ALPS: Adaptive Quantization of Deep Neural Networks with GeneraLized PositS

Hamed F. Langroudi[§,†], Vedant Karia[§], Zachariah Carmichael[§], Abdullah Zyarah[§,†],
Tej Pandit[§], John L. Gustafson[*], Dhireesha Kudithipudi[§]
[§]Neuromorphic AI Lab, University of Texas at San Antonio, TX, USA
[†]Rochester Institute of Technology, NY, USA
[*]National University of Singapore, Singapore

Abstract

In this paper, a new adaptive quantization algorithm for generalized posit format is presented, to optimally represent the dynamic range and distribution of deep neural network parameters. Adaptation is achieved by minimizing the intra-layer posit quantization error with a compander. The efficacy of the proposed quantization algorithm is studied within a new low-precision framework, ALPS*, on ResNet-50 and EfficientNet models for classification tasks. Results assert that the accuracy and energy dissipation of low-precision DNNs using generalized posits outperform other well-known numerical formats, including standard posits.*

1. Introduction

There is an increasing demand to deploy deep neural networks (DNNs) on edge devices due to their pervasiveness in a wide range of domains, from healthcare [29] to cybersecurity [31]. For instance, deploying on edge requires $\sim$153.5$\times$ more memory and $\sim$4$\times$ more power than the available device budget when considering ARM M-7 microcontrollers [2]. To bridge this gap, one approach is to compress the footprint of DNNs by employing low-precision arithmetic [3, 8, 14, 22, 30, 36].

The benefits of low-precision DNNs in reducing hardware complexity comes at the cost of reduced accuracy associated with quantization error. The quantization error generated with post-training quantization approach heavily relies on the distribution and dynamic range of the numerical format chosen. To reduce the quantization error, a numerical format with unequal-magnitude spacing (tapered accuracy) and high dynamic range is needed. The posit numerical format [12] offers both of these characteristics, which are lacking in conventional formats, including fixed-point, floats, and block floating point [35]. Moreover, posit arithmetic in DNN inference has been demonstrated to retain accuracy at 8-bit precision [15, 21, 23].

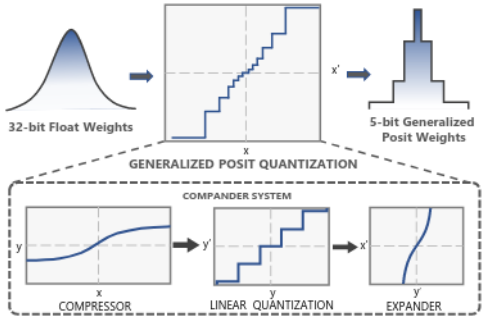

Figure 1: Generalized posit quantization model with a fixed-point quantizer, *compressor* $y{=}\frac{1}{\gamma}\sinh^{-1}(\theta x)$, and *expander* $x' = \frac{1}{\theta}\sinh((\gamma)y')$ functions. The α, β, γ, and θ are real variables that are varied for different generalized posit configurations

One limitation when working with fewer than 8-bits in posit is that it cannot accommodate the variability observed in inter- and intra-layer parameter distributions and dynamic range of DNNs. This leads to a catastrophic drop in accuracy [23]. One approach to mitigate this deficiency is to adapt mixed-precision posits for the parameters of each layer, which leads to exponential search space for optimal representation. For example, the search space size with four numerical format options n={8,6,4,2} for ResNet-110 is $4^{110}{\approx}1.68 \times 10^{66}$. Generalized posit format addresses this issue by adding the capability to parameterize the tapered accuracy and dynamic range.

Therefore, we are motivated to develop the ALPS framework to study the performance of DNN models with generalized posit numerical format. This framework adjusts the distribution and dynamic range of generalized posits and matches it with that of the DNN parameters for each layer. Adaptation is achieved by estimating the hyper-parameters in generalized posit and minimizing the quantization error (while maximizing the signal-to-quantization-noise ratio (SQNR)) in each layer. To formulate the correlation between the SQNR of posit and the performance of DNNs,

finite precision error analysis is used [13]. Finite precision error analysis is explored in prior studies for fixed-point and float [13, 19, 26]. The generalized posit quantization is modeled by a compressor function, expander function, and a fixed-point quantizer as shown in Figure 1, which is inspired by the quantization model of a compander system [4].

The key contributions of this work are as follows:

1. A new SQNR equation for generalized posit is proposed that uses a metric to select the appropriate generalized posit configuration.
2. A new low-precision framework, ALPS, is developed, that utilizes an adaptive quantization algorithm to enhance the performance of DNN inference with generalized posits.
3. A custom digital architecture is designed to analyze the energy cost of DNN inference with generalized posits and other numerical formats.

2. Background

Practical implementations of deep learning systems conventionally employ floating point arithmetic, usually with the widely adopted IEEE-754 format. However, this floating point system manifests arithmetic shortcomings, e.g., rounding, overflow/underflow, and lack of algebraic associativity, which are exacerbated when precision is low. To address these shortcomings, a new arithmetic number format was introduced, called *universal numbers* (unums) [11]. The latest rendition of unums with the most interest are type III unums that, also known as *posits*, which differ significantly from floats. Posits usually yield better arithmetic accuracy, dynamic range, and program reproducibility than IEEE-754 floats with the same bit-length [12]. The tapered-accuracy attribute of posits comes from the variable-length *regime* in their binary representation. Like IEEE floats, binary representations contain a sign bit, exponent bits, and a fraction. However, posits also have a regime that is encoded using a run-length r of `0`s or `1`s that is terminated by a `1` or `0`, respectively, or by the final bit. It represents $k = -r$ if the first regime bit is `0`, and $k = r - 1$ if the first bit is a `1`. After the regime, posits have an *es*-bit exponent e (unsigned integer value e) followed by the fraction bits $f, 0 \leq f < 1$. With a leading sign bit s, the real number represented by a posit is given by (1)

$$x = ((1 - 3s) + f) \times 2^{(1-2s)\times(2^{es}k+e+s)} \tag{1}$$

with special cases for 0 and not-a-real (NaR) excluded. Unlike the NaN values of floats, NaR includes the non-real values ∞ and $-\infty$. Too-large values round to the largest magnitude posit instead of overflowing to ∞. Too-small values round to the smallest magnitude posit instead of underflowing to 0. The sign of the too-large and too-small roundings is preserved.

