet the computational complexity of exact inference can grow exponentially with model size [47]. Alternatively, approximate inference estimates probabilities by sampling from the distribution. Sampling times affect both the computational complexity and the accuracy of the approximation [48]. This method typically requires a high-precision random number generator in hardware. For example, the BNNs accelerator in [44] relies on high-precision floating-point operations to support its random number generation.

Thus, in both exact and approximate probabilistic reasoning, high-precision computation is essential. In this work, we implement our proposed LSE design on a representative set of probabilistic models as shown in Table IV. We expect our design to be applicable to a broader range of log-intensive or high-precision computations, particularly for probabilistic reasoning tasks. Furthermore, the memory access pattern of our LSE design aligns with conventional fixed-point hardware, without introducing additional complexity, and is compatible with different memory architectures.

X. Conclusion

This work proposed LSE-PE, a novel hardware designed for numerically-stable, low-precision, and energy-efficient log-domain probabilistic computing. Our approach combines a double approximation and a resource-efficient error correction scheme. Our accuracy experiments on probabilistic computing benchmarks demonstrate sufficient accuracy and range with less than 0.2% error while all other number systems lead to underflow in large benchmarks. The 24-bit LSE-PE MAC unit configuration achieves a 46% area and 32% power of an equivalent floating-point configuration. We prove that a PC implemented with the proposed LSE-PE can be used for outlier detection and DNN uncertainty estimation, at a fraction of the computing cost of the main DNN model (0.4 to 20% depending on the model's size). This opens the path for a broader use of log-sum-exp operations, simplifying common machine learning operations like softmax.

References

[1] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, May 2015.

[2] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan, "Do deep generative models know what they don't know?" 2019. [Online]. Available: https://arxiv.org/abs/1810.09136

[3] G. Marcus, "The next decade in ai: four steps towards robust artificial intelligence," *arXiv preprint arXiv:2002.06177*, 2020.

[4] A. Heljakka, M. Trapp, J. Kannala, and A. Solin, "Disentangling model multiplicity in deep learning," *arXiv preprint arXiv: 2206.08890*, 2023.

[5] A. Nicolson and K. K. Paliwal, "Sum-product networks for robust automatic speaker identification," in *Proc. Interspeech 2020*, 2020, pp. 1516–1520.

[6] J. Wang and G. Wang, "Hierarchical spatial sum–product networks for action recognition in still images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 1, pp. 90–100, 2018.

[7] K. Stelzner, R. Peharz, and K. Kersting, "Faster attend-infer-repeat with tractable probabilistic models," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 09–15 Jun 2019, pp. 5966–5975.

[8] K. Zheng and A. Pronobis, "From pixels to buildings: End-to-end probabilistic deep networks for large-scale semantic mapping," 2019.

[9] R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. Trapp, G. Van den Broeck, K. Kersting, and Z. Ghahramani, "Einsum networks: Fast and scalable learning of tractable probabilistic circuits," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7563–7574.

[10] M. Kang, N. M. Gürel, L. Li, and B. Li, "Colep: Certifiably robust learning-reasoning conformal prediction via probabilistic circuits," *arXiv preprint arXiv:2403.11348*, 2024.

[11] J. Maene, V. Derkinderen, and P. Z. Dos Martires, "Klay: Accelerating arithmetic circuits for neurosymbolic ai," in *The Thirteenth International Conference on Learning Representations*, 2025.

[12] N. Shah, L. I. G. Olascoaga, S. Zhao, W. Meert, and M. Verhelst, "Dpu: Dag processing unit for irregular graphs with precision-scalable posit arithmetic in 28 nm," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 8, pp. 2586–2596, 2021.

[13] L. Sommer, L. Weber, M. Kumm, and A. Koch, "Comparison of arithmetic number formats for inference in sum-product networks on fpgas," in *2020 IEEE 28th Annual international symposium on field-programmable custom computing machines (FCCM)*. IEEE, 2020, pp. 75–83.

[14] R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. Trapp, K. Kersting, and Z. Ghahramani, "Probabilistic deep learning using random sum-product networks," *arXiv preprint arXiv:1806.01910*, 2018.

[15] R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, and Z. Ghahramani, "Random sum-product networks: A simple and effective approach to probabilistic deep learning," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 334–344.

[16] L. Sommer, J. Oppermann, A. Molina, C. Binnig, K. Kersting, and A. Koch, "Automatic mapping of the sum-product network inference problem to fpga-based accelerators," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 350–357.

