12-2023

Tapered-Precision Numerical Formats for Deep Learning Inference and Training

Seyed Hamed Fatemi Langroudi
sf3052@rit.edu

Tapered-Precision Numerical Formats for Deep Learning Inference and Training

by

Seyed Hamed Fatemi Langroudi

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in Electrical and Computer Engineering

Department of Electrical and Computer Engineering
Kate Gleason College of Engineering

Rochester Institute of Technology
Rochester, New York

Dec 2023

Dissertation title: Tapered-Precision Numerical Formats for Deep Learning Inference and Training
By
Seyed Hamed Fatemi Langroudi

Committee Approval: We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Electrical and Computer Engineering.

Dr. Dhireesha Kudithipudi, Advisor Professor, Department of Computer Engineering, RIT	Date
Dr. Andres Kwasinski , Committee Member Professor, Department of Computer Engineering, RIT	Date
Dr. Majid Rabbani, Committee Member Professor, Department of Electrical and Microelectronic Engineering, RIT	Date
Dr. Christopher Kanan, Committee Member Associate Professor, Department of Computer Science, UR	Date

Certified By:

Dr. Andres Kwasinski , Electrical and Computer Engineering PhD Director, RIT	Date

ABSTRACT

Degree: Doctor of Philosophy
Author's name: Seyed Hamed Fatemi Langroudi
Advisor's name: Dr. Dhireesha Kudithipudi
Dissertation title: Tapered-Precision Numerical Formats for Deep Learning Inference and Training

The demand to deploy deep learning models on edge devices has recently increased due to their pervasiveness, in applications ranging from healthcare to precision agriculture. However, a major challenge with current deep learning models, is their computational complexity. One approach to address this limitation is to compress the deep learning models by employing low-precision numerical formats. Such low-precision models often suffer from degraded inference or training accuracy. This lends itself to the question, **which low-precision numerical format can meet the objective of high training accuracy with minimal resources**?

This research introduces tapered-precision numerical formats for deep learning inference and training. These formats have inherent capability to match the distribution of deep learning parameters by expressing values in unequal-magnitude spacing such that the density of values is maximum near zero and is tapered towards the maximum representable number. We develop low-precision arithmetic frameworks, that utilize tapered precision numerical formats to enhance the performance of deep learning inference and training. Further, we develop a software/hardware co-design framework to identify the right format for inference based on user-defined constraints through integer linear programming optimization. Third, novel adaptive low-precision algorithms are proposed that match the tapered-precision numerical format configuration to best represent the layerwise dynamic range and distribution of parameters within a deep learning model. Finally, a numerical analysis approach and signal-to-quantization-noise ratio equation for tapered-precision numerical formats are proposed that uses a metric to select the appropriate numerical format configuration.

The efficacy of the proposed approaches is demonstrated on various benchmarks. Results assert that the accuracy and hardware cost trade-off of low-precision deep neural networks using tapered precision numerical formats outperform other well-known numerical formats, including floating point and fixed-point.

This thesis is dedicated to my beloved wife and respected parents

for their endless love, support, and encouragement.

This achievement is as much yours as it is mine.

Acknowledgments

First and foremost, I would like to express my deep and sincere gratitude to my adviser, Dr. Dhireesha Kudithipudi, for her continuous support of my PhD research, helpful advice, providing motivation, and invaluable feedback. I offer my sincere appreciation to the committee members: Dr. Majid Rabani, Dr. Andres Kwasinski, Dr. Christopher Kanan, and Dr. Ernest Fokoue, for their insightful comments, and constructive advice.

I would also like to thank RIT faculty and staff, specifically Dr. Edward Hensel, Dr. Ray Ptucha, Dr. Cory Merkel, Mr. Mark Indovina, Mr. Richard Tolleson, and Rebecca Ziebarth, for their tremendous help and support. The PhD adventure would be much more challenging if it were not for their insight and excellent assistance. I am very grateful and fortunate to have had the opportunity to work with outstanding collaborators.

Many thanks go to Dr. John Gustafson (Singapore University of Technology), who is the inventor of posit, generalized posit, and tapered fixed-point (taper) and has provided valuable insights over the years. I am extending my heartfelt thanks to all the members of the Neuromorphic AI lab: Abdullah Zeyarah, Zachariah Carmichael, Tej Pandit, Vedant Karia, Nicholas Soures, Anurag Daram, Humza Syed, Fatima Tuz Zohora, and Peter Helfer. Also to my friends: Masoud Abdullahi, Kamran Binaee, Behrooz Mansouri, and Celal Savur. I am delighted to be working with you guys, who are knowledgeable but also enthusiastic, and friendly. I want to express my gratitude to all of you for supporting me in every possible way.

I would like to express my heartfelt gratitude to my wife, Tahereh, whose unwavering support and encouragement have been a source of strength and motivation throughout my PhD journey. Her understanding and patience have sustained me during challenging times, and her belief in me has pushed me to strive for excellence.

Finally, I extend my gratitude and sincerest thanks to my parents. I could not have undertaken this journey without their exceptional support, kindness, and patience. Your belief in me has kept my enthusiasm and motivation high throughout this process.

Contents

List of Tables

List of Figures

1. Introduction

Over the course of the last decade, deep neural networks (DNNs) have achieved state-of-the-art accuracy in a wide spectrum of applications such as biomedicine [9], precision agriculture [10], nature conservation [11], cybersecurity [12], and space computing [13]. For instance, Evans *et al.* proposed a new DNN model to perform protein folding tasks to diagnose and treat diseases such as Alzheimer's, Parkinson's, and Cystic Fibrosis [14]. Another example is Waldmann *et al.*, who introduced new DNN models for mapping out Saturn's turbulent storms to reveal deeper insight into the planet, such as the presence of ammonia ice clouds [15].

The success of DNNs can be attributed to their ability to hierarchically learn from huge amounts of raw and unstructured training data, where conventional machine learning algorithms have gaps [16]. Another contributing factor to DNNs' success is the significant enhancement in the knowledge capacity of DNN models. For instance, the knowledge capacity of DNNs for language translation has increased by 1942x from the GNMT model (278 million parameters) [17] to the recent PaLM model (540 billion parameters) [18] within a 6-year timespan. Increasing the knowledge capacity of DNN models also increases the number of

floating point operations (FLOPS), ranging from quintillion of floating point operations (ExaFLOPS) to septillion of floating point operations (YottaFLOPS) per DNN training and billions of floating point operations (GigaFLOPS) to trillions of operations (TeraFLOPS) per DNN inference [19]. However, DNN inference and training on these models demand substantial computational resources and high-bandwidth memory, leading to increased latency and power dissipation in performing machine learning tasks. For instance, training the GPT-3 language model costs 4.6 million [20] and consumes energy of 1287 MWh [21] (more than the average monthly electricity usage of U.S. residents in 2021 [22]). As another example, the state-of-the-art EfficientNet-B5 [23] to perform classification on the ImageNet dataset [24] requires 1024 TPU-v3 to train the network in one hour and four minutes and consumes approximately 325 KWh energy [25]. Inference performance on the same model on a V100 GPU is 60 milliseconds and approximately 550 mWh energy consumption per image [26].

Currently, the majority of DNN models are trained and deployed on the cloud to accommodate the large amounts of computations and high-bandwidth memory requirements. These DNN models are classified as CloudML models. However, the computational paradigm is shifting from cloud to edge computing that offers intelligence-at-the-edge of the mobile and sensor networks, known as Edge-AI [27]. It is expected that the Edge-AI hardware market is going to grow 17.9% in the next five years [28], which will deliver $51.6 billion in revenue by 2025, 3.5X larger than cloud revenue [29]. Therefore, a new class of DNN models (MobileML [30, 31] and TinyML [32, 33]) have emerged to address the surging demand for deploying

DNNs on edge devices.

Although Edge-AI will likely play a pivotal role in tackling global challenges and bringing considerable benefits to people's lives, it faces some key challenges, such as a massive gap between compute resources, memory storage, and energy budget required to train and deploy of DNN models and the available hardware resources on edge devices (as shown by Figure 1.1) [19]. The apparent solution to address this gap is compressing the network size and alleviating the computation requirements to match putative edge resources. Several groups have proposed the compressed DNN models with new compute- and memory-efficient neural networks [30, 33] and parameter-efficient neural networks, such as DNN pruning [34], distillation [35], low-rank approximation [36], and low-precision arithmetic [37–45]. Among these approaches, low-precision arithmetic is noted for its ability to reduce memory capacity, bandwidth, latency, and energy consumption associated with multiply and accumulation (MAC) units in DNNs, and increase the level of data parallelism [37, 46, 47]. However, the benefit of using low-precision arithmetic to reduce the complexity of a model comes at the cost of degraded inference or training accuracy [48]. This trade-off lends itself to the question: **which low-precision approach can meet the dual objectives of high inference and training accuracy and minimal hardware resources**? The answer to this question heavily depends on the choice of numerical formats for representing the parameters. A viable solution to this question could lead to significant energy savings in application domains such as autonomous vehicles, smart cities, and healthcare. Therefore, the impact of finding the appropriate numerical format for

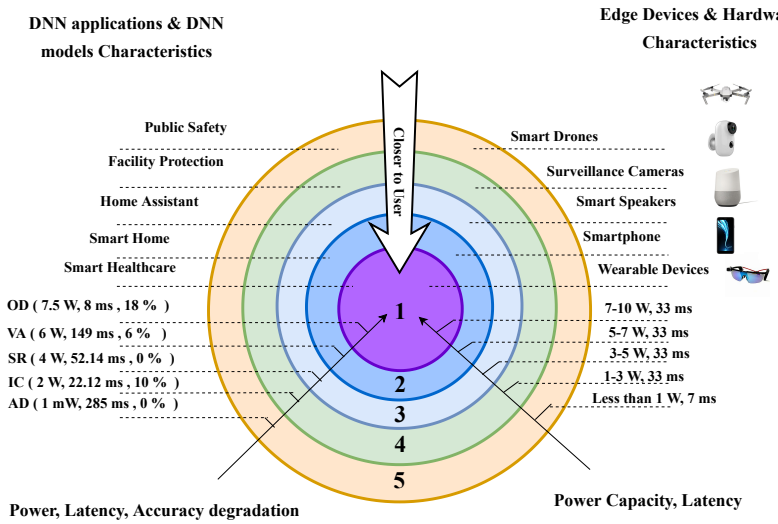

Figure 1.1: The gap between DNN model complexity and edge device resources. OD: Object Detection [1], VA: Video Analysis [2], SR: Speech Recognition [3], IC: Image Classification [4], AD: Activity Detection [5].

low-precision arithmetic is quite high, prompting organizations like IEEE to study this topic and develop a **standard for arithmetic formats for machine learning (P3109)** by 2025 [49].

This dissertation aims to address the aforementioned question through three main aspects. First, frameworks and low-precision approaches are developed to evaluate the efficacy of tapered precision numerical formats as our recommended numerical formats for DNN models. Note that the recent literature in tapered precision, published after our paper on posit numerical format for deep learning in 2018 [50], lacks either support for DNN training with the tapered precision numerical format, or accommodation of the variability in DNN parameter distributions and dynamic ranges. These gaps are addressed in this study by performing DNN training and inference with adaptive tapered precision numerical formats such as

generalized posit and tapered fixed point.

Second, the automated hardware-software co-design framework is developed to identify appropriate numerical format based on the performance and hardware complexity constraints defined by users.

Third, the benefit of tapered precision numerical formats for low-precision arithmetic is justified through the numerical analysis approach.

1.1 Motivation

To understand the correlation between hardware complexity (e.g., energy efficiency) and performance of low-precision neural networks for the edge, a hardware and software co-design framework is required. Previous studies have addressed this by proposing low-precision frameworks [37, 47, 51–58]. However, the scope of these studies is limited, as listed below. This research aims to address these gaps through a comprehensive analysis of tapered-precision hardware-software co-design.

1. **All of the previous studies only explore the efficiency of mixed low-precision floating point and fixed-point numerical formats for both DNN training and inference [37,47,51,52]**. Recently, the posit numerical format has been proposed as an alternative to IEEE-745 floating point format [6]. The posit format provides a higher dynamic range, better decimal accuracy for the ranges around zero, and balanced magnitude over floating point. These advantages are gained by altering the characteristics of the IEEE-745 floating point standard. For instance, only two bits are associated in the

32-bit posit numerical format to represent zero, $\pm$ infinity, and not a real number (NaR) in comparison to $(2^{23}+1)$-bit with 32-bit floating point numerical format. Additionally, the posit numerical format supports tapered precision for full dynamic range while the floating point numerical format supports tapered precision only for subnormal numbers. The benefits of posit numerical formats make it a suitable numerical format for DNN applications. The non-linear distribution of DNN parameters is matched to tapered precision of posit values. This feature significantly reduces the rounding error in comparison to fixed-point number system and floating point numerical format, as shown in Figure 1.2.

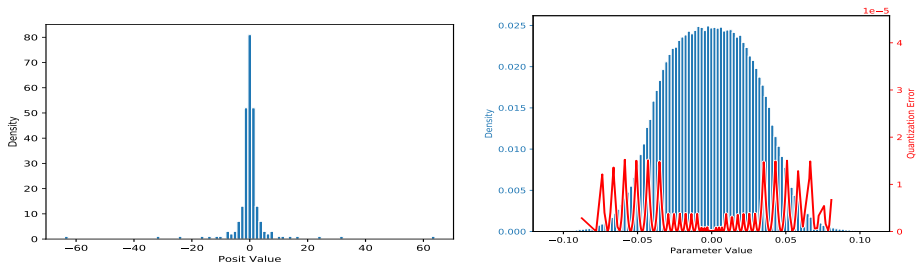

Figure 1.2: (a) 8-bit posit ($es = 0$) value distribution; (b) ResNet parameter distribution overlaid with quantization error (squared error). Both exhibit high density in the [-0.1,+0.1] range.

2. **In most of the previous studies, the hardware complexity comparison across various low-precision approaches has been conducted only across numerical formats. The underlying hardware overhead for quantization (*e.g.* complex vector quantization with k-means clustering) and pre-processing approaches are disregarded [59]. Additionally, in some**

cases, the comparison across numerical formats has been performed for varying bit-widths (*e.g.* 32-bit floating point compared to 8-bit fixed-point [46]). These unfair comparisons do not offer insights into the viability of each low-precision approach over other low-precision approaches for a particular task.

3. **Despite empirical comparison between performance of numerical formats in low-precision DNN models, little work has been devoted to theoretically explain why a specific low-precision approach works better than other low-precision arithmetic techniques.** Realizing theoretical bounds on misclassification rate to perform DNN inference with low-precision fixed-point parameters [60] and providing the dimension-free theoretical bounds on the convergence rate of low-precision fixed-point and floating point training with stochastic gradient descent (SGD) [53] are few of the missing details in these studies.

1.2 Contributions and thesis outline

The primary aim of this research is to understand the correlation between dual objective DNN inference and training performance (accuracy and hardware complexity) and associated tapered-precision numerical format. This correlation is evaluated either through empirical or numerical analysis approaches. The proposed frameworks and algorithms are applied in a wide range of neural networks such as convolutional neural networks, feedforward neural networks, recurrent neural

networks, and transformers, and a broad range of applications such as image classification, video activity classification, keyword spotting, and natural language processing. The key contributions of this research are:

- Low-precision arithmetic frameworks are developed, that utilize tapered precision numerical formats to enhance the performance of DNN inference and training.
- A hardware-software co-design framework which selects an appropriate numerical format for deep learning inference based on custom user-defined constraints through integer linear programming optimization.
- Novel adaptive quantization algorithms that match the tapered-precision numerical format configuration to best represent the layerwise dynamic range and distribution of parameters within a DNN model.
- A signal-to-quantization-noise ratio (SQNR) equation for generalized posit numerical format is proposed that uses a metric to select the appropriate generalized posit configuration. To compute the SQNR equation, the generalized posit quantization is modeled by a compressor function, expander function, and a fixed-point quantizer.

The rest of this dissertation is organized as follows: Chapter 2 discusses the overview of DNN inference and training for popular DNN models. It also explains three tapered-precision numerical formats as appropriate candidates that can satisfy most to all characteristics of numerical formats for DNN. Chapter 3 surveys the state-of-the-art numerical formats for DNN inference and training. Moreover, the approaches to numerically analyze the correlation between the DNN

performance accuracy and low-precision numerical format are discussed. Chapter 4 provides details about the Cheetah-V2 framework that (i) evaluates tapered-precision numerical formats for deep learning inference, (ii) empirically studies the correlation between hardware complexity and inference accuracy of tapered-precision deep learning models, (iii) selects the appropriate tapered-precision numerical format depending on accuracy and hardware complexity. Moreover, the novel ALPS 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 SQNR) in each layer. Chapter 5 demonstrates the efficacy of tapered precision numerical formats using Cheetah-V2 and ALPS frameworks on image classification, video activity detection, natural language processing, and keyboard spotting (Chapters 4 and 5 contents are based on Langroudi et. al [50, 61–66]). Chapter 6 provides the details about training DNN models using tapered-precision numerical formats through Jaapi framework. This framework provides a statistical and numerical analysis algorithm that adapts the tapered-precision numerical formats (mostly focus on asymmetric generalized posit) to best represent the layerwise dynamic range and distribution of parameters within layers and training epochs (contents of Chapter 6 are partially based on [67]). Chapter 7 describes the use case of Cheetah-V2 framework for surveillance video analysis as a case study. In this study, crime-related behaviors are detected through automatic analysis of surveillance video for various edge devices. The dissertation conclusions and future work are presented in Chapter 8.

2. Background

2.1 Deep neural network

Deep neural networks (DNNs) are artificial neural networks that are used for various tasks, such as classification, regression, and prediction, by learning the correlation between examples from training sets [68, 69]. These networks are capable of learning a non-linear input-to-output mapping in either a supervised, unsupervised, or semi-supervised manner. For instance, in a supervised learning scenario, to determine the correctness of classification in DNNs inference, the average error between Y_i and the desired output $\hat{Y}_i$ is calculated as $\mathbb{E}(e_i)$ by using a cost function L. To train DNNs, the weights are learned through mini-batch stochastic gradient descent using the backpropagation algorithm to minimize $\mathbb{E}(e_i)$ as given by Equations (2.1) and (2.2) [70] where α is a learning rate, m is a batch size, θ' is derivative of activation function and $[W, A, \nabla_W, \nabla_A, \nabla_Y]$ indicate weights, activations, weight gradients, activation gradients, and error gradients respectively.

$$\Delta W = -\alpha \nabla_W \big(\mathbb{E}(e_i)\big) = -\alpha \nabla_W \big(\frac{1}{m}\sum_{i=1}^{m} L(Y_i(A,W),\hat{Y}_i)\big) \tag{2.1}$$

$$\begin{cases} \nabla_{Y_n}\big(\mathbb{E}(e_i)\big) = \theta'(Y_n)\nabla_{A_n}\big(\mathbb{E}(e_i)\big) \\ \nabla_{W_n}\big(\mathbb{E}(e_i)\big) = A_{n-1}\nabla_{Y_n}\big(\mathbb{E}(e_i)\big) \\ \nabla_{A_{n-1}}\big(\mathbb{E}(e_i)\big) = \mathbf{w}^{\intercal}\nabla_{Y_n}\big(\mathbb{E}(e_i)\big) \end{cases} \tag{2.2}$$

To learn from different data, different DNN models have emerged [71–76]. These DNN models share a common characteristic, such as containing a sequence of layers with a set of nodes in each. However, the connectivity between layers is varied in DNN models. For instance, the layers are globally connected in feedforward neural networks, locally connected in convolutional neural networks and transformers, and recurrent connections in recurrent neural networks. Moreover, the computation in each node also varies in each DNN model, as summarized in Figure 2.1, where a major computation in these DNN models is the (multiply-and-accumulate) MAC operation. In this section, the most common DNN models are explained based on the computation graph executed on their node as (i) feedforward and convolutional neural network, (ii) recurrent neural network (iii) transformer.