A *generalized posit* parameterizes the exponent bias and maximum regime run-length. Exponent bias lets us shift the location of the zone of maximum accuracy. Restricting the regime length prevents the exponent and fraction bit fields from vanishing at the extremes of the dynamic range.

A parameter $sc \in [-\frac{n-2}{2}..\frac{n-2}{2}]$ biases the power-of-2 scaling downward or upward. With *sc*, the maximum and minimum magnitude values are given by $2^{2^{es}\times(n-2)+sc}$, and $2^{-2^{es}\times(n-2)+sc}$ respectively.

The other parameter is a maximum regime bit width $rs \in [1..n-1]$, which restricts the maximum and minimum positive representable values as given by (2) and (3), respectively, where $t = n - rs - 1$.

$$\text{Max}(x_{GP}) = \begin{cases} 2^{2^{es}\times(rs-2^{-t})}, & \text{if } (t \leq es) \\ 2^{2^{es}\times rs} \times (1 - 2^{es-t-1}), & \text{otherwise} \end{cases} \tag{2}$$

$$\text{Min}(x_{GP}) = \begin{cases} 2^{-2^{es}\times(rs-2^{-t})}, & \text{if } (t \leq es) \\ 2^{-2^{es}\times rs} \times (1 + 2^{es-t}), & \text{otherwise} \end{cases} \tag{3}$$

Notably, generalized posit formats encompass IEEE-like floats with $rs = 1$ (fixed fraction field), standard posits with $rs = n - 1$, and other tapered-precision formats between those bounds.

3. Related Work

Studies considering low-precision arithmetic have experimentally shown that DNNs using 8-bit numbers can achieve inference accuracy comparable to that of 32-bit numbers [36]. However, the performance of these models is degraded significantly when ultra-low-precision ($<$ 8-bit width) numbers are used [20]. To mitigate this problem, researchers have explored various ultra-low-precision or mixed-precision numerical formats [3, 8, 9, 15, 27, 30, 32, 33].

The posit format has been compared with other DNN inference formats [5, 6, 15, 18, 23]. In [18], authors compared low-precision posits and fixed-point with 32-bit high-precision floats for DNN inference on the AlexNet and ImageNet corpora. The outcome of this study indicates that 7-bit posit weight representation is sufficient to achieve inference accuracy within 1% variation of 32-bit floats. In [5, 6], authors introduced the Deep Positron accelerator with an FPGA-based soft-core for $\leqslant$8-bit precision posit exact-MAC operations. In this work, 8-bit posits are shown to have a better trade-off between inference energy consumption and inference performance in comparison to 8-bit fixed-point and float on various benchmarks. To reduce the hardware complexity of posit-based DNN inference, a logarithmic form of posit arithmetic is presented in [15]. On ImageNet dataset, 8-bit log-posit performed within 0.9% inference accuracy of 32-bit floats.

Following the success of 8-bit posits for DNN inference, the efficacy of this numerical format is studied for ultra-low

precision ([5..8]-bit) [16, 23]. However, none of these works capture the variability in inter- and intra-layer DNN parameter distributions and fail to preserve the accuracy when the number of bits is reduced to lower than 6 bits [16, 23]. Recently, in [17], authors presented a parameter-aware numerical format, Adaptive posit, as a plausible solution to capture the variability of inter- and intra-layer DNN parameters for different image classification tasks. However, determining the adaptive posit configuration for DNN inference depends on a large search space.

This research addresses the gap in posit format's ability to capture the variability in parameter distributions across DNN layers. A notable difference between this work and previous work [17, 23] is that the generalized posit quantization is adapted to the DNN parameter distribution without a brute force approach over a large search space. DNN computations are performed in low-precision generalized posit format and the associated energy and delay are studied.

4. Numerical Analysis of Generalized Posit Quantization Error

The goal of this section is to formalize the relationship between quantization error of high-precision float DNN parameters and low-precision generalized posits. We seek to derive a function $\mathcal{F}$ that approximates the misclassification probability, p_m, as given by (4)

$$p_m \approx \mathcal{F}\left(\epsilon(w_i, w_i'), \epsilon(A_i, A_i')\right) \tag{4}$$

where the function $\epsilon(\cdot,\cdot)$ determines the total quantization error between its two arguments, w_i and A_i represent floating point weights and activations, w_i' and A_i' represent generalized posit weights and activations respectively.

4.1. SQNR for Generalized Posits

Defining the SQNR for generalized posits requires modeling the quantization error $\epsilon(x_i, x_i') = \sum_{i=0}^{m} |x_i - x_i'|$ [34], where m represents the number of parameters in a DNN, x_i represents the 32-bit high-precision floating point DNN parameters, and x_i' indicates the q-bit low-precision generalized posit quantized DNN parameters. This model depends on the distribution of the values represented by this numerical format. Low-precision generalized posit has a non-uniform distribution, which is modeled as a non-uniform quantizer with a compander system [10] as shown in Figure 1. The compander system contains a monotonic smooth non-uniform *compressor* function, a fixed-point quantizer, and an inverse function of the *compressor* function called an *expander*. In this paper, the *compressor* and *expander* are approximated by (5) and (6), respectively,

$$\begin{aligned} y &= \sum_{i=0}^{m} \left(\tfrac{1}{\alpha}x + \text{sign}(x)\beta\Delta\right) \times \mathbb{1}_{[i\Delta,(i+1)\Delta)}(|x|) \\ &\approx \sum_{i=0}^{m} \tfrac{1}{\gamma}\sinh^{-1}(\theta x) \times \mathbb{1}_{[i\Delta,(i+1)\Delta)}(|x|) \end{aligned} \tag{5}$$

$$\begin{aligned} x' &= \sum_{j=0}^{m} \left(\alpha(y' - \text{sign}(y')\beta\Delta) \times \mathbb{1}_{[j\Delta,(j+1)\Delta)}(|y'|)\right. \\ &\approx \sum_{j=0}^{m} \tfrac{1}{\theta}\sinh^{-1}(\gamma y')\mathbb{1}_{[j\Delta,(j+1)\Delta)}(|y'|) \end{aligned} \tag{6}$$

where α, β, γ, and θ are real variables, Δ is the quantization step size, m represents the quantization levels, and $\mathbb{1}_{[a,b)}(|x|)$ is the indicator function as given by (7). Note that prior works model quantization error using this approach for non-uniform float formats [34].