[17] L. Yao, M. Trapp, J. Leslin, G. Singh, P. Zhang, K. Periasamy, and M. Andraud, "On hardware-efficient inference in probabilistic circuits," in *The 40th Conference on Uncertainty in Artificial Intelligence*, 2024.

[18] K. Periasamy, J. Leslin, A. Korsman, L. Yao, and M. Andraud, "Autopc: An open-source framework for efficient probabilistic reasoning on fpga hardware," in *2024 22nd IEEE Interregional NEWCAS Conference (NEWCAS)*. IEEE, 2024, pp. 21–25.

[19] M. Haselman, M. Beauchamp, A. Wood, S. Hauck, K. Underwood, and K. S. Hemmert, "A comparison of floating point and logarithmic number systems for fpgas," in *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*. IEEE, 2005, pp. 181–190.

[20] L. Weber, L. Sommer, J. Oppermann, A. Molina, K. Kersting, and A. Koch, "Resource-efficient logarithmic number scale arithmetic for spn inference on fpgas," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 251–254.

[21] 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.

[22] Y. Choi, "Probabilistic reasoning for fair and robust decision making," Ph.D. dissertation, University of California, Los Angeles, 2022.

[23] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, and L. D. Raedt, "Deepproblog: Neural probabilistic logic programming," 2018. [Online]. Available: https://arxiv.org/abs/1805.10872

[24] H. Poon and P. M. Domingos, "Sum-product networks: A new deep architecture," in *UAI*. AUAI Press, 2011, pp. 337–346.

[25] D. Kisa, G. V. den Broeck, A. Choi, and A. Darwiche, "Probabilistic sentential decision diagrams," in *Proc. KR*, 2014.

[26] T. Rahman, P. Kothalkar, and V. Gogate, "Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*. Springer, 2014, pp. 630–645.

[27] M. Trapp, R. Peharz, H. Ge, F. Pernkopf, and Z. Ghahramani, "Bayesian learning of sum-product networks," in *NeurIPS*, 2019, pp. 6344–6355.

[28] R. Gens and P. Domingos, "Learning the structure of sum-product networks," in *International Conference on Machine Learning*, 2013, pp. 873–880.

[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[30] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[31] S. Cheng, G. Shang, and L. Zhang, "Handwritten digit recognition based on improved vgg16 network," in *Tenth International Conference on Graphics and Image Processing (ICGIP 2018)*, vol. 11069. SPIE, 2019, pp. 954–962.

[32] N. Shah, L. I. G. Olascoaga, W. Meert, and M. Verhelst, "Problp: A framework for low-precision probabilistic inference," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[33] N. Shah, W. Meert, and M. Verhelst, "DPU-v2: Energy-efficient execution of irregular directed acyclic graphs," in *International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, oct 2022, pp. 1288–1307.

[34] W. Kahan, "Ieee standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.

[35] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing frontiers and innovations*, vol. 4, no. 2, pp. 71–86, 2017.

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

[37] H. Fu, O. Mencer, and W. Luk, "Comparing floating-point and logarithmic number representations for reconfigurable acceleration," in *2006 IEEE International Conference on Field Programmable Technology*. IEEE, 2006, pp. 337–340.

[38] C. Collange, J. Detrey, and F. de Dinechin, "Floating point or lns: Choosing the right arithmetic on an aapplication basis," in *9th EUROMICRO Conference on Digital System Design (DSD'06)*. IEEE, 2006, pp. 197–203.

[39] J. Detrey and F. de Dinechin, "A vhdl library of lns operators," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2. IEEE, 2003, pp. 2227–2231.

[40] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.

[41] A. Rooshenas and D. Lowd, "Learning sum-product networks with direct and indirect variable interactions," in *International Conference on Machine Learning (ICML)*. PMLR, 2014, pp. 710–718.

[42] M. K. Jaiswal and H. K.-H. So, "Architecture generator for type-3 unum posit adder/subtractor," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[43] C. Aliferis and G. Simon, "Overfitting, underfitting and general model overconfidence and under-performance pitfalls and best practices in machine learning and ai," *Artificial intelligence and machine learning in health care and medical sciences: Best practices and pitfalls*, pp. 477–524, 2024.

[44] R. Dorrance, D. Dasalukunte, H. Wang, R. Liu, and B. R. Carlton, "An energy-efficient bayesian neural network accelerator with cim and a time-interleaved hadamard digital grng using 22-nm finfet," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 10, pp. 2826–2838, 2023.