2.1.1 Feedforward and convolutional neural networks

Feedforward neural networks are an appropriate choice for applications with scalar data, such as recommendation systems. On the other hand, convolutional neural networks are naturally suited for applications with multidimensional and structural data such as image classification. In particular, a node in feedforward neural and convolutional neural networks computes Equation (2.3) where B_j indicates the bias

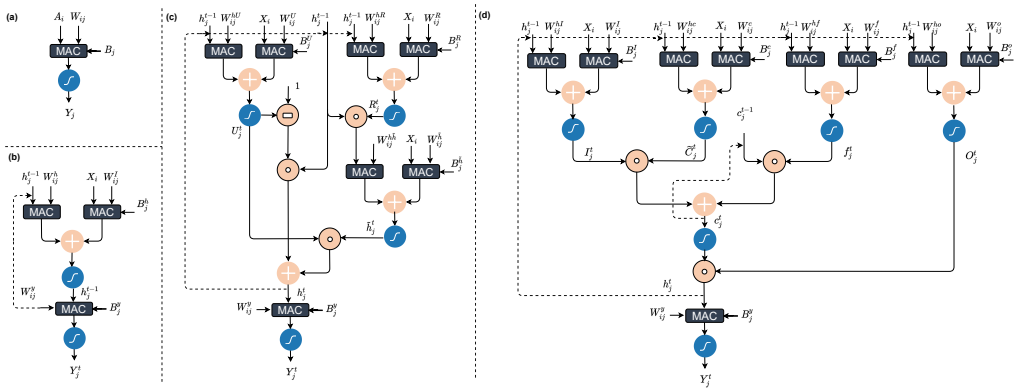

Figure 2.1: Computation-Graph executed on a node in a layer in various neural network topology: (a) Feedforward neural network, (b) Convolutional neural network, (c) Gated recurrent unit (GRU), (d) Long short-term memory (LSTM)

vector, W_{ij} is the weight tensor with numerical values that are associated with each connection, A_i represents the activation vector as input values to each node, θ is the activation function, Y_j is the feature vector as an output of each node, and N equals either the number of nodes for feedforward neural networks or the product of (C,R,S) filter parameters: the number of filter channels, the filter heights, and the filter widths respectively for convolutional neural networks.

$$Y_j = \theta(B_j + \sum_{i=0}^{N} A_i \times W_{ij}) \tag{2.3}$$

2.1.2 Recurrent neural network

The convolutional neural networks and feedforward neural networks may not be appropriate DNN models to learn a task from temporal information since the memory mechanisms are not designed in these networks. The recurrent neural

network connections act as a memory to retrieve information learned from previous time steps. In recurrent neural networks (RNNs), the operations computed by each node depend on recurrent topology (Vanilla RNN [73], Long Short-Term Memory (LSTM) [74], Gated Recurrent Units (GRU) [75]).

A node in Vanilla RNN computes Equation (2.4) where W_{ij}^{k} and W_{ij}^{h} indicate the weight tensors, Y_j^{t-1} represents the output in the previous time step, and N_1 depicts the number of time steps and N_2 represents the number of nodes.

$$Y_j^t = B_j + \sum_{i=0}^{N_1} Y_j^{t-1} \times W_{ij}^{h} + \sum_{i=0}^{N_2} A_i \times W_{ij}^{k} \tag{2.4}$$

However, performing DNN training and inference using the Vanilla RNNs has limitations, such as a lack of capability to learn long-term dependency and training divergence due to vanishing or exploding gradients. The LSTM architecture has been designed to address these limitations by adding memory units [74]. A node in LSTM computes Equation (2.5) where X_i is the input vector, h_j^{t-1} represents hidden state in the previous time step, $[W_{ij}^{hI}, W_{ij}^{I}, W_{ij}^{hf}, W_{ij}^{f}, W_{ij}^{ho}, W_{ij}^{o}, W_{ij}^{hc}, W_{ij}^{c}]$ are a set of weight tensors, $[B_j^h, B_j^f, B_j^o, B_j^c, B_j^y]$ represent bias vectors, [θ, Γ, Φ] are activation functions, $\odot$ represents element-wise multiplication, and $[I_j^t, f_j^t, O_j^t, \tilde{C}_j^t, c_j^t, h_j^t, Y_j^t]$ indicate input gate, forget gate, output gate, candidate for cell gate, cell state,

hidden state, and output respectively.

$$\begin{cases} I_j^t = \theta(B_j^h + \sum_{i=0}^{N_1} h_j^{t-1} \times W_{ij}^{hI} + \sum_{i=0}^{N_2} X_i \times W_{ij}^I), \\ f_j^t = \theta(B_j^f + \sum_{i=0}^{N_1} h_j^{t-1} \times W_{ij}^{hf} + \sum_{i=0}^{N_2} X_i \times W_{ij}^f), \\ O_j^t = \theta(B_j^o + \sum_{i=0}^{N_1} h_j^{t-1} \times W_{ij}^{ho} + \sum_{i=0}^{N_2} X_i \times W_{ij}^o), \\ \tilde{C}_j^t = \Gamma(B_j^c + \sum_{i=0}^{N_1} h_j^{t-1} \times W_{ij}^{hc} + \sum_{i=0}^{N_2} X_i \times W_{ij}^c), \\ c_j^t = f_j^t \bigodot c_j^{t-1} + I_j^t \bigodot \tilde{C}_j^t \\ h_j^t = O_j^t \bigodot \Gamma(c_j^t) \\ Y_j^t = \Phi(B_j^y + \sum_{i=0}^{N_3} h_j^t \times W_{ij}^y) \end{cases} \tag{2.5}$$

LSTMs are able to predict significantly longer sequences as compared to Vanilla RNNs, but have considerable overheads caused due to a multitude of computations and parameters [77]. To reduce the complexity of LSTMs, the GRU was proposed [75]. The forgetting and input units of the LSTM architecture are combined together in GRU and called the update unit. A node in GRU calculates Equation (2.6) where X_i is the input vector, h_j^{t-1} represents the hidden state in the previous time step, $[W_{ij}^{hR}, W_{ij}^R, W_{ij}^{hU}, W_{ij}^U, W_{ij}^{h\tilde{h}}, W_{ij}^{\tilde{h}}, W_{ij}^y]$ are the set of weight tensors, $[B_j^h, B_j^f, B_j^c]$ represent bias vectors, $\bigodot$ represents element-wise multiplication, [θ, Γ, Φ] are activation functions, and $[R_j^t, U_j^t, \tilde{h}_j^t, h_j^t, Y_j^t]$ are reset gate, update gate,

activation vector candidate, activation vector, and output respectively.

$$\begin{cases} R_j^t = \theta(B_j^R + \sum_{i=0}^{N_1} h_j^{t-1} \times W_{ij}^{hR} + \sum_{i=0}^{N_2} X_i \times W_{ij}^R), \\ U_j^t = \theta(B_j^U + \sum_{i=0}^{N_1} h_j^{t-1} \times W_{ij}^{hU} + \sum_{i=0}^{N_2} X_i \times W_{ij}^U), \\ \tilde{h}_j^t = \Gamma(B_j^{\tilde{h}} + \sum_{i=0}^{N_1} (R_j^t \bigodot h_j^{t-1}) \times W_{ij}^{h\tilde{h}} + \sum_{i=0}^{N_2} X_i \times W_{ij}^{\tilde{h}}), \\ h_j^t = (1 - U_j^t) \bigodot h_j^{t-1} + U_j^t \bigodot \tilde{h}_j^t \\ Y_j^t = \Phi(B_j^y + \sum_{i=0}^{N_3} h_j^t \times W_{ij}^y) \end{cases} \tag{2.6}$$

2.1.3 Transformer

The recurrent neural networks have the capability to process the temporal data through cyclic connections and memory units. However, this process is inherently performed sequentially, where the output of each timestep becomes the next timestep input. Recently, transformer models have emerged to address the lack of parallelization within the training corpus in the recurrent neural network through the attention mechanism. In particular, the transformer is a sequence-to-sequence model [76], to learn a temporal task through the attention mechanisms that work as a differentiate associative memory [78]. A transformer is composed of a decoder and an encoder unit, with L identical blocks on each [79]. The encoder and decoder mainly include multi-head self-attention, position-wise fully connected, and normalization layers. In addition to these layers, the decoder has a cross-self-attention layer between the multi-head attention and the fully connected layers [79]. The transformer can be used for classification by using the encoder only or can be used as a generative model by using the decoder only [79]. A node in an encoder unit computes Equation (2.7) where X_i is the input vector, D_K indicates the dimension

of key matrix, *Addnorm* is a function that performs addition and normalization over input operands, $[W_{ij}^{q_1}, W_{ij}^{k_1}, W_{ij}^{v_1}, \ldots, W_{ij}^{q_m}, W_{ij}^{k_m}, W_{ij}^{v_m}, W_{ij}^{O}, W_{ij}^{p_1}, W_{ij}^{p_2}]$ are a set of weight tensors, θ is the activation function, $[Q_j^1, K_j^1, V_j^1, H_j^1, A_j, Z_j, Z'_j, Y_j]$ represent the embedding query, embedding key, embedding input value, self-attention head vector, the multi-head self-attention vector, normalized cross attention vector, position-wise self-attention vector and the output of the encoder.

$$
\begin{cases}
Q_j^1 = \sum_{i=0}^{N_1} X_i \times W_{ij}^{q_1}, \\
K_j^1 = \sum_{i=0}^{N_2} X_i \times W_{ij}^{k_1}, \\
V_j^1 = \sum_{i=0}^{N_3} X_i \times W_{ij}^{v_1}, \\
H_j^1 = \sum_{i=0}^{N_4} (Softmax(1/\sqrt{D_K} \times \sum_{i=0}^{N_1} (Q_i^1) \times (K_i^1)^T) \times V_i^1, \\
\quad . \\
\quad . \\
\quad . \\
Q_j^m = \sum_{i=0}^{N_1} X_i \times W_{ij}^{q_m}, \\
K_j^m = \sum_{i=0}^{N_2} X_i \times W_{ij}^{k_m}, \\
V_j^m = \sum_{i=0}^{N_3} X_i \times W_{ij}^{v_m}, \\
H_j^m = \sum_{i=0}^{N_4} (Softmax(1/\sqrt{D_K} \times \sum_{i=0}^{N_5} (Q_i^m) \times (K_i^m)^T) \times V_i^m, \\
A_j = \sum_{i=0}^{N_6} [H^1 \ldots H^m]_i \times W_{ij}^{O}, \\
Z_j = Addnorm(A_j, X_j) \\
Z'_j = \sum_{i=0}^{N_8} W_{ij}^{p_2} \times \theta(\sum_{i=0}^{N_7} Z_j \times W_{ij}^{p_1}) \\
Y_j = Addnorm(Z_j, Z'_j),
\end{cases}
\tag{2.7}
$$

The decoder computation behaves similarly to Equation (2.7), except the inputs are shifted by one position, and another mask-based multi-head self-attention layer is added to control which inputs transfer to the next layer.

2.2 Low-precision arithmetic for DNN

Low-precision arithmetic offers benefits such as reduction in memory footprint, reduction in latency, reduction in communication bandwidth, and reduction in energy consumption of DNN models. In this approach, the DNN parameters (e.g., weights, activations, and gradients) are represented with less than 32-bit or 16-bit precision (number of bits available to express a value). Additionally, the arithmetic operations in DNN inference and training procedure are performed partially or fully in low-precision arithmetic [80]. By reducing the bit precision, it brings the possibility to deploy these models to resource-constrained edge devices. However, the low-precision arithmetic produces numerical errors that have a side effect on the DNN training and inference accuracy compared to the high-precision DNN models. Therefore, designing a low-precision DNN model with similar performance as the high-precision DNN model counterparts with minimal computational overhead is the potential benefit of using low-precision arithmetic. On the other hand, quantization is an approach to discretize a continuous function into a set of intervals and map each interval to a value (called quantized value) [81]. Although these two terms are sometimes used interchangeably in the literature, quantization is a sub-part of low-precision arithmetic.

Performing DNN inference and training using low-precision arithmetic comprises two key aspects: i) Numerical format, and ii) quantization method.

2.2.1 Numerical format

The numerical format includes a set of numbers and procedures that need to be defined by a designer or a standard. For instance, how many bits are assigned for the exponent and fraction for a new floating-point format, what function should be used to express the values, how arithmetic is computed, how rounding is performed, and how to represent zero, and $\pm\infty$. For instance, a brief description of various rounding functions used in numerical formats is provided.

Rounding function: The various deterministic rounding functions commonly used in various numerical formats are shown in Figure 2.2.

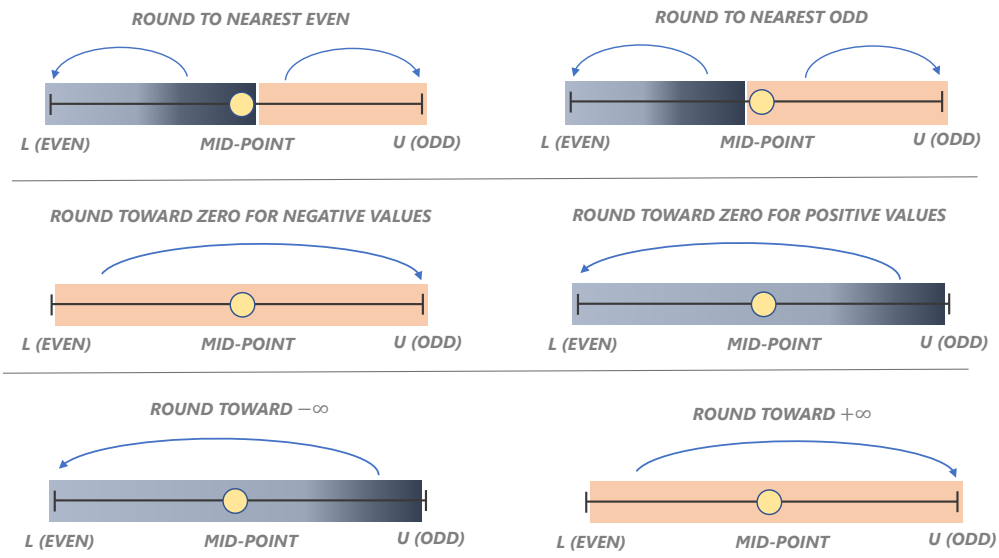

Figure 2.2: Various deterministic rounding functions

While there are many approaches to rounding, the most frequently applied functions are either rounding to the nearest zero (truncation) or to the nearest even number. Among these rounding approaches, the round-to-nearest-even has

less average rounding error (0.25 unit of least precision (ULP) in performing the two-operand function). However, the hardware implementation of this rounding function is more complex than the others. On the other hand, the truncation can be simply implemented in hardware but the average rounding error is increased to 0.5 ULP. However, to perform MAC operations, the occurrence of stagnation during summation (the summation result is not changed when a new summand is added) cannot be avoided by using round-to-nearest-even approach. This problem can be solved using the stochastic rounding [82] where the round-up and down are performed based on probability value (P_i) as shown in Equation (2.8) where x_i are actual values, x_i' are rounded values, and fs indicates the number of fraction bits for 32-bit floating-point format. However, stochastic rounding consumes more energy than the round-to-nearest rounding approach [83].

$$x_i' = SR(x_i, fs) = \begin{cases} \lfloor x_i \rfloor & \text{if } (P_i \geq \frac{x_i - \lfloor x \rfloor}{2^{-fs}}) \\ \lfloor x_i \rfloor + 2^{-fs}, & \text{otherwise} \end{cases} \tag{2.8}$$

2.2.2 Quantization method

The quantization methods to perform DNN inference and training using low-precision arithmetic, fall under three categories: quantization aware training (QAT), post-training quantization (PTQ), and fully quantized training (FQT). To perform DNN inference, the QAT approaches train or fine-tune DNNs with simulated quantization operations to learn quantized weights and activations, which increases computational complexity [37]. On the other hand, no training is involved in the

PTQ approach [39]; the high-precision parameters are directly converted to low-precision parameters without the need to access the training procedure. The PTQ can be data-free (also called zero-shot quantization) or data-dependent (e.g., using the calibration approach [84]). To perform DNN training, the FQT approach is used when in addition to weights and activations, the gradients are also quantized [85].

Quantization function: The quantization function $Q(x_i, q, l, u, s, z)$ estimates each 32-bit floating-point DNN parameter x_i as x_i' (a q-bit low-bit precision numerical format), as defined in Equation (2.9) where s and z are scaling factor and zero point [37], respectively. Given the dynamic range of a low-precision numerical 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 with a rounding function ($Round(x_i)$) to the values that is expressible in low-precision numerical format.

$$x_i' = Q(x_i, q, l, u, s, z) = \text{Round}(\text{Clip}((s \times x_i - z), l, u)) \tag{2.9}$$

Quantization parameters: The s and z is quantization parameters and computed as a 0 for rounding quantization and as in Equations (2.10) and (2.11) for linear quantization where $\alpha = min(x_i)$ and $\beta = max(x_i)$ for asymmetric quantization and $-\alpha = \beta = max(|min(x_i)|, |max(x_i)|)$ for symmetric quantization [81].

$$s = \frac{|l - u|}{|\alpha - \beta|} \tag{2.10}$$

$$z = mean(clip(x_i, \alpha, \beta)) \tag{2.11}$$

Quantization granularity: The quantization parameters are selected based on the statistic of DNN parameters. This statistic depends on the granularity of DNN parameters and is categorized into four subgroups: layerwise quantization, multiple channel-wise quantization, channel-wise quantization, and sub-channel-wise quantization [81]. Selecting between these granularity depends on hardware and accuracy trade-offs. Sub-channel-wise quantization is the most computationally expensive and accurate approach.