$$\mathbb{1}_{[a,b)}(|x|) = \begin{cases} 1, & \text{if } (a \le |x| < b) \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

While modeling the quantization error for generalized posit, the SQNR is computed as in (8) where Δ_{GP} (quantization step size) is computed as (9). The derivation of (8) is provided in the Appendix.

$$\text{SQNR}_{GP}(dB) \approx (10.79 - 20\log(\gamma)) + 20\log(\Delta_{GP}^{-1}) \tag{8}$$

$$\Delta_{GP} = \begin{cases} 2^{-(2^{es}rs - 2^{es-(n-rs-1)}) + sc}, & \text{if } (n - rs \le es + 1) \\ 2^{-(2^{es}rs - es + (n-rs-1)) + sc}, & \text{otherwise} \end{cases} \tag{9}$$

The proposed SQNR captures the various posit parameters that affect the representable distribution of values. Therefore, the SQNR of generalized posits is varied based on the exponent bit-width (*es*), the maximum regime bit-width (*rs*), and the exponent bias (*sc*) parameters. The parameter *sc* shifts the peak of SQNR and dynamic range and *es* controls the dynamic range and width of max SQNR ($[2^{-2^{es}}, 2^{2^{es}}]$) interval. The parameter *rs* adjusts the shape of the SQNR distribution and controls the dynamic range.

4.2. Finite Precision Error Analysis

The output y of a DNN is produced by a sequence of operations, the predominant one being the multiply-accumulate (MAC) operation. This operation is given in (10) where quantization error is considered throughout the network with an activation function T [25].

$$\begin{aligned} y + \epsilon_y = &T_n(...(T_2(T_1(w_1 + \epsilon_{w_1}) \times A_1 + \epsilon_{A_1}) + \\ &\epsilon_{A_2}) \times (w_2 + \epsilon_{w_2})) \end{aligned} \tag{10}$$

For simplicity, we use the shorthand ϵ_x to represent the error $\epsilon(x, x')$ for some arbitrary x. For neural network models

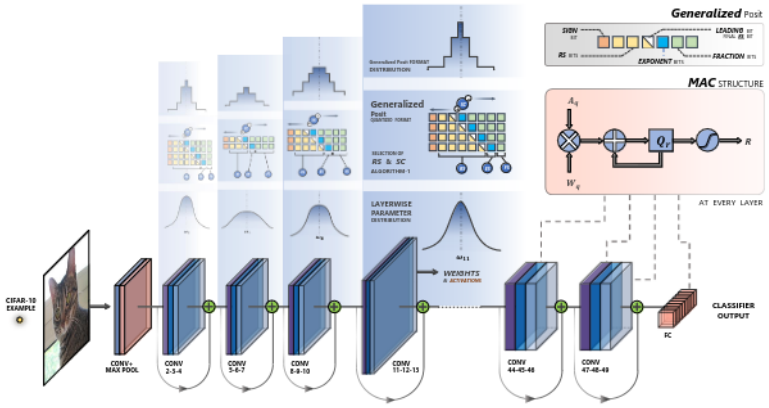

Figure 2: ALPS: a low-precision framework for DNN inference with generalized posit parameters. The framework selects the *rs* and *sc* parameters to match the layer wise parameter distribution. *rs* specifies the *maximum* number of regime bits, and *sc* specifies the degree of shift required (left-shift if positive, right-shift if negative). The multiply-and-accumulate (MAC) unit is presented in detail in Figure 3.

with a non-linear T_i, the error is calculated as in (11) using the chain rule of partial derivatives and approximated as a first-order Taylor series [13].

$$\epsilon_y \approx \sum_{i=1}^{n} \epsilon_{w_i} \frac{\partial T_i}{\partial w_i} + \sum_{i=1}^{n} \epsilon_{A_i} \frac{\partial T_i}{\partial A_i} \tag{11}$$

Finally, by calculating the probability ϵ_y for any y_i [19, 26], we can formulate the relationship between the mismatch probability and numerical precision as in (12), where E_{W_l} and E_{A_l} are weight and activation quantization error gains at layer l [19, 26].

$$p_m \leq \sum_{l=1}^{L} \frac{1}{\text{SQNR}_{w_l}} E_{W_l} + \frac{1}{\text{SQNR}_{A_l}} E_{A_l} \tag{12}$$

Given this equality, it helps selecting appropriate generalized posit numerical format configuration for a specified p_m.

5. ALPS framework

To emulate adaptive quantization with low-precision generalized posits in DNN inference, the ALPS framework is designed. Specifically, at each layer, the generalized posit parameters (*rs* and *sc*) that converge with the dynamic range and distribution of DNN parameters are selected, as shown in Figure 2. Following, the 32-bit floating point learned weights and activations are quantized to the low-precision generalized posit numerical format. Finally, the dot product of quantized weights is performed with low-precision generalized posit MAC structure. Therefore, the ALPS framework comprises three key aspects: generalized posit parameter selection, quantization with generalized posits, and the low-precision generalized posit dot product.