[45] R. Schweiger, Y. Erlich, and S. Carmi, "Factorialhmm: fast and exact inference in factorial hidden markov models," *Bioinformatics*, vol. 35, no. 12, pp. 2162–2164, 2019.

[46] X. Wu, A. Subramaniyan, Z. Wang, S. Narayanasamy, R. Das, and D. Blaauw, "A high-throughput pruning-based pair-hidden-markov-model hardware accelerator for next-generation dna sequencing," *IEEE Solid-State Circuits Letters*, vol. 4, pp. 31–35, 2021.

[47] Y. Choi, A. Vergari, and G. Van den Broeck, "Probabilistic circuits: A unifying framework for tractable probabilistic models," *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, p. 6, 2020.

[48] J. Kwisthout, "Approximate inference in bayesian networks: Parameterized complexity results," *International Journal of Approximate Reasoning*, vol. 93, pp. 119–131, 2018.

Lingyun Yao (Student Member, IEEE) is currently pursuing a Ph.D. degree at Aalto University, Finland, under the supervision of Prof. Martin Andraud. She received the B.Eng. degree in Microelectronics from Central South University, China, in 2020, and the M.Sc. degree in Micro- and nanoelectronic Circuit Design from Aalto University, Finland, in 2022. During her master's studies and research, she worked on alternative number systems for probabilistic circuits. From October 2024 to January 2025, she was a visiting scholar at MICAS, KU Leuven, Belgium, hosted by Prof. Marian Verhelst. Her research interests include the design of AI accelerators for energy-efficient probabilistic and neuro-symbolic models, and exploring ways to enhance the hardware efficiency of explainable AI.

Shirui Zhao (Student Member, IEEE) received his B.S. degree from Northwestern Polytechnical University (NWPU) in 2012 and M.S. degree from the University of Chinese Academy of Sciences (UCAS) in 2015, respectively. He is currently pursuing a Ph.D. degree at ESAT-MICAS, KU Leuven, with a focus on the area of low-power probabilistic reasoning hardware design. His research interests span machine learning, chip architecture, and low-power circuit design.

Martin Trapp is a Research Council of Finland-funded independent postdoctoral researcher at Aalto University, Finland, working on probabilistic machine learning and reliable artificial intelligence. Previously, he was a postdoctoral researcher at Aalto University from 2020 to 2022 under Arno Solin. He received his PhD in machine learning from Graz University of Technology, Austria, in 2020 under the supervision of Franz Pernkopf and Robert Peharz. Martin is an active member of the tractable probabilistic modelling and the Bayesian deep learning communities, a member of the Turing probabilistic programming language, and an active supporter of open-source and open-science practices.

Jelin Leslin (Student Member, IEEE) is a doctoral candidate at Aalto University, specializing in hardware-aware AI models with a focus on energy-efficient implementations via model compression, hardware-aware training, and custom hardware design. He earned his Master's in Electrical Engineering (2020) from TU Eindhoven and KTH, with a thesis on hardware acceleration for autonomous vehicle motion control at Volvo Trucks, Sweden. Currently, he is working with the QT group in Finland on deploying LLMs efficiently at the edge.

This article has been accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCSI.2025.3613975

Marian Verhelst (Fellow, IEEE) is a professor at the MICAS lab of KU Leuven and a research director at imec. Her research focuses on embedded machine learning, hardware accelerators, and low-power edge processing. She received a PhD from KU Leuven in 2008, and worked as a research scientist at Intel Labs from 2008 till 2010. Marian is a scientific advisor to multiple startups, member of the board of ECSA, and served in the board of directors of tinyML. She is a science communication enthusiast as an IEEE SSCS Distinguished Lecturer, as a regular member of the Nerdland science podcast (in Dutch), and as the founding mother of KU Leuven's InnovationLab high school program. Marian received the laureate prize of the Royal Academy of Belgium in 2016, the 2021 Intel Outstanding Researcher Award, and the André Mischke YAE Prize for Science and Policy in 2021.

Martin Andraud (Member, IEEE) is an assistant professor in microelectronics at UCLouvain, Belgium, and a visiting professor at Aalto University, Finland. He received his PhD from Grenoble University, France, in 2016. He was a postdoctoral researcher successively with TU Eindhoven in 2016 and KU Leuven from 2017 to 2019. Between 2019 and 2024, he was an assistant professor at Aalto University, Finland. His research interests include the interface between edge AI, hardware/software co-design, testing, and reliability of custom ASIC for various AI accelerators, for instance, mixed-signal Compute-In-Memory architectures and alternative to deep learning models (probabilistic reasoning or hybrid AI).