Applicability of quantization approach: The quantization approach for DNN inference is categorized into three levels, depending on access to training data and retraining requirement [86]: (level-1) data and training free, (level-2) data-dependent, and training free, (level-3) data-dependent and training dependent. The QAT approach is a level-3 approach, and PTQ can be either the level-1 or level-2 approach. Proposing a level-1 quantization approach with no loss in accuracy is desired but in most cases, using the level-1 approach results in accuracy loss; thus level-2 and level-3 approaches are more commonly implemented to achieve the application target accuracy.

2.3 Tapered-Precision numerical format

The weights and activations of DNN models are distributed non-uniformly (bell-shaped) and are approximated with normal distribution [87]. On the other hand, the DNN gradient distributions are heavy-tailed and approximated with a log-normal distribution [88]. The DNN parameters (weights, activations, and gradients) can be

expressed using an equal-magnitude spacing numerical format such as fixed-point along with adaptive non-uniform quantization [81]. However, performing adaptive non-uniform quantization, commonly through a clustering algorithm, leads to a surge in hardware resource requirements for DNN training and inference [81]. The alternative approach is to suitably represent DNN parameters with the tapered precision numerical format that expresses values in unequal-magnitude spacing such that the density of values is maximum near zero and is tapered towards the maximum representable number. The tapered precision numerical format was proposed by Morris in 1971 [89]. Afterward, variants of these numerical formats have been studied in the literature [90–94]. However, these numerical formats either have redundant encoding to represent a value or are not hardware-oriented approaches. Recently, John L. Gustafson proposed new tapered precision numerical formats such as posit [6], generalized posit [95], and tapered fixed point [96] that addresses the shortcoming of the previous tapered precision numerical format. In this section, we introduce these numerical formats and compare them with floating point and fixed-point numerical formats in terms of numerical error.

2.3.1 Posit

The posit numerical format, a type III *universal numbers* (unums) [97], is proposed to improve upon the arithmetic shortcomings of the IEEE-754 floating point format such as complicated sign-magnitude arithmetic, dependence upon overflow/underflow detection, and lack of algebraic associativity. Posits also address complaints about hardware complexity of Type I due to variable-length encod-

ing and scalability of type II unums due to lookup table (LUT) based encoding. Posits yield better arithmetic accuracy for near zero values, dynamic range, and reproducibility than IEEE-754 floating point of the same bit-length [6]. Tapered-accuracy, where the density of values is maximum near zero and is tapered towards the maximum representable number [95], is one of posits' characteristics that distinguishes it from floating point that has roughly constant relative values' density. The tapered-accuracy attribute of posits comes from the variable-length regime (signed unary encoded) in their binary representation.

Like IEEE floating point, binary representations contain a sign bit, exponent bit, and fraction bits (shown in Figure 2.3).

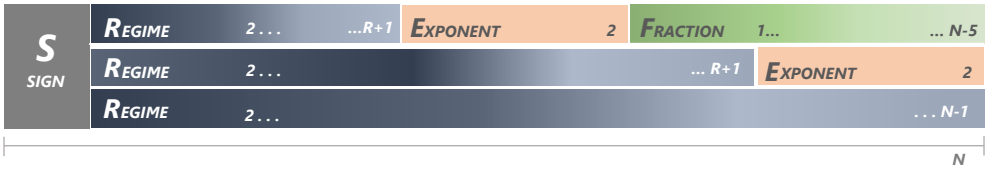

Figure 2.3: Illustration of possibilities for N-bit posit numerical format with 2-bit exponent ($es = 2$). Either exponent or fraction bits can be truncated and padded with zero values.

However, posits also have regime bits. The regime is encoded using a run-length (m) of identical bits $(r...r)$. It is terminated either by a termination bit $\bar{r}$ or by the least significant bit. The regime value k equals $-m$ if the r is zero, and $m-1$ if the r is one. After the regime, posits have an es-bit exponent e (unsigned integer value e) followed by fraction bits $f, 0 \leq f < 1$. Note that in posit format (shown in Figure 2.3), truncated bits for the exponent and fraction are padded with zero values. With a leading sign bit, s, the real number represented by a posit is

given by Equation (2.12),

$$x = \Big((1-3s)+f\Big) \times 2^{(1-2s)\times(2^{es}k+e+s)}, \tag{2.12}$$

with special cases for zero and not-a-real (NaR) excluded. Unlike the NaN values of floating point, NaR includes the non-real values ∞ and $-\infty$. The values that are too large will round to the largest magnitude posit instead of overflowing to ∞. The values that are too small will round to the smallest magnitude posit instead of underflowing to zero. The dynamic range of this numerical format is shown as Equation (2.13) where x_P are positive representable values.

$$D_P = \frac{\text{Max}(x_P)}{\text{Min}(x_P)} = 2^{2^{es+1}\times(N-2)} \tag{2.13}$$

For instance, 32.3125 in posit, with $N = 16$ and $es = 2$, can be represented as shown in Figure 2.4.

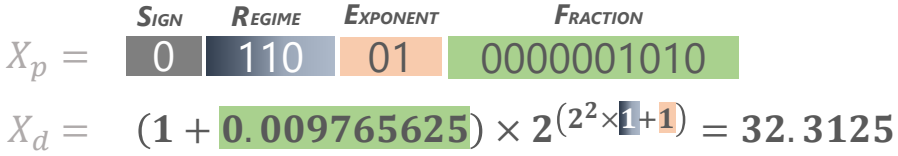

Figure 2.4: Representation of a number in the posit format $P(N = 16, es = 2)$.

Conversion from posit to decimal value (posit decoding): The value in posit representation is decoded in order to extract the sign, regime, exponent, and fraction and then uses Equation (2.12) to compute the decimal value. As the regime bit field has variable size, this process is nontrivial. Algorithm 1 in the Appendix A presents the posit decoding procedure.

In the first step, the zero, and NaR values and sign are captured, and the two's complement of the negative input is calculated (lines 2–4). According to the first bit of regime encoding, the inverse of the result is calculated (lines 5–6). This step is performed to bypass the requirement to compute both Leading Zero Detection (LZD) and Leading One Detection (LOD) to extract regime value. The regime value is extracted using the LZD algorithm. The regime bits are shifted out, and the exponent based on the *es* value and fraction is obtained (lines 7–12). Note that a *regime-terminating bit* $\overline{r}$ and extended zero are assumed if we run out of space (n bit-width) to compute the regime-bit as well as the exponent and the fraction bits.

Conversion from decimal to posit (posit encoding): Algorithm 2 in the Appendix A presents the posit encoding procedure. To encode posits, at the first step, the zero, NaR values, and sign are determined (lines 2–3). By capturing the sign bit, the absolute value of the real number is enough to determine other posit encode bits (line 4). The regime bit (*reg*) is computed by R times dividing or multiplying a real number by $2^{2^{es}}$ until the number is in the range [1, $2^{2^{es}}$). Otherwise, the aforementioned range condition is enough to terminate this process (lines 5–19). To find the exponent, this process is continued until the number is in the range [1, 2) (lines 20–27). To compute the fraction, the remaining value is diminished by one and rounded to the nearest even number (lines 28–29).

Quire: Quire format is a wide fixed-point two's complement format that can be used for fused/exact multi-operand computations, which is re-branded as the "exact accumulator" proposed by Ulrich Kulisch [98]. In particular, the bit width of the quire format is selected such that the maximum and minimum values produced

during accumulation are expressible. Therefore rounding errors become zero when the quire format is used. The posit standard [99] supports the quire format, as shown in Figure 2.5. For instance, to compute the fused multiply-add ($a \times b + c$), the a and b are multiplied in a posit format without rounding or truncation at the end of multiplication. The product ($a \times b$) and c are stored in a quire format and accumulated. The result of accumulation is rounded to the nearest value that is representable in the posit numerical format. Therefore, in this approach, rounding is delayed until the last operation in the computations.

Figure 2.5: Illustration of quire format with 2-bit exponent ($es = 2$). For instance, the quire size for 8-bit posit with $es = 2$ is 128 bits).

2.3.2 Generalized posit

The generalized posit (GP) format, as shown in Figure 2.6, is an extended version of posit numerical format where two parameters (exponent bias (e_b) and maximum regime run-length (rs)) are added to have the capability to adapt to the distribution of parameters across various applications. There is no additional memory requirement to store these parameters if the values of these parameters are predefined for specific applications.

In particular, a parameter $e_b \in [-\frac{N-2}{2}, \frac{N-2}{2}]$ can re-center the location of maximum tapered accuracy from zero to e_b. Restricting the regime length $rs \in [1, N-1]$ prevents the exponent and fraction bit fields from vanishing at the extremes of the

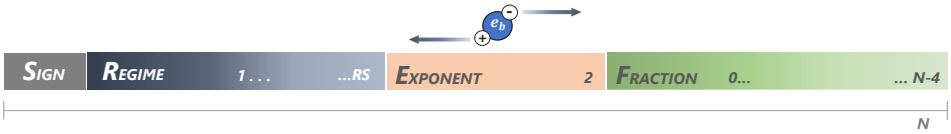

Figure 2.6: Illustration of N-bit generalized posit numerical format with 2-bit exponent ($es = 2$).

dynamic range. The effect of these parameters on the distribution of values is shown in Figure 2.7. Notably, generalized posit formats encompass IEEE-like floating point with $rs = 1$ (fixed fraction field), standard posits with $rs = N - 1$, and other tapered-precision formats between those bounds. The real number represented by a generalized posit is given by (2.14).

$$x = ((1 - 3s) + f) \times 2^{(1-2s)\times(2^{es}k+e+s+e_b)} \tag{2.14}$$

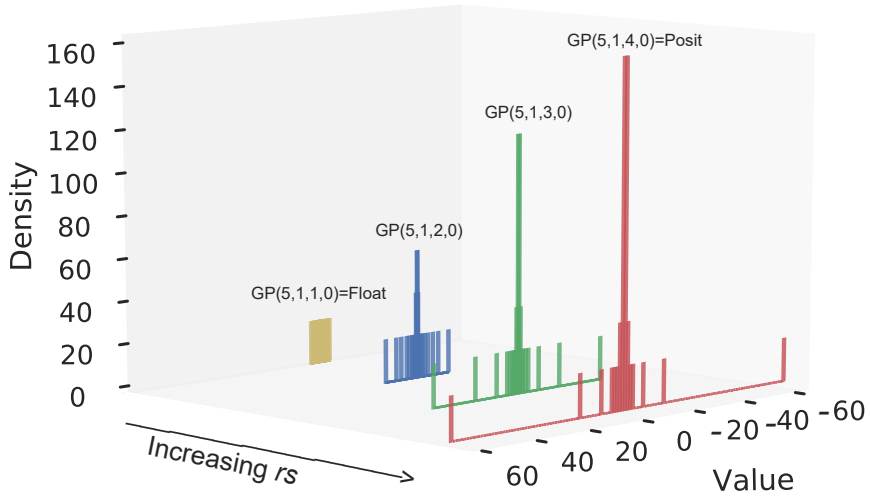

Figure 2.7: Illustration of the distribution of values in generalized posit numerical format with $N = 5$, $rs = \{1,2,3,4\}$ and $e_b = 0$

The dynamic range of this numerical format is shown as Equation (2.15) where

$t = N - rs - 1$ and x_{GP} are positive representable values. Note that e_b does not change the dynamic range but only shifts the range of values represented by generalized posit numerical format.

$$D_{GP} = \frac{\text{Max}(x_{GP})}{\text{Min}(x_{GP})} = \begin{cases} 2^{2^{es+1}\times rs} \times 2^{2^{es}-t-1}, & \text{if } (t \leq es) \\ 2^{2^{es+1}\times rs} \times \frac{(1-2^{es-t-1})}{(1+2^{es-t})}, & \text{otherwise} \end{cases} \tag{2.15}$$

For instance, the same bit pattern illustrated in Figure 2.4 can be represented in the generalized posit numerical format with $N = 16$, $es = 2$, $rs = 1$, $e_b = -2$ as shown by Figure 2.8.

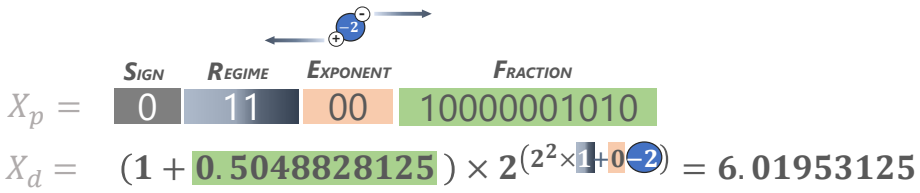

Figure 2.8: Representation of a number in the $GP(N = 16, es = 2, rs = 2, e_b = -2)$ format

Conversion from generalized posit to decimal (GP decoding): Algorithm 3 in the Appendix A presents the generalized posit decoding procedure. The generalized posit decoding procedure is similar to the posit decoding procedure. The major difference between them is the LZD operation (line 7). To extract the regime bits, the GP decoder should perform the LZD operation on fewer bits (rs bits) compared to the posit decoder that should compute the LZD operation on N bits.

Conversion from decimal to generalized posit (GP encoding): Algorithm

4 in the Appendix A presents the generalized posit encoding procedure. The generalized posit encoding procedure is similar to the posit encoding procedure. The main difference between them is the generalized posit encoder requires an additional comparison operation to check the regime overflow occurrence which adds additional hardware complexity to the encoder (lines 7–14). This comparison operation is not needed in the posit encoder since the zero and NaR values are separately extracted.

Quire: The quire format for generalized posit is not defined in the posit standard [99]. It is possible to use the quire in posit format for generalized posit, as shown in Figure 2.9, since it captures the dynamic range of generalized posit numerical format for various rs. The alternative approach is to use Equation (2.16) to compute the quire size.

$$w_q = 32 + 2 \times \theta = 32 + 2 \times \left\lceil \log_2 \left(\frac{\text{Max}(x_{GP})}{\text{Min}(x_{GP})} \right) \right\rceil \tag{2.16}$$

Figure 2.9: Illustration of quire format with variable es and rs (e.g, the quire size for 8-bit posit with $es = 2$ and $rs = 3$ is 80 bits)

Symmetric vs asymmetric tapered-precision: The tapered-precision is symmetric about the origin when the generalized posit values are represented in the log domain as shown in Figure 1. However, the outcome of prior studies indicated that DNN parameters are distributed asymmetrically about the origin in the log domain. To address this mismatch, it is possible to separate the rs parameter for values

greater than one as rs_u and values less than one rs_d. This gives the advantage of supporting dynamic ranges and tapered-precision for numbers less or greater than one.

2.3.3 Tapered fixed-point

The tapered fixed-point numerical format (TFX), or taper [96], can be illustrated as a combination of the generalized posit and fixed-point numerical formats as shown in Figure 2.10. It combines the hardware-oriented characteristics of fixed-point and the high accuracy of generalized posit with tapered-precision. Specifically, the binary encoding to represent the integer bits in fixed-point is replaced with the regime encoding that was previously used to represent the regime bits in the generalized posit format. This change adds tapered-precision characteristics to the fixed-point numerical format. The fraction remains the same as the standard fixed-point where the fraction is added to the integer rather than scaling, which reduces the hardware complexity as compared to generalized posit and floating-point.

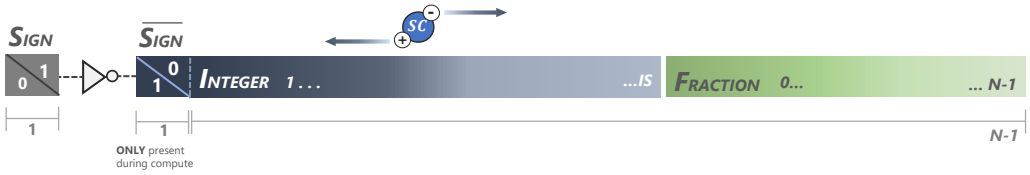

Figure 2.10: Illustration of N-bit tapered fixed-point format.

In particular, IS (in a range of $[1,N]$) indicates the maximum number of regime encoded integer bits and $SC \in [-\frac{N-1}{2}, \frac{N-1}{2}]$ is the power-of-2 scaling downward or upward. The IS value controls both the dynamic range and the tapered-precision.

Similar to generalized posit, the dynamic range is shifted with variation in the *SC* parameter. The effect of these parameters on the distribution of values that are expressible by tapered fixed-point is shown in Figure 2.11, where the real number represented by a tapered fixed-point is given by Equation (2.17). In this equation, I is computed in Equation (2.18) representing the integer value, f indicates the fraction value, and fs the maximum number of bits allocated for the fraction.

$$X = (I + \frac{f}{2^{fs}}) \times 2^{SC} \tag{2.17}$$

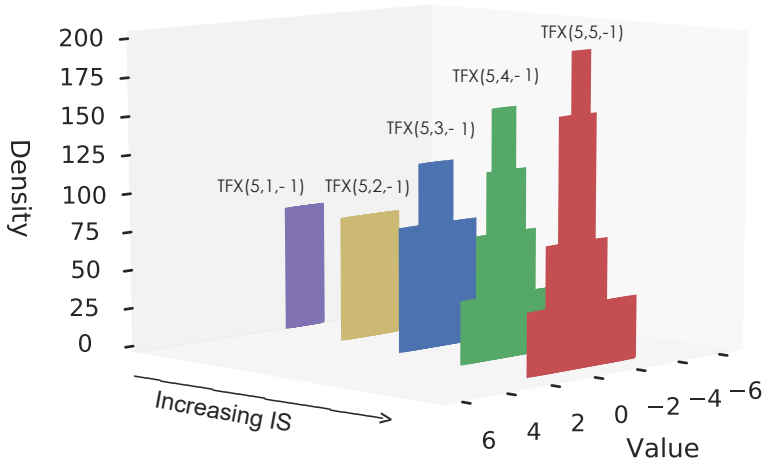

Figure 2.11: Illustration of the distribution of values in tapered fixed-point format with $N = 5$, $IS = \{1,2,3,4,5\}$ and $SC = -1$

The integer bit-field is encoded as similar to the regime encoding in generalized posit except for the sign bit (s) that is flipped and is also considered as the first bit

of the integer ($\overline{s} = i$).

$$I = \begin{cases} -m, & \text{if } i = 0 \\ m-1, & \text{if } i = 1 \end{cases} \tag{2.18}$$

The dynamic range of this numerical format is shown as Equation (2.19) where x_{TFX} are positive representable values.

$$D_{TFX} = \frac{Max(x_{TFX})}{Min(x_{TFX})} = \begin{cases} 2^{N-1}, & \text{if } IS = 1 \\ IS \times 2^{N-2}, & \text{otherwise} \end{cases} \tag{2.19}$$

For instance, the same bit pattern illustrated in Figure 2.4 can be represented with tapered fixed-point format with $N = 16$, $IS = 4$ and $SC = -2$ as shown in Figure 2.12.