5.1. Generalized posit parameter selection

As mentioned in section 4, the performance of the DNN using the low-precision generalized posit depends on the p_m and SQNR as illustrated in (12). To improve the performance of DNN model with low-precision numerical format, the p_m requires to minimized, which can be accomplished by balancing the summation in the equation (12) [27]. This computes *rs* and *sc* from the equations (13), (14), (15), and (16) which are inspired from [27].

$$rs_{w_l} = \text{RNE}\left(\log_2 \sqrt{\frac{E_{w_l}}{E_{w_r}}}\right) + rs_{w_r} \tag{13}$$

$$rs_{A_l} = \text{RNE}\left(\log_2 \sqrt{\frac{E_{A_l}}{E_{w_r}}}\right) + rs_{w_r} \tag{14}$$

$$sc_{w_l} = \text{RNE}\left(\log_2 \sqrt{\frac{E_{w_l}}{E_{w_r}}}\right) + sc_{w_r} \tag{15}$$

$$sc_{A_l} = \text{RNE}\left(\log_2 \sqrt{\frac{E_{A_l}}{E_{w_r}}}\right) + sc_{w_r} \tag{16}$$

In these equations, RNE is the round-to-the-nearest-even function, E_{w_r} is the quantization error gain for weights of the reference layer (selected randomly), and Algorithm 1 presents the rs_{w_r} and sc_{w_r} optimization procedure. The selection of rs_{w_r} and sc_{w_r} is governed by the mean and excess kurtosis (Kurtosis-3) of the reference layer weights, which is computed in the initialization steps (lines 2–6).

To compute rs_{w_r}, the difference between the excess kurtosis of DNN weights and the excess kurtosis of generalized posit values with varied *rs* is computed and then the generalized posit numerical format configuration which has the closest excess kurtosis to that of the DNN parameters is

Algorithm 1 Compute the maximum regime bit width (*rs*) and scaling factor (*sc*) of generalized posit for reference parameter (W_r)
Input: reference layers weights (W_r)
Output: rs_{w_r}, rs_{A_r} generalized posit parameters

1: **procedure** *rs*, *sc* SELECTION (W_r)
2: $\mathtt{sc}_{\mathtt{r}_0} \leftarrow 0$
3: $\mathtt{rs}_{\mathtt{r}_0} \leftarrow \mathtt{n}-1$
4: $\mathtt{W}_{\mathtt{amax}} \leftarrow \max(|\mathtt{W}_\mathtt{r}|)$
5: $\mathtt{K}_\mathtt{W} \leftarrow \mathtt{Kurt}(\mathtt{W}_\mathtt{r}, -\mathtt{W}_{\mathtt{amax}}, \mathtt{W}_{\mathtt{amax}})$
6: $\mathtt{M}_\mathtt{W} \leftarrow \mathtt{mean}(\mathtt{W}_\mathtt{r})$

Compute the rs_{w_r}
7: $\mathtt{K}_{\mathtt{GP}_0} \leftarrow \mathtt{Kurt}(\mathtt{GP}(\mathtt{n},\mathtt{es},\mathtt{rs}_{\mathtt{r}_0},\mathtt{sc}_{\mathtt{w}_0}), -\mathtt{W}_{\mathtt{amax}}, \mathtt{W}_{\mathtt{amax}})$
8: $\mathtt{K\_diff}_0^{\mathtt{w}} \leftarrow |\mathtt{K}_\mathtt{w} - \mathtt{K}_{\mathtt{GP}_0}|$
9: **for** $i \leftarrow 3$ to $n-2$ **do**
10: $\mathtt{K}_{\mathtt{GP}_\mathtt{i}} \leftarrow \mathtt{Kurt}(\mathtt{GP}(\mathtt{n},\mathtt{es},\mathtt{i},\mathtt{sc}_{\mathtt{w}_\mathtt{r}}), -\mathtt{W}_{\mathtt{amax}}, \mathtt{W}_{\mathtt{amax}})$
11: $\mathtt{K\_diff}_\mathtt{i}^{\mathtt{w}} \leftarrow |\mathtt{K}_\mathtt{w} - \mathtt{K}_{\mathtt{GP}_\mathtt{i}}|$
12: **if** $\mathtt{K\_diff}_\mathtt{i}^{\mathtt{w}} < \mathtt{K\_diff}_0^{\mathtt{w}}$ **then**
13: $\mathtt{rs}_{\mathtt{w}_\mathtt{r}} \leftarrow \mathtt{i}$
14: $\mathtt{K\_diff}_0^{\mathtt{w}} \leftarrow \mathtt{K\_diff}_\mathtt{i}^{\mathtt{w}}$
15: **end if**
16: **end for**

Compute the sc_{w_r}
17: $\mathtt{M}_{\mathtt{GP}_0} \leftarrow \mathtt{mean}(\mathtt{GP}(\mathtt{n},\mathtt{es},\mathtt{rs}_{\mathtt{r}_\mathtt{w}},\mathtt{sc}_{\mathtt{r}_0}))$
18: $\mathtt{M\_diff}_0^{\mathtt{W}} \leftarrow |\mathtt{M}_\mathtt{W} - \mathtt{M}_{\mathtt{GP}_0}|$
19: **for** $i \leftarrow 1$ to 3 **do**
20: $\mathtt{M}_{\mathtt{GP}_\mathtt{i}} \leftarrow \mathtt{mean}(\mathtt{GP}(\mathtt{n},\mathtt{es},\mathtt{rs}_{\mathtt{r}_0},\mathtt{i}))$
21: $\mathtt{M\_diff}_\mathtt{i}^{\mathtt{W}} \leftarrow |\mathtt{M}_\mathtt{W} - \mathtt{M}_{\mathtt{GP}_\mathtt{i}}|$
22: **if** $\mathtt{M\_diff}_\mathtt{i}^{\mathtt{A}} < \mathtt{M\_diff}_0^{\mathtt{W}}$ **then**
23: $\mathtt{sc}_{\mathtt{W}_\mathtt{r}} \leftarrow -\mathtt{i}$
24: $\mathtt{M\_diff}_0^{\mathtt{W}} \leftarrow \mathtt{M\_diff}_\mathtt{i}^{\mathtt{W}}$
25: **end if**
26: **end for**
27: **end procedure**

selected (lines 7–16). A similar procedure is applied to compute sc_{w_r}, except the mean of the DNN weights is used as metric to select sc_{w_r}.