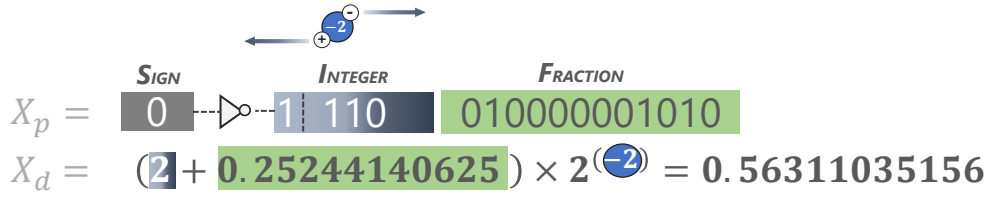

Figure 2.12: Representation of a number in the $TFX(N = 16, IS = 3, SC = -2)$ format

Conversion from decimal to tapered fixed-point (TFX decoding): Algorithm 5 in the Appendix A presents the tapered fixed-point decoding procedure. The tapered fixed-point decoder is similar to generalized posit decoder with $es = 0$ except that the inverse of the sign bit is represented as the first bit of the integer. It is computationally more complex than fixed-point decoding due to the LZD

requirement.

Conversion from decimal to tapered fixed-point (TFX encoding): An algorithm 6 in the Appendix A presents tapered fixed-point encoding procedure. The tapered-precision encoding is different from generalized posit encoding with $es = 0$. In the first step, the sign is captured to encode tapered fixed-point. The absolute value of a real number is used to determine other tapered fixed-point encoding. The integer value is computed by $R \leq IS$ time subtracting and adding a real number by one until the number is in the range [0,1). To compute the fraction, the remaining value is rounded to the nearest even number.

Quire: The quire is not defined for tapered fixed-point [96]. However, it is possible to define quire format as shown in Figure 2.13 and use Equation (2.20) to compute the quire size.

$$w_q = 32 + 2 \times \theta = 32 + 2 \times \left\lceil \log_2 \left(\frac{\text{Max}(x_{TFX})}{\text{Min}(x_{TFX})} \right) \right\rceil \tag{2.20}$$

Figure 2.13: Illustration of Quire format with variable IS and SC (e.g, the quire size for 8-bit tapered fixed-point with $IS = 8$ and $SC = 0$ is 50 bits)

2.4 Error analysis for tapered-precision formats

In numerical analysis, the most commonly used metric for analyzing the quantity of computation errors is absolute error, especially for values expressed uniformly

by a numerical format [100]. However, the absolute error does not produce a meaningful result when a number is represented by a numerical format that has a large dynamic range [100]. The alternative solution is to define a relative error and relative accuracy as in Equation (2.21) where the x_d is an actual decimal value that is approximated as an expressible number in arbitrary numerical format (x_{nf}).

$$RA = \frac{1}{RE} = \frac{1}{|\ln(x_{nf}/x_d)|} \tag{2.21}$$

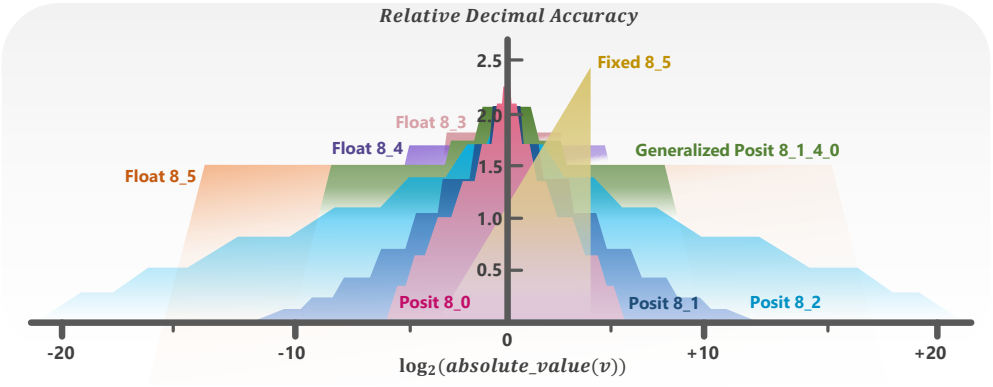

Figure 2.14: The relative decimal accuracy [6] for various 8-bit numerical formats Float 8_5 , Float 8_4, Float 8_3 are 8-bit floating point numerical formats with 5, 4 and 3 exponent bits, respectively, and Posit 8_0, Posit 8_1, and Posit 8_2 are 8-bit posit numerical formats with 0, 1, and 2 exponent bits respectively. The Fixed 8_5 indicates fixed-point numerical format with 5-bit integer and 3 fraction bits, and Generalized posit 8_1_4_0 is 8-bit generalize posit numerical format with $es = 1$, $rs = 4$, and $e_b = 0$.

The decimal accuracy of various 8-bit numerical formats is shown in Figure 2.14. The posit's decimal accuracy has a tent-shaped distribution, and maximum decimal accuracy occurs where $x_p \in [-2^{2^{es}}, 2^{2^{es}}]$. The decimal accuracy is tapered towards the minimum magnitudes value $-2^{2^{es} \times (n-2)}$ and maximum magnitudes

value $2^{2^{es}\times(n-2)}$ (tapered decimal accuracy). On the other hand, the floating point has an almost uniform-shaped decimal accuracy distribution with a ramp-shape decimal accuracy for subnormal values. Moreover, the decimal accuracy of fixed-point distribution takes long ramp-shape where it is tapered from the maximum magnitude value to the minimum magnitude value. In variant applications, the parameters are distributed in a bell-shape as shown in Figure. 2.14. Therefore, posit can present these parameters more accurately as compared to floating point and fixed-point due to tapered-precision. By using the *rs* parameters the decimal accuracy distribution of generalized posit captures the decimal accuracy of float, posit, and other numerical formats in between.

2.5 Summary

In the first section of this chapter, an overview of the most widely used deep learning models (feedforward neural network, convolutional neural network, recurrent neural network, and transformer) and training/inference procedures are presented. The use cases, connectivity between layers, and a computation graph executed on a node in a layer, are highlighted. When it comes to deep learning model computations, MAC operation is the dominant computation in a DNN node. One practical solution to reduce the computational complexity of MAC operations and other arithmetic operations in DNN models is performing DNN inference and training within low-precision arithmetic. However, the benefit of using low-precision arithmetic to reduce the complexity of a model comes at the

cost of degraded inference and/or training accuracy. Meeting this dual objective (high inference and training accuracy and minimal hardware resources) heavily depends on the choice of numerical formats and quantization approaches. Thus, the characteristic of appropriate numerical format for DNN models and the quantization approaches landscape is discussed in the second section of this chapter. As a result of numerous prior studies, it has been found that the low-precision numerical format for DNN models requires: (i) to have the capability to represent parameters with non-uniform and, in many cases, skewed distributions, (ii) can capture the dynamic range of DNN parameters, (iii) the ability to accommodate the variability observed in inter-, and intra-layer parameter distributions and dynamic range of DNNs, and (iv) to provide the best trade-off between quantization error and hardware complexity. Therefore, the three newest tapered precision numerical formats (posit, generalized posit, and tapered fixed-point) is explained as candidate numerical formats for DNN models. Through the dynamic range and numerical error analysis of these numerical formats, It can be concluded that they can support some to all of these characteristics which are lacking in conventional formats, such as fixed-point, floating point formats.

3. Related work

3.1 Empirical analysis of low-precision

As early as the 1980s, low-precision arithmetic has been studied for shallow neural networks to reduce compute and memory complexity on performing training and inference without degradation in performance [80, 101]. It may also improve the training and inference performance when the noise generated by the low-precision shallow neural network parameters hypothetically acts as a regularization method [102,103]. The outcome of these studies indicated that 8- to 16-bit precision is sufficient for training and inference on shallow networks [80,101,103]. Motivated by prior studies, the benefits of low-precision arithmetic are reevaluated in the deep learning era to reduce memory footprint and energy consumption during training and inference [104, 105]. Several of these studies can be categorized in terms of numerical formats as (i) variants of IEEE-754 standard numerical format [106], (ii) variants of block floating point, (iii) variants of fixed-point, and (iv) posit.

3.1.1 Variants of IEEE-754 standard numerical format

Low-precision DNN training: Traditionally 32-bit IEEE standard floating point format is used for DNN training. However, this floating point format is designed for representing a wide dynamic range ($> 10^{80}$) of numeric values; this range is much larger than what is needed to represent DNN parameters. For example, the DNN models studied in this thesis only span a dynamic range of ($\approx 10^{20}$).

Several prior studies trained DNN with 16- and 32-bit mixed-precision IEEE standard floating point parameters to reduce this dynamic range mismatch (as shown in Figure 3.1 and Table 3.1). In these studies, 16-bit floating point is used to represent weights, activations, gradients, and errors to perform forward and backward passes. In most of these approaches, the weights are updated in 32-bit floating point format to prevent accuracy loss caused by underflow in the product of learning rate and gradients in stochastic gradient descent (SGD) optimization. Additionally, the dynamic range of 16-bit floating point is shifted using a new loss scaling approach to represent gradients with a very small magnitude [107]. Unfortunately, this approach fails to preserve inference accuracy in DNN models such Single-Shot Detector (SSD), which has high variability in the dynamic range of parameters across layers [107]. To capture this dynamic range's variability, the loss scaling factor's bit-width may be increased, however, this also increases the computational complexity. A less computational complex alternative approach is representing DNN parameters with the brain floating point (BFloat16) format that is proposed by Google and used as a numerical format in TPU V2 AI accelerator [108]. This numerical format has a similar exponent bit-width (8-bit), and less fraction

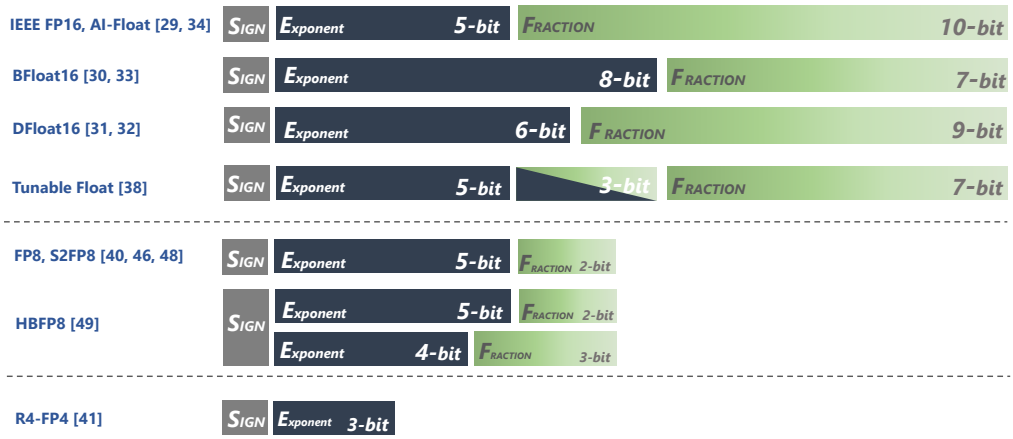

Figure 3.1: Variants of IEEE-754 low-precision numerical format are used in DNN training. The main differences among approaches are the bit width of exponent and fraction. The Tunable Float has three configurable bits that can be assigned to either the exponent or the fraction, based on application requirements. The 4-bit training requires radix-4 representation to capture the dynamic range of DNN gradients.

bit-width (7-bit) compared to 32-bit floating point. The same exponent bit-width of BFloat16 and 32-bit floating point reduces the conversion complexity between these two formats by performing Round-to-Nearest-Even (RNE) operation on lower 16 bits of 32-bit floating point DNN parameters. In training a ResNet-50 model on the ImageNet dataset, BFloat16s achieve the same performance as 32-bit floating point when bias parameters and a master copy of the weights in SGD are represented in 32-bit floating point [109].

However, this approach requires to support of both BFloat16 and 32-bit float arithmetic units in DNN training and thereby doubles the memory footprint. To reduce the memory footprint, Zamirai *et al.* trained ResNet-50 models on the ImageNet dataset with only BFloat16 arithmetic by combining it with either stochastic

Table 3.1: The DNN training performance using variants of the IEEE-754 standard floating point format on CIFAR-10, and ImageNet.

Numerical Format	Bit-Configuration[1]								Approach[2]				CIFAR-10		ImageNet	
	W	A	G	E	Acc	UP	BN	Act	LS	KS	SR	CA	ResNet-18	ResNet-50	ResNet-18	ResNet-50
FP16 [107]	16	16	16	16	32	32	32	16	✓	×	×	×	-	-	-	76.04(+0.12)%
BFloat16 [109]	16	16	16	16	32	32	16	16	×	×	×	×	-	-	-	74.04(+0.00)%
DFloat16 [112]	16	16	16	16	32	32	16	16	×	×	×	×	-	-	-	74.00(0.00)%
DFloat16-d [42]	16	16	16	16	32	32	16	16	✓	×	×	×	-	-	-	75.90(0.00)%
BFloat16 [111]	16	16	16	16	32	16	16	16	×	✓	✓	×	95.36(-0.09)%	-	-	75.61(-0.09)%
AI-Float16 [113]	16	16	16	16	16	16	16	16	✓	×	✓	×	-	-	-	75.90(0.00)%
HBFP8-B [115]	8	8	8	8	32	32	32	32	✓	×	×	×	-	-	70.29(-0.06)%	76.61(+0.04)%
S2FP8 [115]	8	8	8	8	32	32	32	32	✓	×	×	×	91.10(-0.40)%	93.2(+0.20)%	69.60(-0.70)%	75.20(-1.00)%
L-FP8 [116]	8	8	8	8	24	16	32	16	✓	×	✓	✓	93.41(-0.10)%	-	-	76.14(-0.24)%
FP8 [117]	8	8	8	8	16	16	16	16	✓	×	✓	✓	-	-	66.95(-0.48)%	73.14(-0.42)%
HBFP8 [118]	8	8	8	8	16	16	16	16	✓	×	✓	✓	-	-	69.39(+0.01)%	76.22(-0.22)%
FP4(R4)-FX4 [48]	4	4	4	4	16	16	16	16	✓	×	×	✓	92.74(-0.27)%	93.42(-1.28)%	68.27(-1.13)%	74.01(-2.47)%

[1] W: Weights; A: Activations, G: Gradients, E: Error, Acc: Accumulation, UP: Update, BN: Batch Normalization, Act: Activation
[2] LS: Loss Scaling, KS: Kahan Summation, SR: Stochastic Rounding, CA: Chunk-Based Accumulation

rounding [82] or Kahan summation [110] approaches [111].

Although the BFloat16 numerical format is supported in most of AI accelerators as a replacement for IEEE-754 floating point, the small fraction bit-width (7-bit) of BFloat16 introduces large quantization error in gradients computation in DNN training [111]. To reduce the quantization error, various approaches are proposed [42, 112–114]. For instance, DFloat-16 proposed by IBM allocated 6 bits for the exponent, therefore, the number of fraction bits is increased from 7 bits in BFloat16 to 9 bits in DFloat16 [112]. The DFloat16 shows 5% performance improvement over BFloat16 in training a transformer on WMT 2014 English-to-German translation task [112]. Recently, to have more flexibility in fractional bit-width, Nannarelli proposed a tunable floating point (TFP) numerical format where fractional bit-width can be dynamically modified from 7- to 10-bit to meet the application requirements.

Additionally, to further reduce the hardware complexity compared to variant

16-bit floating point formats, researchers trained DNN models utilizing DNN parameters with two precision levels: (8-bit and 16-bit), [38] and (8-bit and 32-bit) [119]. In these studies, stochastic rounding, 2-level chunk-based accumulation (chunk size equals 64), and loss scaling approaches are applied to minimize the rounding and overflow errors. For instance, Xiao Sun *et al.* successfully trained a variety of DNN models such as ResNet-50, Transformer, and LSTM with a hybrid 8-bit precision floating point format (HFP8). In this format, 8-bit (four exponent bits and three fraction bits) represents high precision demands of weights and activations in the forward pass, and 8-bit (five exponent bits and two fraction bits) represents high dynamic range demands of gradients in the backward pass. The DFloat16 numerical format is used for arithmetic operations in batch normalization layers, activation layers, SGD procedures, and all accumulation operations. To prevent the requirement of the hybrid 8-bit precision, Cambier *et al.* proposed a new 8-bit floating point numerical format with shift and squeeze hyperparameters (S2FP8). In this approach, the mean and standard division of DNN parameters distribution is adapted to the mean and standard division of distribution of 8-bit floating point values over training epochs, [115]. The outcome of this study indicates that 8-bit floating point is sufficient to achieve training accuracy within 1% variation of 32-bit floats with ResNet-50 on the ImageNet dataset. To further reduce the hardware complexity, the possibility of training DNN models with hybrid 4-bit radix-4 floating point and radix-2 fixed-point (FP4(R4)-FX4) representation is being examined [48]. To achieve DNN training convergence approaches such as PACT [120], SAWP [121], two-phase rounding, and gradient scaling are

adopted along with FP4(R4)-FX4 numerical format [48]. Overall, **Training DNNs with hybrid 4-bit and 8-bit radix2 floating point arithmetic is still an open question**.

Low-precision DNN inference: Several groups have demonstrated that 8-bit floating point numerical format are adequate to represent weights and activations without significantly degrading performance yielded with 32-bit floating point as shown in Figure 3.2 and Table 3.2 [38, 47, 122, 123]. For instance, Deng *et al.* demonstrated 8-bit floating point numerical format (4-bit exponent and 3-bit fraction) to represent weights for AlexNet and VGG-16 DNN models on the ImageNet dataset [122]. The results indicated that it is possible to represent 20% of the weights in 8-bit floating point representation with less than 1% degradation of accuracy. Following this work, Gysel *et al.* showed that 8-bit floating point (5-bit exponent and 2-bit fraction) weights and activations, 8-bit multipliers, and 32-bit accumulation results in less than 1% accuracy loss on AlexNet with the ImageNet corpus [47]. Moreover, Sun *et al.* evaluated that 8-bit floating point (4-bit exponent and 3-bit fraction) with a fixed scaling factor of 4, shows less quantization error compared to 8-bit floating point (5-bit exponent and 2-bit fraction) for a variety of DNN models [38]. To further reduce bit precision and hardware complexity, Tambe *et al.* [123] introduced an adaptive ≤ 8 floating point numerical format (AFP). This numerical format has the capability to accommodate the variability observed in inter- and intra-layer parameter dynamic range of DNNs through parameterizing the exponent bias (E_b) in floating point numerical format. The result of this study indicated that it is possible to reduce the bit precision of weights and activations to

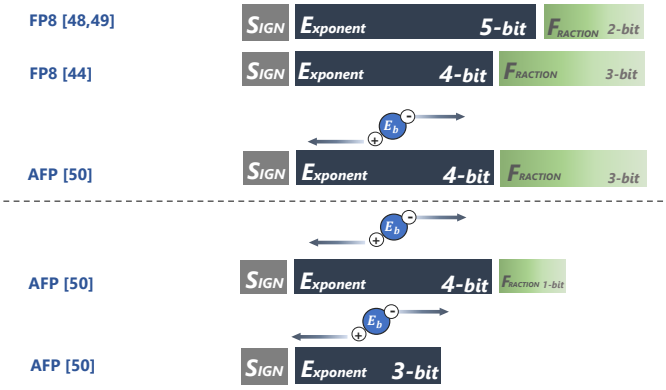

Figure 3.2: Variants of IEEE-754 low-precision numerical format are used in DNN Inference. An exponent bias is used in the adaptive floating point to adjust the dynamic range of the floating point with the layerwise DNNs parameters' dynamic range.