5.2. Quantization with Generalized Posits

The quantization function $Q(x_i, l, u, q)$ estimates each 32-bit floating-point DNN parameter x_i as x_i' (a q-bit generalized posit), as defined in (17). Given the dynamic range of a low-precision generalized posit format, the 32-bit high-precision float values that lie outside this dynamic range are clipped to the format minimum (l) and maximum (u) appropriately. The clipped values are then rounded to the A value that is between consecutive generalized posits is rounded to nearest number ($RNE(x_i)$).

$$x_i' = Q(x_i, q, l, u) = \mathrm{RNE}(\mathrm{clip}(x_i, l, u)) \qquad (17)$$

Algorithm 2 Generalized posit dot product operations for vector elements each with n bits, *es* exponent bits, *rs* with $\lfloor \log_2(n-1) \rfloor$ bit-width, *sc* with 3 bit-width.
Input: layers quantized weights (W_{l_q}), layers quantized activations (A_{l_q}),
Output: R as a dot product result

1: **procedure** GENERALIZED POSIT DP (W_{l_q}, A_{l_q})
2: $\mathtt{sign}_\mathtt{w}, \mathtt{reg}_\mathtt{w}, \mathtt{exp}_\mathtt{w}, \mathtt{frac}_\mathtt{w} \leftarrow \mathrm{DECODE}(W_{l_q}, rs_w)$
3: $\mathtt{sign}_\mathtt{a}, \mathtt{reg}_\mathtt{a}, \mathtt{exp}_\mathtt{a}, \mathtt{frac}_\mathtt{a} \leftarrow \mathrm{DECODE}(A_{l_q}, rs_a)$

Gather total scale factors
4: $\mathtt{sf}_\mathtt{w} \leftarrow 2^{\mathtt{es}\times\mathtt{reg}_\mathtt{w}} + \mathtt{exp}_\mathtt{w} + \mathtt{sc}_\mathtt{w}$
5: $\mathtt{sf}_\mathtt{a} \leftarrow 2^{\mathtt{es}\times\mathtt{reg}_\mathtt{a}} + \mathtt{exp}_\mathtt{a} + \mathtt{sc}_\mathtt{a}$

Multiplication
6: $\mathtt{sign}_{\mathtt{mult}} \leftarrow \mathtt{sign}_\mathtt{w} \oplus \mathtt{sign}_\mathtt{a}$
7: $\mathtt{frac}_{\mathtt{mult}} \leftarrow \mathtt{frac}_\mathtt{w} \times \mathtt{frac}_\mathtt{a}$
8: $\mathtt{normfrac}_{\mathtt{mult}} \leftarrow \mathtt{frac}_{\mathtt{mult}} \gg \mathtt{frac}_{\mathtt{mult}}[\mathtt{MSB}]$
9: $\mathtt{sf}_{\mathtt{mult}} \leftarrow \mathtt{sf}_\mathtt{w} + \mathtt{sf}_\mathtt{a} + \mathtt{frac}_{\mathtt{mult}}[\mathtt{MSB}]$
10: $\mathtt{p}_{\mathtt{mult}} \leftarrow (-1)^{\mathtt{sign}} \times 2^{\mathtt{sf}_{\mathtt{mult}}} \times (1 + \mathtt{frac}_{\mathtt{mult}})$

Accumulation
11: $\mathtt{fracs}_{\mathtt{mult}} \leftarrow \mathtt{sign}_{\mathtt{mult}} \; ? \; -\mathtt{frac}_{\mathtt{mult}} : \mathtt{frac}_{\mathtt{mult}}$
12: $\mathtt{sf}_{\mathtt{biased}} \leftarrow \mathtt{sf}_{\mathtt{mult}} + 2^{es+1} \times (n-2)$
13: $\mathtt{fracs}_{\mathtt{fixed}} \leftarrow \mathtt{fracs}_{\mathtt{mult}} \ll \mathtt{sf}_{\mathtt{biased}}$
14: $\mathtt{sum}_{\mathtt{quire}} \leftarrow \mathtt{fracs}_{\mathtt{fixed}} + \mathtt{sum}_{\mathtt{quire}}$

Rounding & Encode
15: $\mathtt{R} \leftarrow \mathrm{ROUNDING\ \&\ ENCODING}(\mathtt{sum}_{\mathtt{quire}})$
16: **return** R
17: **end procedure**

5.3. Low-precision Generalized Posit Dot Product

The generalized posit dot product is presented in Algorithm 2. In the first step, the set of quantized weights and activations are decoded to the generalized posit format and the scaling factor is computed (lines 2–5). Then, the product of the generalized posit weights and activations is calculated without truncation or rounding at the end of multiplications (lines 6–10). The products are then stored in a wide signed fixed-point register, the *quire* [12], for m multipliers with size $w_{quire} = \lceil \log_2(m) \rceil + 2 \times \lceil \log_2(\frac{\mathrm{Max}_{GP}}{\mathrm{Min}_{GP}}) \rceil + 2$ (lines 11–14). The stored products are then converted and accumulated using a fixed-point arithmetic. At the end, the accumulated result is converted back to the generalized posit numerical format (lines 15–17).