Table 3.2: The DNN Inference performance using variants of the IEEE-754 standard floating point formats on CIFAR-10 and ImageNet.

Numerical Format	Bit-Configuration[1]					Approach[2]				CIFAR-10		ImageNet	
	W	A	Acc	BN	Act	PQ	EB	SR	CA	ResNet-20	ResNet-50	ResNet-20	ResNet-50
FP8 (S:1, E:5, F:2) [122]	8	32	32	32	32	✓	×	×	×	-	-	-	-
FP8 (S:1, E:5, F:2) [47]	8	8	32	32	32	×	×	×	×	-	-	-	-
FP8 (S:1, E:4, F:3) [38]	8	8	16	16	16	✓	×	✓	✓	-	-	68.99(-0.33)	76.42(+0.04)
AFP8 (S:1, E:4, F:3) [123]	8	8	32	32	32	×	✓	×	×	-	-	-	75.7(-0.05)%
AFP6 (S:1, E:4, F:1) [123]	6	6	32	32	32	×	✓	×	×	-	-	-	73.90(-2.40)%
AFP4 (S:1, E:3, F:0) [123]	4	4	32	32	32	×	✓	×	×	-	-	-	29.0(-47.30)%

[1] W: Weights; A: Activations, Acc: Accumulation, BN: Batch Normalization, Act: Activation
[2] PQ: Partial Quantization, EB: Exponent Bias, SR: Stochastic Rounding, CA: Chunk-Based Accumulation

6-bit in performing DNN inference with ResNet-50 on ImageNet dataset within 2.4 accuracy degradation compared to 32-bit floating point. Any further reduction of bit precision, such as 4-bit, for both weights and activations, results in significant degradation of accuracy (e.g., 47.3% accuracy degradation using the 4-bit floating point on ResNet50 and ImageNet dataset). **Therefore, DNN inference with 4-bit floating point numerical format is still an open question.**

3.1.2 Variants of block floating point

The block floating point (BFP) is a decades-old approach that was first used in the 1950s, where a block of floating point numbers used a shared exponent [124]. The shared exponent is selected as a common exponent in the block (either minimum or maximum exponent) and all significants in the block are aligned according to the new shared exponent. After this alignment, the operation in the block can be performed with fixed-point arithmetic. Therefore, BFP provides a balance between the dynamic range (benefit of floating point numerical format) and hardware complexity (benefit of fixed-point numerical format). While the BFP seems to be a promising approach for deep learning, the shared exponent in the BFP numerical format needs to be updated according to the DNN parameter statistics, thus increasing the computational complexity of deep learning training and inference [125].

Low-precision training: Several researchers have evaluated the efficacy of BFP numerical format for deep learning training as shown in Figure 3.3 and Table 3.3 [105, 118, 125–127]. For instance, Courbariaux *et al.* trained a low-precision DNN on the MNIST, CIFAR-10, and SVHN datasets with the floating point, fixed-point, and BFP numerical formats [105]. They demonstrated that BFP is the most suitable choice for low-precision training since it can capture the dynamic range of DNN parameters [105].

Following this work, Koster *et al.* proposed the 21-bit BFP numerical format (5-bit shared exponent, 16-bit fraction) re-branded as Flexpoint numerical format. Moreover, a new algorithm called Autoflex is proposed to automatically predict the

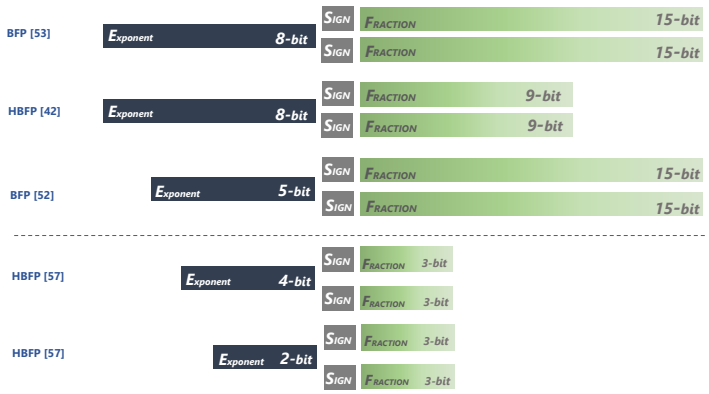

Figure 3.3: Low-precision Block floating point numerical format used in DNN Training. In BFP format, the exponent is shared between the block of floating point numbers. The bit width of shared exponent and fraction is the main difference between various approaches.

optimal shared exponent's value for DNN parameters in each iteration of SGD by statistically analyzing the values of DNN parameters in previous iterations [125]. On training a ResNet-18 model on CIFAR-10 dataset, the Flexpoint numerical format with the Autoflex algorithm demonstrates similar performance compared to 32-bit floating point [125]. However, the Autoflex algorithm selects shared exponents several times per training batch; this leads to large quantization errors where the exponent values are changed abruptly between blocks. To mitigate this problem, Drumond *et al.* applied the tiling approach (dividing the weights parameters to multiple tiles or blocks), and wider bit-width for intermediate accumulations with 16-bit BFP (8-bit shared exponent and 8-bit fraction) parameters [118]. They also showed that 8-bit (4-bit shared exponent and 4-bit fraction) fails to capture the variability between the dynamic range of parameters across blocks. To address this issue, Zhang *et al.* combined 8-bit BFP training with stochastic rounding and

Table 3.3: The DNN training performance using block floating point format, on CIFAR-10 and ImageNet.

Numerical Format	Bit-Configuration[1]								Approach[2]				CIFAR-10		ImageNet	
	W	A	G	E	Acc	UP	BN	Act	AEM	KS	SR	CA	ResNet-20	ResNet-50	ResNet-20	ResNet-50
BFP16-5 [125]	21	21	21	21	32	32	32	32	✓	×	×	×	-	-	-	-
BFP16-8 [126]	24	24	24	24	32	32	32	16	✓	×	✓	✓	-	-	-	75.77(+0.7)%
HBFP8-10 [118]	18	18	18	18	32	32	32	32	✓	×	✓	✓	-	-	-	76.12(+0.24)%
HBFP4-4 [44]	8	8	8	8	32	32	32	32	✓	×	✓	✓	-	-	68.57(-0.03)%	75.13(-0.02)%
HBFP3-4 [44]	7	7	7	7	32	32	32	32	✓	×	✓	✓	-	-	68.10(-0.50)%	73.98(-1.19)%
HBFP2-4 [44]	6	6	6	6	32	32	32	32	✓	×	✓	✓	-	-	63.10(-5.5)%	72.10(-3.07)%

[1] W: Weights; A: Activations, G: Gradients, E: Error, Acc: Accumulation, UP: Update, BN: Batch Normalization, Act: Activation
[2] AEM: Automated Exponent management, KS: Kahan Summation, SR: Stochastic Rounding, CA: Chunk-Based Accumulation

achieved similar training accuracy with 32-bit floating point [44]. **Although 8-bit BFP has shown a promising result in DNN training, the 4-bit BFP training did not converge [44] and is currently an open research question**.

Low-precision Inference: The BFP is also evaluated on DNN inference as shown in Figure 3.4 and Table 3.4. For instance, Song *et al.* introduced a 12-bit BFP (4-bit shared exponent and 8-bit fraction) to represent DNN parameters. Moreover, the analytical bound on the bit width of the BFP format is derived in this work. The result indicated that it is possible to classify ImageNet on the ResNet-50 model with less than 0.3% inference accuracy degradation compared to 32-bit floating point. Following this work, Rouhani *et al.* proposed a 12-bit BFP (8-bit shared exponent and 4-bit fraction) rebranded as a Microsoft floating-point (MSFP) and performed DNN inference on a variety of DNN models [41]. The outcome of this study showed that the BFP is more suitable for the transformer models due to a wider dynamic range of parameters compared to that of other DNN models [41]. One drawback of the MSFP format is that it can not represent non-uniform and skewed distribution due to the use of sign-magnitude fixed-point arithmetic. This drawback causes the quantization error to be increased when the

DNN parameter bit width is reduced to $\leq$ 8-bit [41]. **Therefore, DNN inference with 4-bit floating point numerical format is an active area of research.**

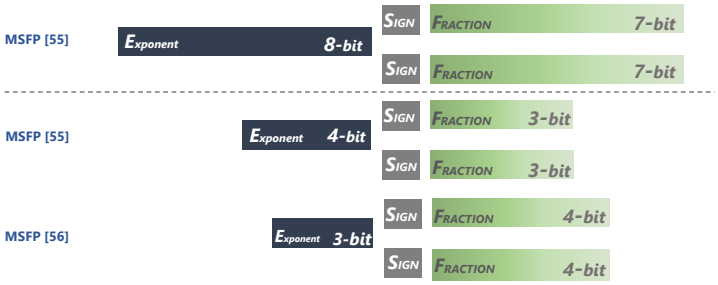

Figure 3.4: Low-precision block floating point numerical format used in DNN Inference. The exponent is shared in BFP format. Different approaches varied the bit width of shared exponent and fraction.

Table 3.4: The DNN Inference performance using block floating point format, on CIFAR-10 and ImageNet.

Numerical Format	Bit-Configuration[1]					Approach[2]				CIFAR-10		ImageNet	
	W	A	Acc	BN	Act	PQ	AEM	SR	CA	ResNet-20	ResNet-50	ResNet-20	ResNet-50
MSFP8-8 [41]	16	16	16	16	16	✓	×	×	✓	-	-	-	75.26(-0.00)%
BFP8-4 [128]	12	12	16	16	16	✓	×	×	✓	-	-	-	76.40(+0.27)%
BFP8-4 [129]	12	12	16	16	16	✓	×	×	✓	-	-	-	75.75(-0.38)%
MSFP4-8 [41]	12	12	16	16	16	✓	×	×	✓	-	-	-	72.77(-2.49)%

[1] W: Weights; A: Activations, Acc: Accumulation, BN: Batch Normalization, Act: Activation
[2] PQ: Partial quantization, AEM: Automated Exponent management, SR: Stochastic Rounding, CA: Chunk-Based Accumulation

3.1.3 Variants of fixed-point

Low-precision training: Training DNN models using fixed-point numerical formats is a challenging task due to the fixed-point narrow dynamic range that fails to capture the dynamic range of DNN gradients and quickly runs up against the quantization error. To mitigate this problem, several approaches have been proposed in the literature as shown in Figure 3.5 and Table 3.5, and a few of them

successfully trained DNN models with 8- and 16- bits fixed-point numerical format. For instance, Banner *et al.* proposed a hybrid 8- and 16-bit fixed-point representation where the activation gradients are represented with 16-bit precision. The reason behind the 16-bit representation of activation gradients is that the statistics of activation gradients do not follow the normal distribution, which contradicts the assumption of most of the quantization approaches [130]. However, with prior knowledge of the distribution of activation gradients and introducing a new batch normalization algorithm (Range BN), they successfully trained Resnet-50 model on imageNet with 0.4% accuracy degradation [130]. Following this work, Rajagopal *et al.* proposed a new algorithm to automatically switch the DNN parameter's precision every epoch based on a gradient diversity metric [131]. Specifically, the DNN is initially trained with the smallest precision (e.g., 8-bit precision), and the diversity of gradient d_g is evaluated on each epoch. Afterward, the precision of DNN parameters gradually increased from initial precision to 12-bit, 14-bit, and 16-bit fixed-point numerical formats when the d_g is decreased. The ResNet-18 is trained on the ImageNet using this approach with 0.39% accuracy degradation as compared to the same network with 32-bit floating point [7]. Recently, Zhao [132] *et al.* observed that the gradients over various channels have two different distributions (normal and log-normal distribution). Based on this observation, the gradients are quantized channel-wise, and the scale factor for quantization is calculated based on the distribution of gradients. By using this approach, the ResNet-50 model on ImageNet is trained without accuracy degradation. However, this approach requires to store the weights, activations, and gradients in 32-bit floating point to analyze

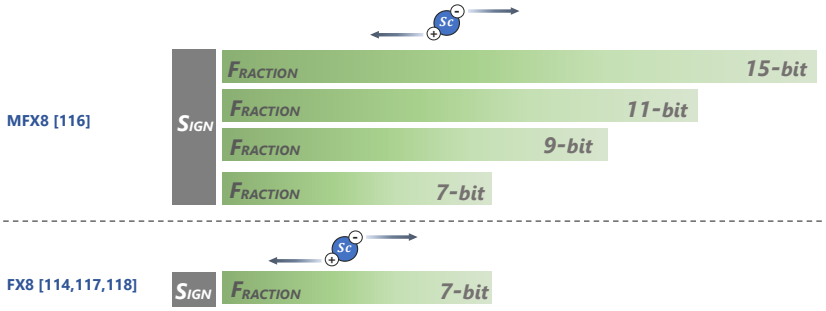

Figure 3.5: Low-precision fixed-point numerical formats used in DNN Training. The MFX8 represents gradients with 8-bit precision and increases the precision gradually during the training epochs based on the diversity of gradient metric [7].

Table 3.5: The DNN training performance using variants of fixed-point numerical format, on CIFAR-10 and ImageNet.

Numerical Format	Bit-Configuration[1]								Approach[2]				CIFAR-10		ImageNet	
	W	A	G	E	Acc	UP	BN	Act	LS	KA	SR	CA	ResNet-20	ResNet-50	ResNet-20	ResNet-50
FX8 [133]	8	8	8	8	32	32	32	32	✓	×	×	×	91.95(-0.32)%	-	69.67(-0.63)%	76.34(-0.26)%
FX8 [132]	8	8	8	8	32	32	32	32	✓	×	✓	×	92.76(+0.41)%	-	70.21(-0.01)%	76.59(+0.09)%
FX8 [130]	8	8	8	8	32	32	8	32	✓	×	×	×	-	-	65.10(-0.05)%	72.10(-0.10)%
MFX8 [7]	8	8	8	8	32	32	32	32	✓	×	✓	×	90.86(+0.78)%	-	69.09(-0.39)%	-

[1] W: Weights; A: Activations, G: Gradients, E: Error, Acc: Accumulation, UP: Update, BN: Batch Normalization, Act: Activation
[2] LS: Loss Scaling, KS: Kahan Summation, SR: Stochastic Rounding, CA: Chunk-Based Accumulation

their distributions used for quantization, and the accumulations are performed in 32-bit fixed-point.

Low-precision inference: Numerous studies in low-precision arithmetic investigate approaches to perform DNN inference with $\leq$ 8-bit fixed-point numerical format as shown in Figure 3.6 and Table 3.6 [40, 43, 84, 134–139]. For instance, Migacz introduced an 8-bit fixed-point quantization approach by scaling either down or up the 32-bit high-precision floating point parameters and then converting to the fixed-point number [84]. The Kullback–Leibler (KL) divergence between the high precision value and quantized value over the partial training dataset (calibration dataset) is used to find the appropriate scaling factor. The result of this

study showed a negligible accuracy degradation (0.13%) for the ResNet-50 model on the ImageNet dataset when 1250 training images were used as a calibration dataset [84]. Recently, this approach has been studied for a variety of DNN models. The outcome of this study showed that in some cases (e.g., EfficientNet-B0), the accuracy loss is substantial (4.79% accuracy loss) [40]. To mitigate this problem, the granularity of the scaling factor is reduced to vector size that is analogous to the BFP approach and has similar drawbacks such as large quantization error due to abrupt change in scaling factor between channels [134]. The alternative approach to perform DNN inference without managing the scaling factor is to represent DNN parameters with mixed-precision fixed-point numerical format [43, 135–137]. However, this approach requires a precision selection algorithm for each layer either based on first-order error information (Jacobian norm) [135], or the second-order error information (Hessian matrix) of DNN parameters [43, 136, 137]. For instance, Yao *et al.* used a Hessian matrix to compute the sensitivity of DNN parameters in each layer to quantization error and assigned the least number of bits for parameters of layers to minimize quantization error. The DNN using 4-bit and 8-bit mixed-precision fixed-point numerical format achieved a comparable accuracy compared to high-precision 32-bit floating point (2.53% accuracy degradation) [43]. **However, performing DNN inference using 4-bit fixed-point numerical format is still an open question**. The main reason the fixed-point fails to preserve accuracy is the narrow dynamic range of this numerical format. A few proposals are proposed to extend the dynamic range of fixed-point numerical format, such as dual fixed-point [140], and triple fixed-point numerical format [138].

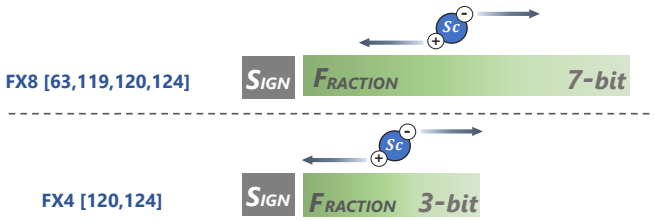

Figure 3.6: Low-precision fixed-point numerical format used in DNN Inference. In most approaches, quantization techniques are used to compensate for accuracy degradation due to the narrow dynamic range of fixed-point numerical format

However, the benefits of the 4-bit precision of these numerical formats are not evaluated on DNN inference.

Table 3.6: The DNN Inference performance using variants of fixed-point format, on CIFAR-10 and ImageNet.