6. Hardware System Design and Architecture

The ALPS framework and theoretical analysis gives insights into adopting a new posit quantization method. This section describes the framework designed to simulate DNNs on hardware platforms and evaluate the performance of gen-

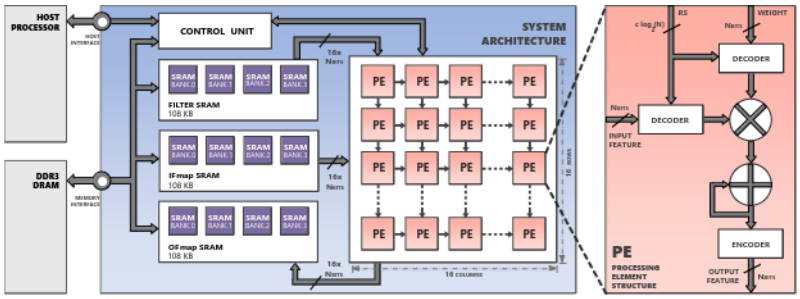

Figure 3: Deep neural network accelerator microarchitecture for the generalized posit format with customized processing elements. The architecture is evaluated in a full cycle-emulator to analyze the performance and energy constraints.

eralized posit representation in terms of latency and energy consumption. Figure 3 shows the high-level architecture of the designed framework guided by Eyeriss v2 [7] design. Primarily, it is composed of processing elements (PEs) arranged in a 2D systolic array architecture and a hierarchical memory organization. Most of the 2D-systolic architectures commonly used to perform convolution operations adopt input stationary (IS), weight stationary (WS), and output stationary (OS) dataflows. However, OS has shown reduced execution time and energy consumption since PE computations are confined to a single pixel in the output feature map. The PE in the systolic array has a generalized posit based MAC unit with configurable bit-precision. It is controlled by an external control signal (*rs*), which defines the exponent value (k value in (1)). For modeling memory hierarchy, 128 MB off-chip DRAM and 3×108 kB on-chip scratchpad memory (SRAM) are used. The DRAM is dedicated to storing input features and parameters that are loaded by the host processor, whereas the SRAM serves as a global buffer.

7. Experimental Setup, Results & Insights

The ALPS framework is implemented in C++ and extended to the TensorFlow framework [1]. To demonstrate the efficacy of the ALPS empirical framework, the performance of generalized posits with *rs* and *sc* is evaluated on two inference tasks and compared to both posits and scaled floats (the float numerical format with a scaling factor similar to sc) with two different DNN architectures. The specifications of the tasks and inference performance with 32-bit floats DNNs are summarized in Table 1. In the evaluation of each format, $es \in \{0, 1, 2\}$, $rs \in [1..n-1]$ and $sc \in [-3, 3]$ is considered for generalized posits, $es \in \{0, 1, 2\}$ is considered for posits, and $e \in \{3, 4\}$ is considered for scaled floats. To estimate latency, we bridge our framework with the SCALE-Sim tool [28]. SCALE-Sim, however, does not consider the cycles consumed by shuttling data back and forth between the global buffer and the DRAM. Therefore, the total latency is re-approximated by considering PE array execution time and DRAM access time (Micron MT41J256M4). For energy estimation analysis, the execution time, and power consumption, we factor in the 32-nm CMOS technology node.

7.1. DNN Inference Performance Using the ALPS Framework

The efficacy of the ALPS framework is evaluated for DNN inference using generalized posits with varied *es*, as shown in Tables 2 and 3. The findings show that the low-precision generalized posit (with es=1) outperforms the posit and scaled float formats with various DNNs by an average of 14% and 6%, respectively. For instance, the performance of a 5-bit low-precision generalized posit ResNet-50 network on the CIFAR-10 dataset is improved by 41.63% compared to the posit-based network. Both generalized posit and scaled float auto-adjust to the dynamic range of the weights and activations where quantization error is reduced, thus outperforming vanilla posits. Moreover, the performance of DNNs using 5-bit scaled floats is reduced significantly in comparison to generalized posit since it is not possible to represent 5-bit scaled floats with 4 exponent bits. In summary, the best performance on all the benchmarks (when analyzed across the full $[5..8]$-bit range) is achieved by utilizing generalized posits.

7.2. SQNR Impact on DNN Accuracy

As aforementioned in Section 4, the SQNR has a linear relationship with accuracy (as shown in Figure 4). The range of weights is mostly centered at zero and tapered to a dynamic range of 2 as mentioned in Table 1. Since posits have maximum SQNR in this range, the DNN accuracy using posits outperforms floats only if the weights are quantized. However, when both activations and weights are quantized, the dynamic range of activations changes across layers, which means that the SQNR of posits surpass floats, and in a few cases floats surpass posits. Therefore, it is valuable to have a format, such as generalized posits, where the SQNR is variable and can be matched to the variability of activations.

Table 1: The DNN models and benchmarks using 32-bit float parameters description.

Dataset	DNN Model	W-Range [1]	A-Range [1]	# Parameters	# MACs [2]	Performance
CIFAR-10	ResNet-50	$[-2.10, 2.29]$	$[0, 8.53]$	0.86 M	0.803 M	92.10%
	EfficientNet-B0	$[-2.23, 2.36]$	$[0, 8.96]$	4.0 M	3.12 M	98.00%
ImageNet	ResNet-50	$[-1.48, 2.62]$	$[0, 9.61]$	25 M	10.3 M	74.60%
	EfficientNet-B4	$[-1.72, 2.80]$	$[0, 10.2]$	19 M	10.5 M	83.00%

[1] W: Weights; A: Activations
[2] The number of MACs is calculated for a DNN inference with a batch size of 1.

Table 2: The DNN inference performance using the generalized posit, posit, and scaled float formats on CIFAR-10.

Dataset	Bit Precision	Generalized posit		Posit		Scaled float	
		RESNET-50	EfficientNet	RESNET-50	EfficientNet	RESNET-50	EfficientNet
CIFAR-10	8-bit	91.75%	97.37%	91.15%	96.89%	91.66%	96.91%
	7-bit	90.62%	92.91%	88.66%	91.27%	90.35%	92.03%
	6-bit	76.00%	70.64%	58.31%	60.09%	70.00%	66.59%
	5-bit	51.65%	53.66%	10.02%	10.00%	46.70%	47.20%

Table 3: The DNN inference performance using the generalized posit, posit, and scaled float formats on ImageNet.

Dataset	Bit Precision	Generalized posit		Posit		Scaled float	
		RESNET-50	EfficientNet	RESNET-50	EfficientNet	RESNET-50	EfficientNet
ImageNet	8-bit	74.11%	81.39%	73.61%	80.24%	74.06%	80.70%
	7-bit	72.46%	77.04%	69.10%	75.07%	70.76%	76.65%
	6-bit	63.76%	66.41%	53.46%	57.33%	62.31%	64.20%
	5-bit	46.22%	48.15%	0.10%	0.10%	10.00%	11.37%

7.3. Theoretical vs. Experimental Performance:

Figure 4(b) compares the theoretical misclassification upper bound with the misclassification rate that is obtained empirically in DNN inference with the ResNet and EfficientNet models on the ImageNet dataset using generalized posits. This theoretical bound depends on the SQNR of weights and activations and is given by (12). Since the SQNR of generalized posits is related to the precision, Δ_{GP}, the misclassification grows exponentially when the precision (Figure 4(b)). Overall, the theoretical bound is shown to approximate the misclassification rate in most cases.

7.4. Digital Architecture Results

The execution time of the DNN model is mainly governed by the dataflow and the PE array architecture. Output stationary dataflow has shown to offer 24% reduction in latency as compared to weight stationary dataflow in performing one inference. For a compute-bound DNN, this is a significant improvement, considering that inference favors latency over throughput [24]. The homogeneous 16×16 PE configuration offers improvement in computing efficiency from 89.58% to 91.82%, with a significant reduction in energy consumption. Figure 5 illustrates the energy-delay product (EDP) for ResNet-50 with posits while performing inference on the ImageNet dataset. It is worth noting that generalized posit offers in the range of 0.6% (n = 8) to 12% (n=6) improvement in classification accuracy with a negligible EDP overhead (6%) compared to posit. One may also observe from Figure 5 that lower *es* results in a greater reduction in energy consumption due to simpler encoding and decoding schemes. Reduced bit-precision economizes the local memory storage size and the number of operational cycles in both formats: generalized posits and posits.

8. Conclusions

Through the ALPS framework, we propose a numerical analysis of quantization error to discover the optimal generalized posit parameters. This allows us to adapt the quantization scheme with generalized posits to the distribution of DNN parameters. To accomplish this, we defined a novel SQNR formulation for generalized posits. This adaptive quantization approach yields an improvement in the average classification accuracy during inference by 14% and 6% over posits and scaled floats, respectively. Furthermore, we show

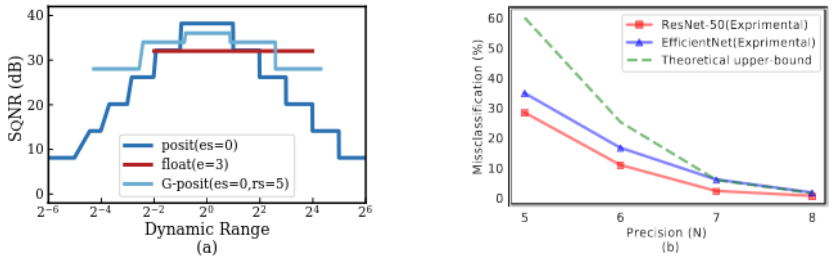

Figure 4: (a) The SQNR of 8-bit generalized posits compared to 8-bit posits and 8-bit floats. (b) ImageNet misclassification rate as a function of the generalized posit bit-precision with optimal *rs* for two DNNs and the theoretical upper bound (as formalized in (12)).

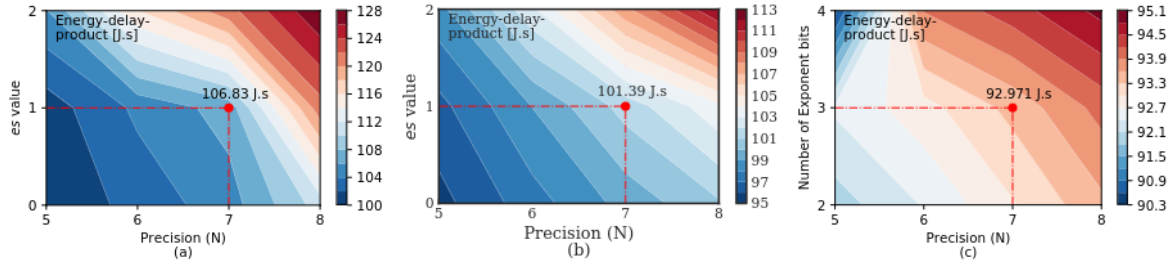

Figure 5: Energy-delay product of ResNet-50 benchmarked with ImageNet when using generalized posit (a), posit (b) and floating point(c). The performance of posits is evaluated for different bit widths, by changing number of exponent bits (*es*), to represent the weights and activations.

that generalized posits can achieve substantial performance improvements with a relatively moderate increase in energy consumption over posits.

9. Appendix: Derivation of Equation 8

The quantization error of the generalized posit numerical format $\epsilon_p(x_i, x_i') = x_i - x_i'$ is the difference between the 32-bit floats x_i input and x_i' as a q-bit generalized posit quantized output. This quantization error is approximated by fixed-point quantizer, compressor, and expander functions. Therefore, the quantization error of generalized posit is calculated in (18) where $\epsilon_{fx}(x_i, x_i')$ presents the fixed-point quantizer error, y' is the input of the expander function, x' is the output of the expander function computed in (19), Δ is quantization step size and γ is real number. Note that to derive the Equation 8, we follow [34].

$$\epsilon_p(x_i, x_i') = \epsilon_{fx}(x_i, x_i')\,\frac{dx'}{dy'} \tag{18}$$

$$x' = \frac{1}{\theta}\sinh(\gamma y') = \frac{1}{2\theta}\left(e^{(\gamma y')} - e^{-(\gamma y')}\right) \tag{19}$$

By obtaining the derivative ($\frac{dx'}{dy'}$) using (19) and replace result in (18), we have (20)

$$\epsilon_p(x_i, x_i') = \epsilon_{fx}(x_i, x_i')\frac{\gamma}{2\theta}\left(e^{(\gamma y')} + e^{-(\gamma y')}\right) \tag{20}$$