Numerical Format	Bit-Configuration					Approach				CIFAR-10		ImageNet	
	W	A	Acc	BN	Act	PQ	AEM	SR	CA	ResNet-20	ResNet-50	ResNet-20	ResNet-50
FX8 [84]	8	8	32	32	32	✓	×	×	✓	-	-	-	73.10(-0.13)%
FX8 [40]	8	8	32	32	32	✓	×	×	✓	-	-	-	76.05(-0.11)%
FX8 [134]	8	8	32	32	32	✓	×	×	✓	-	-	-	75.15(-1.01)%
FX8 [43]	8	8	32	32	32	✓	×	×	✓	-	-	71.56(+0.09)%	77.58(-0.14)%
FX4 [134]	4	4	32	32	32	✓	×	×	✓	-	-	-	74.36(-1.80)%
FX4 [43]	4	4	32	32	32	✓	×	×	✓	-	-	68.45(-3.02)%	74.24(-3.48)%

[1] W: Weights; A: Activations, Acc: Accumulation, BN: Batch Normalization, Act: Activation
[2] PQ: Partial Quantization, AEM: Automated Exponent Management, SR: Stochastic Rounding, CA: Chunk-Based Accumulation

3.1.4 Posit

The capability to represent the non-uniformly distributed (bell-shaped) parameters by posit numerical format due to its tapered precision motivated us to evaluate the benefits of posit numerical format for DNN parameters for the first time in the literature. In particular, we demonstrated that the weights of a deep learning model's parameters, which are non-uniformly distributed (bell-shaped), can be most

suitably represented with posits instead of uniformly distributed fixed-point [50]. The outcome of this study indicates that deep learning network weights can be represented by 7-bit posits to achieve inference accuracy similar to that of 32-bit floating point (with 1% variation). After this work and relevant to this thesis research, the extensive studies in understanding the benefits and overheads of posit numerical format for deep learning by researchers in academia and industries [58, 123, 141–145, 145–150, 150–157]. In this section, we only present a summary of previous works on deep learning with posit that offer different viewpoints compared to the algorithms presented in this thesis.

Low-precision training: Few studies have proposed a framework to evaluate the efficacy of 16-bit and 8-bit posit numerical format for DNN training as shown in Figure 3.7 and Table 3.7 [145, 148–151, 158]. Unfortunately, the scope of these studies is either limited to the use of posit for memory operations by representing DNN parameters in 8-bit posit formats (the computation is performed in 32-bit floating point) [148, 150] or has been only evaluated on low-dimensional datasets, such as MNIST [149]. For instance, Lu *et al.* demonstrated that the memory footprint is reduced between $2\times$ to $4\times$ when training ResNet-18 model with 8-bit posit while achieving an accuracy similar to 32-bit floating point. Their approach also relies on the warm-up method (initially for some epoch trained with 32-bit low-precision floating point) that increases the training overhead [148]. Recently, Goncalo Raposo *et al.* evaluated the DNN training using 8-bit posit for both memory and computation operations. They successfully trained a LeNet-5 model on the MNIST dataset without accuracy degradation. However, the scope of this

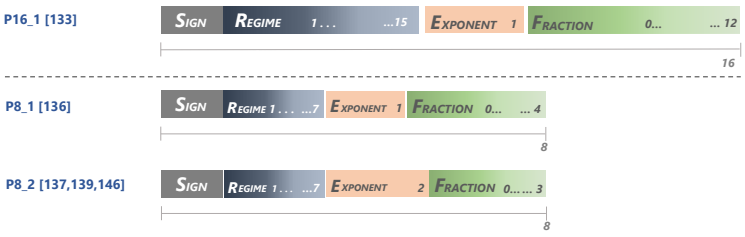

Figure 3.7: Low-precision posit numerical formats are used in DNN Training. The most successful approach allocates 2 bits for the exponent.

Table 3.7: The DNN training performance with variants of fixed-point numerical format, on CIFAR-10 and ImageNet.

Numerical Format	Bit-Configuration[1]								Approach[2]				CIFAR-10		ImageNet	
	W	A	G	E	Acc	UP	BN	Act	LS	Q	MO	WU	ResNet-20	ResNet-50	ResNet-20	ResNet-50
P16_1 [145]	16	16	16	16	16	16	16	16	×	×	×	×	-	-	-	-
P8_1 [148]	8	8	8	8	32	16	16	32	✓	×	✓	✓	92.07(-0.14)	-	70.83(-0.11)	-
P8_2 [158]	8	8	8	8	32	32	32	32	✓	×	✓	✓	89.70(-1.30)	-	-	-
P8_2 [158]	8	8	8	8	32	32	32	32	✓	×	✓	✓	91.60(0.60)	-	-	-
P8_2 [149]	8	8	8	8	32	12	8	32	×	✓	×	×	-	-	-	-
LP8_2 [151]	8	8	8	8	20	32	32	32	✓	×	✓	×	93.10(-0.3)	-	70.10(-0.8)	-

[1] W: Weights; A: Activations, G: Gradients, E: Error, Acc: Accumulation, UP: Update, BN: Batch Normalization, Act: Activation
[2] LS: Loss Scaling, Q: Quire, MO: Memory Only, WU: Warm-up

study was limited to the low-dimensional dataset, and small DNNs [149]. Finally, Nhut-Minh Ho *et al.* proposed the Qtorch+ framework as an extension to the QPytorch framework to represent weights, activations, and gradients with a hybrid 8-bit posit and 32-bit floating point format. Based on the result of this study, it is possible to train the DNN models with this numerical format similar to 32-bit floating point as long as the gradient scaling approach is used. However, the arithmetic operations for DNN models in this work are performed in the 32-bit floating point. **Therefore, training DNN models with posit numerical format that perform both memory and arithmetic operations with either hybrid $\leq$ 8-bit and 32-bit precision or hybrid $\leq$ 8-bit and 16-bit precision is an active research area.**

Low-precision inference: Deep learning inference with low-precision posit has been studied in literature, as shown in Figure 3.8 and Table 3.8 [58, 123, 145–147, 156]. For instance, Jeff Johnson proposed a logarithmic form of posit arithmetic [58]. On the ImageNet dataset, 8-bit log-posit performed within 0.9% inference accuracy of 32-bit floating point. The inference performance on DNNs with 10-bit posits has also shown similar performance in autonomous driving applications, such as the CIFAR-10 benchmark [146]. The new posit-based exponential linear unit (ELU) activation function was proposed to achieve this performance in this work. Recently, the DNN inference with $\leq$ 8-bit posit formats has been explored, to further decrease the hardware complexity of performing DNN inference [123, 158]. The results of these studies showed that 8-bit low-precision DNN inference, combined scaling factor achieves similar performance to 32-bit floating point DNN inference on the Imagenet dataset. However, further reducing the bit precision to less than 8-bit causes a significant drop in accuracy [123, 158]. **Therefore, performing DNN inference using $\leq$ 8-bit posit numerical format is still an open question**.

3.2 Numerical error analysis of low-precision

Low-precision Inference: Studies considering low-precision arithmetic have experimentally shown that neural networks using 8-bit numbers can achieve inference accuracy comparable to that of 32-bit numbers. However, few studies have proposed numerical analysis frameworks that explain the function between the

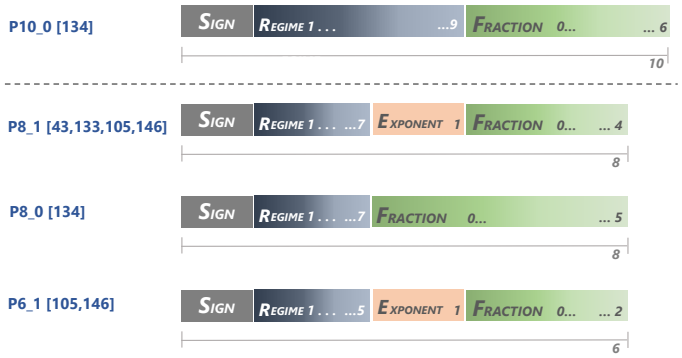

Figure 3.8: Low-precision posit numerical format used in DNN Inference. In most cases, an exponent bit is enough to represent the DNN parameters.

Table 3.8: The DNN Inference performance using posit format, on CIFAR-10 and ImageNet.

Numerical Format	Bit-Configuration					Approach				CIFAR-10		ImageNet	
	W	A	Acc	BN	Act	PS	Q	SR	CA	ResNet-20	ResNet-50	ResNet-20	ResNet-50
P10_0 [146]	10	10	32	32	32	✓	×	×	×	-	93.7(0.0)%	-	-
LP-8_1 [58]	8	8	38	8	32	✓	✓	×	×	-	-	-	75.23(-0.90)%
P8_1 [145]	8	8	32	32	32	✓	×	×	×	-	-	-	-
P8_0 [146]	8	8	32	32	32	✓	×	×	×	-	85.00(-13.7)%	-	-
P8_1 [123]	8	8	32	32	32	✓	×	×	×	-	-	-	75.40(-0.80)
P8_1 [158]	8	8	32	32	32	✓	×	×	×	-	-	-	75.70(-0.40)
P6_1 [123]	6	6	32	32	32	✓	×	×	×	-	-	-	68.80(-7.40)%
P6_1 [158]	6	6	32	32	32	✓	×	×	×	-	-	-	69.10(-7.00)%
P4_1 [123]	4	4	32	32	32	✓	×	×	×	-	-	-	0.70(-75.70)%

[1] W: Weights; A: Activations, Acc: Accumulation, BN: Batch Normalization, Act: Activation
[2] PS: Parameter Scaling, Q: Quire, SR: Stochastic Rounding, CA: Chunk-Based Accumulation

precision of numerical formats and the inference accuracy [135, 159–162].

To find such a function theoretically, Holt *et al.* analyzed the error generated by finite-precision computation in shallow neural networks [159]. In this study, the low-precision floating point parameters are quantized by either a truncation or rounding operation. They prove that the expected square error $[e_y^2]$ and q-bit precision for each layer in a feedforward neural network follows a function

presented by (3.1).

$$q = \frac{1}{2}\log_2([e_y^2]) + \frac{1}{2}\log_2(12) \tag{3.1}$$

To extend this work for DNNs with fixed-point arithmetic, Lin *et al.* proposed an analytic solution for the optimal bit-width and range by calculating the SQNR of all DNN parameters [161]. They proved that the overall DNN performance is indicated by the *Harmonic Mean* of the SQNR of all DNN parameters.

However, the optimization of DNN parameters' bit precision based on SQNR tends to require multiple bit precisions, which increases the hardware complexity. To solve this drawback, Sakr *et al.* derived two theoretical upper bounds for low-precision DNN inference by calculating the mismatch classification rate between low-precision fixed-point and high-precision floating point [135]. The mismatch probability p_m has an upper bound given by (3.2), where E_A and E_W represent the maximum tolerable error of a network with respect to the different activation and weight quantization, and B_A and B_W indicate the activation and weight precision, respectively. Based on the upper theoretical bound, they found a double-exponential relationship between the bit precision of DNN parameters and the performance of DNNs during inference:

$$p_m \leq 2^{-(B_A-1)}E_A + 2^{-(B_W-1)}E_W \tag{3.2}$$

They also analytically determined that the precision of the weights has a similar impact on convolutional neural networks and a larger impact in feedforward neural networks than that of activations. A similar theoretical approach is explored to

determine the minimum per-layer precision of DNNs in [162]. Recently, Lauter *et al.* proposed a semi-theoretical approach by using interval and affine arithmetic to maintain the accuracy of neural networks with low-precision floating point [160]. The result from this study showed that the rounding error generated from low-precision linear operations in feedforward and convolutional layers is remedied by the activation function.

Low-precision training: Most of the current work highlights theoretical bounds for DNN inference. Furthermore, to realize the function between DNN training performance and bit-precision, Li *et al.* proposed the convergence bound with convex assumption given by (3.3), where the objective $f(w'_T) - f(w^*)$ is expected error, σ^2_{max} is an upper bound on the second moment of the stochastic gradient samples, Δ is a quantization step, and d is the dimension of the DNN model [85]. Note that the expected error is calculated by the difference between the low-precision DNN output and the output of the true global minimizer after T iterations. With this convergence bound, they theoretically showed that the approximate number of bits needed to perform DNN training equals the log of DNN parameters' dimension (e.g. the DNN models with 10^8 parameters require 13-bit training precision). However, dependence on dimensionality is undesirable in deep learning applications [163]. Thus, to find a dimension-free upper bound for low-precision training with floating point parameters, Li *et al.* modified the $f(w'_T) - f(w^*)$ with L_1 norm term instead of L_2 norm term. Based on this approach, the number of bits needed for $\mathbb{E}[f(w'_T) - f(w^*)] < \in$ is given by $\log_2 \mathcal{O}(R \times \frac{\sigma_1}{\varepsilon})$

where R indicates the radius of the range of DNN parameters [53].

$$\mathbb{E}[f(w'_T) - f(w^*)] \leq \frac{(1 + log(T+1)) \times \sigma_{max}^2}{2\mu \times T} + \frac{\sqrt{d} \times \Delta \times \sigma_{max}}{2} \tag{3.3}$$

3.3 Summary

In this chapter, an overview of the state-of-the-art low-precision numerical formats (variants of IEEE floating point, block floating point, fixed-point, and posit) for deep learning training and inference is presented. The benefits and drawbacks of numerical formats in terms of inference and training accuracy and hardware complexity are highlighted. To perform DNN training, the current state-of-the-art approaches utilize mixed-precision numerical formats from [8, 16, 32] precision set where the majority of operations are usually performed in the least precision numerical format. Evaluation of these numerical formats on DNN training has revealed that the significant challenge to achieving high training and inference accuracy is the mismatch between the dynamic range of DNN parameters and numerical format, and accumulation of quantization error. The approaches such as loss scaling, Kahan summation, stochastic rounding, Kulisch accumulator, and chunk-based addition are used in the literature to mitigate this problem. However, further reducing the bit precision to less than 8-bit is a challenging task and, in most cases, requires high-radix and logarithmic number systems that increase the computational complexity. **Therefore, DNN training using numerical formats with $\leq$ 8-bit is an active research area.** To perform DNN inference, the mixed precision 8-bit and 4-bit fixed-point numerical formats have been commonly

used compared to other numerical formats. The prior studies concluded that compared to high-precision 32-bit floating point, the DNN utilizing 4-bit and 8-bit mixed-precision fixed-point numerical formats obtained slight accuracy degradation (2.53%). **However, performing DNN inference with ≤ 8-bit numerical format is still an active research area**. To successfully perform DNN training and inference with ≤ 8-bit precision, we introduce the posit numerical format for deep learning for the first time in the literature. Extensive studies from academia and industry indicated that the ≤ 8-bit posit numerical format is appropriate for DNN training and inference. Finally, to the best of our knowledge, no numerical error analyses are devoted to the posit format that we will study through this thesis.

4. Tapered-Precision numerical formats for deep learning inference

Compressing the DNN parameters using a low-precision numerical format is one of the efficient approaches that enable us to deploy deep learning models on edge devices. Unfortunately, this compression can jeopardize performance as a consequence of the discrepancy between the expressible values of low-precision numbers and the unaltered high-precision DNN parameters. Multiple low-precision numerical formats have been proposed, as summarized in Chapter 3, as replacements for the 32-bit high-precision floating point formats. However, most of these numerical formats either have failed to achieve similar performance to the 32-bit floating point or require being combined with approaches such as mixed-precision quantization [37] and numerical format scaling [123] to reduce the computational complexity. Tapered-precision numerical formats offer unequal-magnitude spacing (tapered accuracy), and a flexible dynamic range that is appropriate to represent the DNN model. Therefore, in this chapter, the efficacy of various tapered-precision numerical formats is comprehensively studied for deep learning inference. The

characteristics of various proposed frameworks are summarized in Table 4.1. The tapered-precision numerical formats for deep learning inference are explained through Cheetah-V2 and ALPS frameworks that support all features of other frameworks.

Table 4.1: Frameworks for Tapered-Precision Numerical Formats for Deep Learning Inference

Framework	Numerical format[1]					Neural network[2]			Implementation[3]			Approach	
	P	GP	TFX	FP	FX	FF	DCNN	RNN	SW	HW	SW/HW	Experimental	Numerical
Mem-Posit [50]	✓	×	×	×	✓	×	✓	×	✓	×	×	✓	×
Deep Positron [56]	✓	×	×	✓	✓	✓	×	×	✓	✓	×	✓	×
PositNN [61]	✓	×	×	✓	✓	×	✓	×	✓	×	×	✓	×
Cheetah [62]	✓	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×
Adaptive-Posit [63]	✓	✓	×	×	×	×	✓	×	✓	✓	×	✓	×
ALPS [65]	✓	✓	×	✓	×	×	✓	×	✓	✓	×	✓	✓
TENT [64]	×	×	✓	×	✓	×	✓	×	✓	✓	×	✓	×
ACTION [66]	✓	✓	✓	✓	✓	×	✓	×	✓	✓	✓	✓	×
Cheetah-V2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×

[1] P: Posit; GP: Generalized Posit, TFX: Tapered fixed-point, FP: Floating-point, FX: Fixed-point
[2] FF: Feed-Forward, DCNN: Deep Convolutional Neural Network, RNN: Recurrent Neural Network
[3] SW: Software, HW: Hardware, HW/SW: Hardware/Software co-design.

4.1 Empirical approach: Cheetah-V2 framework

The goal of Cheetah-V2 framework is threefold. First, evaluate the efficacy of tapered-precision numerical formats in terms of accuracy and hardware complexity for deep learning inference. Second, empirically understand the correlation between hardware complexity and inference performance accuracy of deep learning models with tapered-precision numerical formats. Third, select the appropriate tapered-precision numerical format based on accuracy and hardware complexity constraints provided by practitioners. As shown in Table 4.1, the current version of Cheetah-V2 supports five numerical formats (fixed-point, floating point, posit,

generalized posit, tapered fixed-point) and three DNN models feedforward neural networks, convolutional neural networks, and recurrent neural networks for Deep learning inference. The framework supports a range of 5- to 32-bit precision for DNN inference. Moreover, the framework supports seven important metrics for the evaluation of tapered-precision numerical formats, including power, energy, energy-delay product (EDP), MAC frequency, area, memory footprint, and accuracy.

This framework comprises four key aspects, as shown in Figure 4.1: User Interface, Initialization, Optimizer, and Evaluator. The evaluator is composed of Inference Accuracy and Hardware Complexity Evaluator aspects.

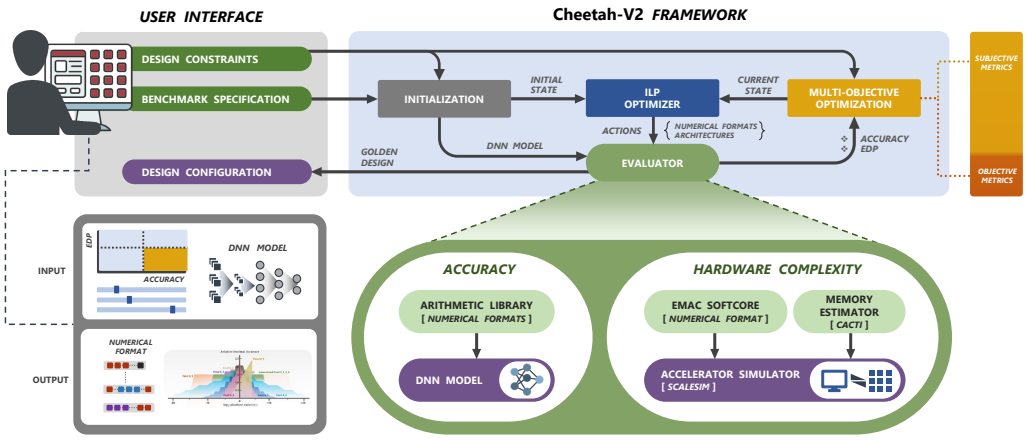

Figure 4.1: The Cheetah-V2 High-level low-precision Hardware & Software Co-design framework for DNN models on edge platforms

4.1.1 User interface

The goal of the user interface is to preselect the benchmark specification and design constraints given as input to the framework. These include benchmark specifications (applications, datasets, deep learning models), evaluation metrics (e.g., accuracy and energy), constraints (e.g., 90% accuracy and 10 mJ energy consumption), and variables (e.g., numerical format configurations).