By using the approximation $\frac{\left(e^{(\gamma y')}+e^{-(\gamma y')}\right)}{2\theta} \approx \frac{\left(e^{(\gamma|y'|)}-e^{-(\gamma|y'|)}\right)}{2\theta}$ the $\epsilon_p(x_i, x_i')$ is correlated to x' as (21)

$$\begin{aligned}\epsilon_p(x_i, x_i') &= \epsilon_{fx}(x_i, x_i')\frac{\gamma}{2\theta}\left(e^{(\gamma y')} + e^{-(\gamma y')}\right)\\ &\approx \epsilon_{fx}(x_i, x_i')\gamma\,|x'|\end{aligned} \tag{21}$$

The generalized posit quantization error can be expressed in terms of x rather than x' as in (22) where the generalized posit quantization error $\epsilon_p(x_i, x_i')$ is much smaller than the inputs (and underflow and overflow does not occur).

$$\begin{aligned}\epsilon_p(x_i, x_i') &\approx \epsilon_{fx}(x_i, x_i')\gamma|x_i + \epsilon_p(x_i, x_i')|\\ &\approx \epsilon_{fx}(x_i, x_i')\gamma|x_i|\end{aligned} \tag{22}$$

From (22), the generalized posit SQNR can be expressed as (23) and (24). $\Delta_{GP}^2 = u^2$ and $\epsilon_{fx}^2(x_i, x_i') = \frac{u^2}{12}$ where u is the smallest value that can be represented by the generalized posit numerical format.

$$\text{SQNR}_{GP} \approx \frac{\mathbb{E}\{x^2\}}{\mathbb{E}\{\epsilon_p^2(x_i, x_i')\}} \approx \frac{12}{\gamma^2} \times (\Delta^{-1})^2 \tag{23}$$

$$\text{SQNR}_{GP}(dB) \approx (10.79 - 20\log(\gamma)) + 20\log(\Delta_{GP}^{-1}) \tag{24}$$

$$\Delta_{GP} = \begin{cases} 2^{-(2^{es}rs - 2^{es-(n-rs-1)})+sc}, & \text{if } (n - rs \leq es + 1) \\ 2^{-(2^{es}rs - es + (n-rs-1))+sc}, & \text{otherwise} \end{cases} \tag{25}$$

References

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakkar, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *arXiv preprint arXiv:2010.11267*, 2020.

[3] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, pages 7948–7956, 2019.

[4] William Ralph Bennett. Spectra of quantized signals. *The Bell System Technical Journal*, 27(3):446–472, 1948.

[5] Zachariah Carmichael, Hamed F. Langroudi, Char Khazanov, Jeffrey Lillie, et al. Performance-efficiency trade-off of low-precision numerical formats in deep neural networks. In *Proceedings of the Conference for Next Generation Arithmetic*, CoNGA'19, pages 3:1–3:9, Singapore, Singapore, 2019. ACM.

[6] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi. Deep positron: A deep neural network using posit number system. In *Design, Automation & Test in Europe (DATE) Conference & Exhibition*. IEEE, 2019.

[7] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[8] Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, et al. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. *Advances in Neural Information Processing Systems*, 33, 2020.

[9] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 293–302, 2019.

[10] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

[11] John L Gustafson. *The End of Error: Unum Computing*. Chapman & Hall/CRC Computational Science. Taylor & Francis, 2015.

[12] John L Gustafson and Isaac T Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, 2017.

[13] Jordan L Holi and J-N Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, 42(3):281–290, 1993.

[14] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[15] Jeff Johnson. Rethinking floating point for deep learning. *arXiv preprint arXiv:1811.01721*, 2018.

[16] Hamed F Langroudi, Zachariah Carmichael, David Pastuch, and Dhireesha Kudithipudi. Cheetah: Mixed low-precision hardware & software co-design framework for dnns on the edge. *arXiv preprint arXiv:1908.02386*, 2019.

[17] Hamed F Langroudi, Vedant Karia, John L Gustafson, and Dhireesha Kudithipudi. Adaptive posit: Parameter aware numerical format for deep learning inference on the edge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 726–727, 2020.

[18] S. H. Fatemi Langroudi, T. Pandit, and D. Kudithipudi. Deep learning inference on embedded devices: Fixed-point vs posit. In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 19–23, Mar. 2018.

[19] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[20] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[21] Raul Murillo, Alberto A Del Barrio, and Guillermo Botella. Deep pensieve: A deep learning framework based on the posit number system. *Digital Signal Processing*, page 102762, 2020.

[22] Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. *arXiv preprint arXiv:2004.10568*, 2020.

[23] Suresh Nambi, Salim Ullah, Aditya Lohana, Siva Satyendra Sahoo, Farhad Merchant, and Akash Kumar. Expan (n) d: Exploring posits for efficient artificial neural network design in fpga-based systems. *arXiv preprint arXiv:2010.12869*, 2020.

[24] David A Patterson. Latency lags bandwith. *Communications of the ACM*, 47(10):71–75, 2004.

[25] Stephen M Pizer. *To compute numerically: Concepts and strategies (Little, Brown computer systems series)*. Atlantic/Little, Brown, 1983.

[26] Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. Analytical guarantees on numerical precision of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3007–3016. JMLR. org, 2017.

[27] Charbel Sakr and Naresh Shanbhag. An analytical method to determine minimum per-layer precision of deep neural networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1090–1094. IEEE, 2018.

[28] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.

[29] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, pages 1–5, 2020.

[30] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, et al. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 4901–4910, 2019.

[31] Mariarosaria Taddeo, Tom McCutcheon, and Luciano Floridi. Trusting artificial intelligence in cybersecurity is a double-edged sword. *Nature Machine Intelligence*, pages 1–4, 2019.

[32] Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David Brooks, and Gu-Yeon Wei. Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[33] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Hardware-centric automl for mixed-precision quantization. *International Journal of Computer Vision*, pages 1–14, 2020.

[34] Bernard Widrow and István Kollár. *Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications*. Cambridge University Press, 2008.

[35] James Hardy Wilkinson. Rounding errors in algebraic processes. In *IFIP Congress*, pages 44–53, 1959.

[36] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.