The framework generates optimal numerical format configurations to satisfy user constraints as an output file. The input and output of the interface are specified in YAML format.

4.1.2 Initialization

The trained weights are needed to perform DNN inference. Therefore, in the initialization step, the model is trained with 32-bit floating point values. The high-precision 32-bit floating point trained weights and activations are transferred to the evaluator.

4.1.3 Inference accuracy evaluator

The inference accuracy evaluator is a two-level software framework that is used to evaluate the performance of various numerical format configurations by emulating low-precision DNN inference. The Evaluator unit is explained with the help of an example of a single hidden layer convolutional neural network to highlight its key components and operations clearly, although it can be generalized for any

DNN models. Note that the low-precision numerical format parameters (e.g., *es* bit-width) are required to be predetermined prior to performing the computation in a single hidden layer convolutional neural network.

Low-precision numerical format parameter selection

Posit parameters selection: No specific algorithm is proposed to select the parameters in posit numerical format. In the evaluation step, all possible options for $n \in \{5,6,7,8\}$ and $es \in \{0,1,2\}$ are tested.

Generalized posit parameters selection: Algorithm 7 presents the *rs* and E_b optimization procedure. The selection of *rs* and E_b is governed by the dynamic range of DNN parameters. As mentioned in Chapter 2, E_b recenters the location of maximum accuracy (the mean of posit values distributions in the log domain) from 2^0 to 2^{e_b}. Therefore E_b is selected in a way that the mean of DNN parameters in the log domain matches the maximum accuracy of the posit value distribution. After selecting E_b, *rs* is selected based on the dynamic range of the DNN distribution (lines 6–10). In most cases, the dynamic range of DNN parameters is captured with the dynamic range of generalized posits by fixing *es* value and varying *rs*. In the worst case scenario, when the dynamic range of DNN parameters is larger than the dynamic range of the generalized posit numerical format, *es* could be increased to compensate for the dynamic shortage coverage by generalized posit.

Tapered fixed-point parameters selection: Algorithm 8 in the appendix A presents the *IS* and *SC* optimization procedures. This algorithm 8 defines the process by selecting *IS* an *SC* to adjust the numerical format range into the range of

parameters in each individual layer. To select these parameters in the first step, the maximum absolute value of the DNN parameters at each layer (W_{amax}, and A_{amax}) is computed (lines 2–3) and rounded up to generate the appropriate *IS* value. In the worst case scenario, when the range of DNN parameters (e.g, W_{amax}) is larger than the range of tapered fixed-point (*IS*), the maximum possible value of n, where n is the total number of bits represented in tapered fixed-point format, is selected for *IS* (lines 4–13). Selecting *IS* based on the algorithm 8 reduces the overflow error in quantization. However, when the maximum absolute value of a DNN parameter (e.g, W_{amax}) is less than the maximum absolute value of the tapered fixed-point format representation (*IS*), many bit-patterns in the numerical format are unused. To address this issue, the tapered fixed-point format values are scaled down by 2 raised to the power of *SC*. The *SC* parameter is determined by base-2 logarithm of w_{amax} (lines 14–19).

Low-precision arithmetic operations

As mentioned in Chapter 2, a computational node in a single hidden layer convolutional neural network computes (4.1) where B indicates the bias vector, W is the weight tensor with numerical values that are associated with each connection, A represents the activation vector as input value to each node, θ is the activation function, Q denotes the quantization function, Y is a feature vector consisting of the output of each node, and M is equal to the product of (C,R,S) filter parameters; where (C,R,S) are the number of filter channels, the filter heights, and the filter weights respectively. The computation in (4.1) is performed N times, where N is a

product of batch size, output activation size (height and width), and the number of filters.

$$Y_j = \theta(Q(B_j) + \sum_{i=0}^{M} Q(A_i) \times Q(W_{ij})) \tag{4.1}$$

At the first step, each 32-bit floating point DNN parameter (x_i) is mapped to an l-bit low-precision numerical format value (x_i') through the quantization function as defined in (4.2), where s and z are the scaling factor and zero point, respectively. Given the low-precision numerical format values range ($[l,u]$), the large magnitude 32-bit floating point numbers that are not expressible in this range are clipped either to the format lowerbound (l) and upperbound (u). Moreover, the clipped values that lie in the interval $[a,b]$ (the two consecutive low-precision numerical format values) are rounded to the nearest even number (RNE(x_i)). [1]

$$x_i' = Q(x_i, q, l, u, s, z) = \text{RNE}(\text{Clip}(s \times x_i + z, l, u)) \tag{4.2}$$

Following that, the MAC operation over quantized weights and activations inputs is employed to calculate Y_j. To minimize arithmetic error, the MAC operation in this thesis is calculated using the EMAC algorithm. The EMAC algorithm varies among various numerical formats. In this section, the EMAC algorithm for posit, generalized posit, and tapered fixed-point is explained [2].

Posit EMAC: The Posit EMAC procedure can be found in Algorithm 9 in the

[1] Algorithms 2, 4, and 6 in the appendix A can be used to quantize 32-bit floats to posit, generalized posit and tapered fixed point encoding

[2] The EMAC algorithm for fixed-point and floating point is behind the scope of this thesis topic. More details are provided in [57]

appendix A. In the posit EMAC algorithm, to preserve precision in computing the products, the posit weights and activations are multiplied in a posit format without truncation or rounding at the end of multiplications. To avoid rounding during accumulation, the products are stored in a wide register (*quire*), with a bit-width given by (4.3). Note that the quire bit-width is adaptive (depend on N_{op} and es) and different than the fixed bit-width allocated for quire in standard posit format as mentioned in Chapter 2. Following that, the products are converted to the fixed-point format $FX_{(m_k,n_k)}$, where $m_k = 2^{es+1} \times (n-2) + 2 + \lceil \log_2(N_{op}) \rceil$ is the exponent bit-width and $n_k = 2^{es+1} \times (n-2)$ is the fraction bit-width. Finally, the N_{op} fixed-point products are accumulated, and the result is converted back to posit format.

$$w_q = \lceil \log_2(N_{op}) \rceil + 2^{es+2} \times (n-2) + 2 \tag{4.3}$$

Generalized posit EMAC: The generalized posit EMAC is presented in Algorithm 10. In the first step, a set of quantized weights and activations is 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 product is then stored in a wide signed fixed-point register, the *quire* [6], for m additions with size $w_{quire} = \lceil \log_2(m) \rceil + 2 \times \lceil \log_2(\frac{\text{Max}_{GP}}{\text{Min}_{GP}}) \rceil + 2$ (lines 11–14). The stored product is then converted and accumulated using fixed-point arithmetic. Finally, the accumulated result is converted back to the generalized posit numerical format (lines 15–17).

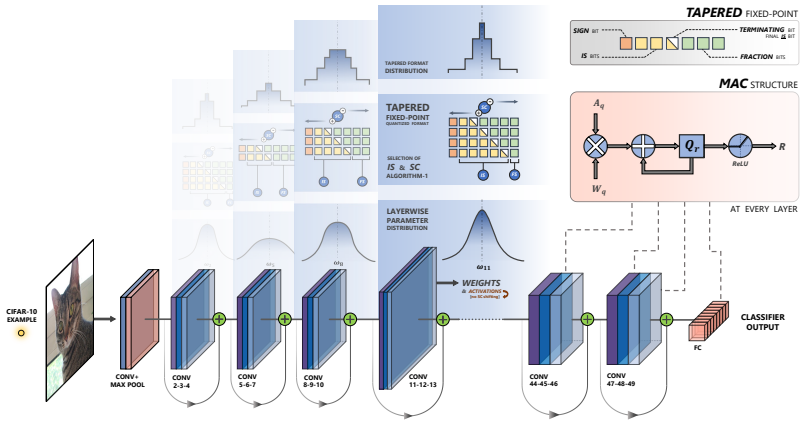

Figure 4.2: an example of DNN inference accuracy evaluator with tapered fixed-point parameters. The evaluator is applied to each layer individually, selecting specific IS and SC values to match the distribution and range of parameters within the layer. IS specifies the *maximum* number of integer bits, and SC specifies the degree of shift required (left-shift if positive, right-shift if negative). The MAC structure displays the tapered fixed-point EMAC unit explained in this section

Tapered fixed-point EMAC: The tapered fixed-point EMAC is presented in Algorithm 11. In the first step, a set of quantized weights and activations is decoded to the tapered fixed-point (lines 2–3). To decode the tapered fixed point format, the sign bit, integer bits (through the leading zero detection algorithm), and remaining fractional bits require to be extracted (lines 4–7). Then, the products of the tapered fixed-point weights and activations are calculated without truncation or rounding after multiplication operations (lines 8–9). The products are then stored in a wide register (*quire* [6]) for m multipliers with the size of w_{quire} as in (4.4) (lines 10–12).

$$w_{quire} = \lceil \log_2(m) \rceil + 2 \times \lceil \log_2(D_l) \rceil + 2 \tag{4.4}$$

An example of an inference accuracy evaluator is illustrated in Figure 4.2.

4.1.4 Hardware complexity evaluator

The hardware complexity evaluator is a hardware framework that combines Cacti [164] and SCALE-Sim simulators [165]. The hardware framework is a soft-core implementation of EMAC operations on FPGA/ASICs and is used for evaluating hardware characteristics of the low-precision EMAC operations in various numerical formats as a fundamental computation in DNN models. Most existing frameworks such as [166] evaluate the performance of the numerical formats by comparing only the energy consumption of the MAC operations which overlooks the constraints imposed by memory and dataflow in the accelerator. However, in this thesis, we measure the hardware metric through a DNN accelerator that is simulated using SCALE-Sim software [165].

The high-level architecture of the DNN accelerator is used to evaluate the hardware complexity of DNN inference with various numerical formats, as shown in Figure 4.3. This architecture is guided by Eyeriss v2 [167] design. Primarily, it is composed of processing elements (PEs) arranged in a 2D systolic array architecture and a hierarchical memory organization. This architecture adopts three dataflows including input stationary (IS), weight stationary (WS), and output stationary (OS). The PE in the systolic array includes float, posit, generalized posit, fixed point, and tapered-fixed point MAC unit with configurable bit-precision and parameters. For modeling the memory hierarchy, a 128 MB off-chip DRAM and a 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.

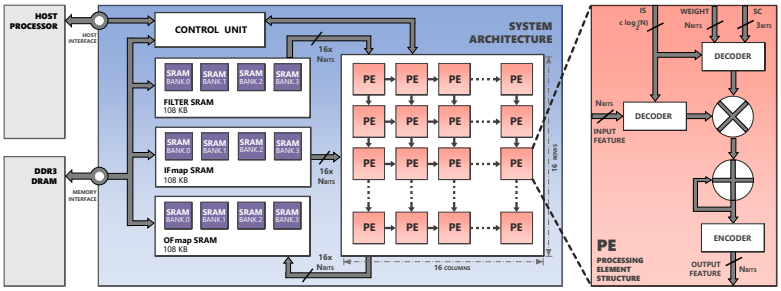

Figure 4.3: Deep Neural Network accelerator architecture with custom tapered fixed-point processing elements. The architecture is evaluated in a full cycle-emulator to analyze the performance and energy constraints.

4.1.5 Optimization

In the Cheetah-V2 framework, we use integer linear programming (ILP) to find the optimal numerical format configuration. In this thesis, the ILP is defined as (4.5), where y_i, x_j, and A are, respectively, objective metrics, subjective metrics, and configuration sets (e.g., sets of numerical format configurations), selected from Table 5.2. C_j are constraints with respect to subjective x_j,

$$\begin{aligned} \min_{y_i} \quad & \sum_{i=0}^{k=1} y_i(A) \\ \text{s.t.} \quad & x_j(A) < c_j, 0 < j < 6 \end{aligned} \tag{4.5}$$

For instance, the accuracy degradation (ACC_d) is selected as an objective. Area, EDP, memory footprint, and MAC frequency are chosen as subjective metrics, as in (4.6). In this study, we set the maximum number of subjective metrics to 4 from 6 proposed subjective metrics since some of the subjective metrics overlap with each other, such as power and EDP. Note that there is no limitation on the number of subjective metrics. However, the probability of finding the optimal solution by

the ILP approach decreases as the number of subjective metrics increases.

$$\begin{aligned} \min_{ACC_d} \quad & ACC_d(A) \\ \text{s.t.} \quad & \text{EDP}(A) < \text{EDP Constraint} \\ & \text{Area}(A) < \text{Area Constraint} \\ & \text{Memory footprint}(A) < \text{Memory footprint. Constraint} \\ & \text{MAC Frequency}(A) < \text{MAC Frequency Constraint} \end{aligned} \tag{4.6}$$

4.2 Numerical analysis approach: ALPS Framework

The goal of the ALPS framework is to formalize the relationship between quantization error of high-precision floating point DNN parameters and low tapered-precision numerical format. Our goal is to derive a function $\mathscr{F}$ that approximates the misclassification probability, p_m, as given by (4.7).

$$p_m \approx \mathscr{F}\left(\varepsilon(w_i, w_i'), \varepsilon(A_i, A_i')\right) \tag{4.7}$$

where the function $\varepsilon(\cdot,\cdot)$ 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 low tapered-precision weights and activations respectively. Figure 4.4 conceptually illustrates the relationship described by (4.7) where a cat image is misclassified by ResNet-50 as a dog due to the errors generated in each layer and accumulated in the last layer as $\varepsilon(w_n, w_n')$ due to low-precision quantization process.

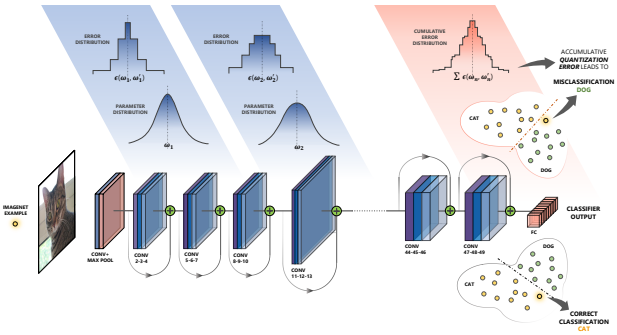

Figure 4.4: The impact of quantization error in inference misclassification with ResNet-50. An image sample is misclassified as a dog instead of a cat due to the accumulation of weight quantization error.

In particular, the ALPS numerical analysis framework comprises two key aspects: SQNR and Finite Precision Error Analysis.

4.2.1 SQNR for tapered-precision numerical formats

SQNR for generalized posit: Defining the SQNR for generalized posits requires modeling the quantization error $\varepsilon(x_i, x_i') = \sum_{i=0}^{m} |x_i - x_i'|$ [168], 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 quantized DNN parameters with low-precision generalized posit. This model depends on the distribution of the values represented by this numerical format. Low-precision generalized posit numerical format has a non-uniform distribution, which is modeled as a non-uniform quantizer with compander system [169], as shown in Figure 4.5. The compander system contains a monotonic smooth non-uniform *compressor* function, a fixed-point quantizer, and an inverse function of the *compressor* function (*expander*

function). In this chapter, the *compressor* and *expander* are approximated by (4.8) and (4.9), respectively,

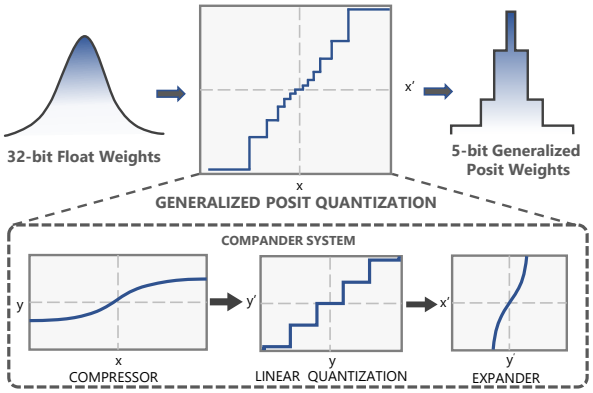

Figure 4.5: Generalized posit quantization model with a fixed-point quantizer, *compressor* $y=\frac{1}{\gamma}\sinh^{-1}(\theta x)$, and *expander* $x' = \frac{1}{\theta}\sinh((\gamma)y')$ functions. The α, β, γ, and θ are real variables that are varied for different generalized posit configurations

$$\begin{aligned} y &= \sum_{i=0}^{m} (\tfrac{1}{\alpha}x + \text{sign}(x)\beta\Delta) \times 1_{[i\Delta,(i+1)\Delta)}(|x|) \\ &\approx \sum_{i=0}^{m} \tfrac{1}{\gamma}\sinh^{-1}(\theta x) \times 1_{[i\Delta,(i+1)\Delta)}(|x|) \end{aligned} \tag{4.8}$$

$$\begin{aligned} x' &= \sum_{j=0}^{m} (\alpha(y' - \text{sign}(y')\beta\Delta) \times 1_{[j\Delta,(j+1)\Delta)}(|y'|) \\ &\approx \sum_{j=0}^{m} \tfrac{1}{\theta}\sinh^{-1}(\gamma y') 1_{[j\Delta,(j+1)\Delta)}(|y'|) \end{aligned} \tag{4.9}$$

where α, β, γ, and θ are real variables, Δ is the quantization step size, m represents the quantization levels, and $1_{[a,b)}(|x|)$ is the indicator function as given by (4.10). Note that prior works model the quantization error by using this approach for

non-uniform float formats [168]. The rationale for choosing the sinh function as an expander is to approximate the distribution of the generalized posit derivatives (spacing between two numbers), which maintain a constant value (equal to one) near zero and taper towards infinity for the maximum representable number. The $tanh^{-1}$ function was not chosen as an expander because its derivative rapidly approaches infinity for inputs larger than two, which contrasts with the distribution of the generalized posit derivatives. While a more accurate (optimal) function for the expander might exist in comparison to the sinh function, no mathematical approach currently exists to find the optimal function [168].

$$1_{[a,b)}(|x|) = \begin{cases} 1, & \text{if } (a \leq |x| < b) \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

While modeling the quantization error for generalized posit, the SQNR is computed as in (4.11) where Δ_{GP} (quantization step size) is computed as (4.12). The derivation of (4.11) is provided in Appendix A .

$$\text{SQNR}_{GP}(dB) \approx (10.79 - 20\log(\gamma)) + 20\log(\Delta_{GP}^{-1}) \tag{4.11}$$

$$\Delta_{GP} = \begin{cases} 2^{-(2^{es}rs - 2^{es-(n-rs-1)}) + sc}, & \text{if } (n - rs \leq es + 1) \\ 2^{-(2^{es}rs - es + (n-rs-1)) + sc}, & \text{otherwise} \end{cases} \tag{4.12}$$

The proposed SQNR captures the several 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. The parameter *es* controls the dynamic range and width of the max SQNR interval ($[2^{-2^{es}}, 2^{2^{es}}]$). The parameter *rs* adjusts the shape of the SQNR distribution and also controls the dynamic range.

SQNR for posit: The definition of the posit SQNR is similar to the generalized posit SQNR except the Δ_{GP} in (4.11) is changed to Δ_P, as in (4.13).

$$\text{SQNR}_P(dB) \approx (10.79 - 20\log(\gamma)) + 20\log(\Delta_P^{-1}) \tag{4.13}$$

$$\Delta_P = 2^{-(2^{es}(n\text{-}2))} \tag{4.14}$$

SQNR for tapered fixed-point: The tapered fixed-point SQNR in (4.15) is defined similar to posit SQNR except $es = 0$ and Δ_P in (4.13) is changed to Δ_{TFX}, as in (4.15).

$$\text{SQNR}_{TFX}(dB) \approx (10.79 - 20\log(\gamma)) + 20\log(\Delta_{TFX}^{-1}) \tag{4.15}$$

$$\Delta_{TFX} = 2^{-((n\text{-}2))} \tag{4.16}$$

4.2.2 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 (4.17), where quantization error is considered throughout the network with an

activation function T [170].

$$y + \varepsilon_y = T_n(\cdots(T_2(T_1(w_1 + \varepsilon_{w_1} \times A_1 + \varepsilon_{A_1}) + \varepsilon_{A_2}) \times (w_2 + \varepsilon_{w_2})) \tag{4.17}$$

For simplicity, we use the shorthand ε_x to represent the error $\varepsilon(x, x')$ for some arbitrary x. For neural network models with a non-linear T_i, the error is calculated as in (4.18) by using the chain rule of partial derivatives and approximated as the first-order Taylor series [159].

$$\varepsilon_y \approx \sum_{i=1}^{n} \varepsilon_{w_i} \frac{\partial T_i}{\partial w_i} + \sum_{i=1}^{n} \varepsilon_{A_i} \frac{\partial T_i}{\partial A_i} \tag{4.18}$$

Finally, by calculating the probability ε_y for any y_i [135, 161], we can formulate the relationship between the mismatch probability and numerical precision as in (4.19), where E_{W_l} and E_{A_l} are weight and activation quantization error gains at layer l [135, 161].

$$p_m \leq \sum_{l=1}^{L} \frac{1}{\mathrm{SQNR}_{w_l}} E_{W_l} + \frac{1}{\mathrm{SQNR}_{A_l}} E_{A_l} \tag{4.19}$$

Given this equality, it helps us to find a theoretical upperbound on classification accuracy of DNN inference with various tapered-precision numerical formats. Moreover, we can use this equality to select an appropriate generalized posit numerical format configuration for a specified p_m. This use case of the equation 4.19 is elaborated in the next section.

4.2.3 ALPS framework use-case

Generalized Posit parameter selection

As mentioned before, the performance of the DNN using the low-precision generalized posit depends on the p_m and the SQNR, as illustrated in (4.19). To improve the performance of DNN model with low-precision numerical format, the p_m requires to be minimized, which can be accomplished by balancing the summation in the equation (4.19) [162]. This computes *rs* and *sc* from the equations (4.20), (4.21), (4.22), and (4.23), which are inspired from [162].

$$rs_{w_l} = \text{RNE}\left(\log_2\sqrt{\frac{E_{w_l}}{E_{w_r}}}\right) + rs_{w_r} \tag{4.20}$$

$$rs_{A_l} = \text{RNE}\left(\log_2\sqrt{\frac{E_{A_l}}{E_{w_r}}}\right) + rs_{w_r} \tag{4.21}$$

$$sc_{w_l} = \text{RNE}\left(\log_2\sqrt{\frac{E_{w_l}}{E_{w_r}}}\right) + sc_{w_r} \tag{4.22}$$

$$sc_{A_l} = \text{RNE}\left(\log_2\sqrt{\frac{E_{A_l}}{E_{w_r}}}\right) + sc_{w_r} \tag{4.23}$$

In these equations, RNE is the round-to-the-nearest-even function and E_{w_r} is the quantization error gain for weights of the reference layer (selected randomly). Algorithm 12 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). As altering the *rs* parameters causes the distribution of the posit numerical

format to shift from approximately uniform to near-Normal and near-Laplace distributions [171], excess kurtosis is employed to differentiate between these distributions. It should be noted that the excess kurtosis is calculated over the small dynamic range of the generalized posit numerical format (similar to the dynamic range of weights) in order to minimize the impact of extremely large outliers that would result in very high kurtosis values.

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. Then, the generalized posit numerical format configuration which has the closest excess kurtosis to that of the DNN parameters is selected (lines 7–16). A similar procedure is applied to compute sc_{w_r}, except that the mean of the DNN weights is used as a metric to select sc_{w_r}.

4.3 Summary

In this chapter, the proposed tapered-precision frameworks (Cheetah-V2 and ALPS) and adaptive tapered-precision algorithms are summarized. Through Cheetah-V2 frameworks, the practitioners can evaluate and compare the efficacy of tapered-precision numerical formats such as generalized posit, posit, and fixed-point in comparison with traditional numerical formats (fixed-point and floating point) in terms of accuracy and different hardware metrics, such as power consumption, on various DNN models. In addition, the trade-off between hardware metrics and performance accuracy for 5- to 8-bit precision of each model and dataset can be

reported. Moreover, the Cheetah-V2 framework can be used by practitioners and startups as an EarlyDSE (early stage design space exploration) framework that identifies design variables, including the numerical format specification, to discover the optimal numerical formats and accelerator configurations based on custom user-defined constraints. The configuration selection problem in this framework is solved by integer linear programming (ILP), which can reduce the cost of the AI accelerator design by identifying the optimal numerical format and accelerator configuration faster than common approaches such as reinforcement learning approaches.

In the second part of this chapter, the novel tapered-precision framework, ALPS, is presented which has the ability to adapt the generalized posit configurations to the layerwise dynamic range and distribution of parameters within a DNN model. Adaptation in this framework is achieved by estimating the hyper-parameters in generalized posit and minimizing the quantization error while maximizing the SQNR in each layer. To define the generalized posit SQNR function, the generalized posit quantization is modeled by a compressor function, expander function, and a fixed-point quantizer. This model is inspired by the quantization model of a compander system. To formulate the function between the SQNR of posit and the performance of DNNs, finite precision error analysis is used. This approach for adapting the generalized posit configurations ultimately enhances the performance of DNN inference with this numerical format. A similar approach can be used for other adaptive tapered-precision numerical formats, such as the tapered fixed-point that is explored in the TENT framework.

5. DNN inference results and discussion

Having described the algorithms and frameworks that enable us to efficiently use the tapered-precision format for deep learning inference and provide the optimal numerical formats to match with edge device characteristics in chapter 4, this chapter discusses the 16 benchmarks that are used to evaluate the efficacy of various tapered-precision formats in the Cheetah-V2 framework. Moreover, we summarize the performance of various tapered-precision numerical formats in these benchmarks. The specifications of the tasks and inference performance with 32-bit floating point DNNs are summarized in Table 5.1.

5.1 Benchmark specification

5.1.1 Datasets & pre-processing

Fashion-MNIST [172]: Fashion-MNIST dataset is comprised of 70000 images of 28x28 grayscale pixels to classify 10 various clothes. The test set of this dataset includes 10000 images. The test set is normalized in a range between zero to one.

CIFAR-10 [173]: The CIFAR-10 dataset contains images of 10 various cate-

Table 5.1: The DNN models and benchmarks using 32-bit float parameters description.

Application	Dataset	DNN/RNN Model	# Parameters	Performance
Image classification	Fashion-MNIST	4 FC[1]	0.34 M	89.51±0.25%
		2 Conv,[1] 3 FC, 2 PL,[1] 1 BN[1]	1.88 M	92.54±0.26%
	CIFAR-10	ResNet-8	78.67 K	86.26±0.30%
		ResNet-18	0.27 M	91.54±0.23%
		ResNet-50	0.86 M	92.10±0.24%
		EfficientNet-B0	4.00 M	98.00±0.36%
	ImageNet	ResNet-18	11.70 M	68.10±0.52%
		ResNet-50	25.00 M	74.60±0.68%
Keyword Spotting	Speech Commands v2	DS-CNN[2]	24.91 K	92.15±0.41%
Visual Wake Words	VWW Dataset	MobileNet-V1	221.79 K	82.72±0.47%
Activity classification	DogCentric	2 V-RNN,[3] 1 FC	1.58 M	50.78 ±0.65 %
		2 LSTM, 1 FC	6.30 M	63.79 ±0.76 %
		2 GRU, 1 FC	4.73 M	61.97 ±0.80%
NLP	PTB	2 V-RNN, 1 FC	4.17 M	PPW[4]= 164.36 ±1.21
		2 LSTM, 1 FC	4.65 M	PPW= 114.25±0.91
		2 GRU, 1 FC	3.72 M	PPW= 117.64±0.87

[1] FC: Fully Connected layer, Conv: Convolutional layer, PL: Pooling, BN: Batch Normalization layer
[2] DS-CNN: Depthwise separable convolutional neural network
[3] V-RNN: Vanilla RNN
[4] PPW: Perplexity per word

gories, such as airplanes, and automobiles which are collected from the web. In this dataset, 10000 images are assigned to the test set. The test set is normalized in a range between zero to one.

ImageNet [174]: The ImageNet dataset contains images of 1000 categories, that is, image classes built through WordNet's hierarchical structure. In this dataset, 50000 images are assigned to the test set. The images in the test set are resized to 256x256 pixels with cubic spline interpolation. Afterward, the images are center cropped to 224x224 pixels.

Speech commands v2 [175]: The speech commands v2 dataset includes 105,829 utterances from 2,618 speakers. The dataset is labeled with 12 command classes (e.g., "yes", "no", "up", and "down"). In this dataset, 4800 utterances

are selected for the test set. The features from the speaker's speech are extracted with a sample rate of 16000 audio samples per second and 1000 millisecond audio length. Other audio feature extractions preprocessing are adapted from MLPerf™ Tiny deep learning benchmarks (V0.5) [176].

Visual wake words [177]: The application of this dataset is person detection for smart doorbell devices [176]. The visual wake words dataset is built upon the COCO 2014 dataset [177]. The images in the COCO dataset are labeled as a person or no person. The test set of this dataset includes 8059 images. The images are resized to 96x96 dimensions [176].

DogCentric dataset [178]: DogCentric dataset contains 208 videos recorded by cameras attached to the backs of dogs while they performed 10 different classes of activity (e.g., playing with a ball or sniffing an object). The test set of this dataset is 107 videos. The videos in DogCentric do not have a fixed length. We chose to use a fixed sequence length of 160 frames for simplicity. Shorter videos received zero padding, and longer videos were truncated to provide a consistent 160 frames. The frames are resized and center cropped to 224x224 pixels.

PTB dataset [179]: The English Penn Treebank (PTB) corpus (the section of the corpus corresponding to the articles of the Wall Street Journal) is one of the most known and used corpora for the evaluation of models for sequence labeling. The task consists of annotating each word with its Part-of-Speech tag. The vocabulary used in this dataset is 10k words. The 82430 tokens are allocated for the test set.

5.1.2 Experiment setup

The Cheetah-V2 framework is implemented in C++ and Python languages and extended to the TensorFlow framework [180]. A summary of the metrics and numerical configurations are used in different benchmarks is presented in Table 5.2. Note that the generalized posit and tapered fixed-point hyperparameters ($rs, Is \in \{1, 2, \cdots, 7\}$, and $e_b, sc \in \{-4, -3, \cdots, 3\}$) are not mentioned in the Table 5.2 since these values are predetermined based on algorithms 7 and 8 in the appendix A. Additionally, there are constraints that need to be defined by the users. However, as an example to show the performance of Cheetah-V2 framework in the selection of numerical format, the constraint was derived by adding the mean with the standard deviation of the metric. Note that the input features belonging to recurrent neural networks are provided through the embedding layer [181], and off-the-shelf CNN features using pre-trained ResNet-50 and VGG-16 neural networks [182] for video activity classification. Moreover, to estimate latency, we bridge our framework with the SCALE-Sim tool [183]. 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.

Table 5.2: The metrics, variables and search space configuration

Metrics	Accuracy, EDP, Energy, Power, MAC frequency, Area, Memory footprint
Variables	Posit ($n \in \{5,6,7,8\}$,es$\in\{0,1,2\}$)
	Generalized Posit ($n \in \{5,6,7,8\}$,es$\in\{0,1,2\}$)
	Tapered Fixed-Point ($n \in \{5,6,7,8\}$)
	Floating point ($n \in \{5,6,7,8\}$,e$\in\{3,4,\cdots,n-2\}$)
	Fixed-Point ($n \in \{5,6,7,8\}$,f$\in\{1,2,\cdots,n-1\}$)
Search Space	60

5.2 Tapered precision numerical formats performance

The efficacy of tapered-precision numerical formats is evaluated for DNN and RNN inference using Cheetah-V2 framework as shown in Tables 5.3 and 5.4, and Figures 5.1 and 5.2 (corresponding to Tables A.1, and A.2 in the appendix). The best accuracy is selected among 5 to 8 bits formats with a varying of the *es*, *e*, and f parameters (as shown in Table 5.2) for both generalized posit and posit, floating point, and fixed-point formats, respectively.

On Fashion-MNIST, the findings show that the low-precision generalized posit (with es=1) outperforms other numerical formats in terms of accuracy with fewer bits. In particular, it is possible to perform inference on a 4-layer fully connected neural network with 6 bits parameters using generalized posit numerical format without accuracy degradation in comparison to the performance of the same network with 32-bit floating point. Achieving similar accuracy with low-precision posit, floating point, and tapered fixed-point requires 7, 8, and 8 bits respectively. Moreover, the performance of DNNs using 5-bit floating point and fixed-point

Table 5.3: The DNN inference performance using the tapered-precision numerical formats on Fashion-MNIST dataset (P: posit, FP: floating point, FX: fixed-point, GP: generalized posit, TFX: tapered fixed-point).

Format	Fashion-MNIST (FC)				Fashion-MNIST (Conv)			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	89.59±0.18%	89.44±0.22%	89.24±0.21%	88.14±0.25%	92.70±0.21%	92.60±0.27%	91.64±0.30%	88.92±0.32%
FP	89.56±0.23%	89.36±0.26%	88.92±0.28%	83.00±0.31%	92.63±0.29%	92.22±0.25%	89.58±0.30%	68.21±0.42%
FX	89.16±0.27%	87.27±0.24%	85.20±0.31%	83.97±0.35%	89.59±0.28%	88.63±0.32%	85.31±0.35%	83.46±0.40%
GP	**89.68±0.14%**	**89.65±0.13%**	**89.58±0.17%**	**89.21±0.26%**	**92.75±0.19%**	**92.63±0.25%**	**92.32±0.28%**	**91.65±0.30%**
TFX	89.66±0.19%	89.59±0.22%	89.38±0.28%	89.02±0.32%	92.59±0.26%	92.47±0.31%	92.14±0.33%	89.35±0.39%
32-bit FP	89.51±0.25%				92.54±0.26%			

is reduced significantly in comparison to generalized posit since these numerical formats are not able to capture the dynamic range of parameters (e.g., it is not possible to represent 5-bit floating point with four exponent bits). We also observed that the performance of tapered fixed-point is not only better than fixed-point, but also, on average, comparable with floating point and posit formats. This finding emphasizes that tapered fixed-point could be a good candidate for applications and models where a trade-off between performance and hardware complexity is desired.

On CIFAR-10 dataset, amongst the evaluated numerical formats, generalized posit shows the best performance. For instance, the inference accuracy on classifying images using ResNet-8 model with generalized posit is improved by an average of 11.55%, 28.79%, and 16.28%, as compared to posit, floating point, and tapered fixed-point, respectively. Moreover, in this dataset, the inference performance gains are further noticeable with ultra-low bits generalized posit (e.g., 5-bit generalized posit having 16.53%, 31.93%, and 25% improvements over posit, floating point and tapered fixed-point, respectively. The high performance of the generalized posit numerical format on image classification benchmarks can be credited to

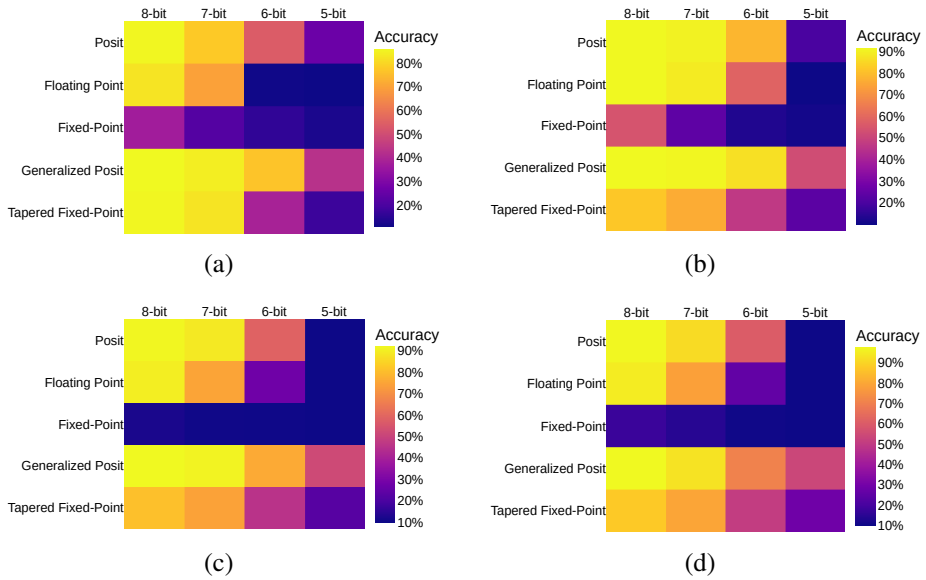

Figure 5.1: The DNN inference performance using the tapered-precision numerical formats on the CIFAR-10 dataset. (a) ResNet8; (b) ResNet18; (c) ResNet50; (d) EfficientNet-B0

the capability of this numerical format to auto-adjust to the dynamic range and distribution of the weights and activations. On the other hand, as the number of bits is decreased to 7-bits and below, the floating point, fixed-point, and tapered fixed-point formats show poor performance accuracy. This can be attributed to the discrepancy between the dynamic range provided by these numerical formats and the actual dynamic range of weights and activations. Finally, in contrast to the DNN inference results on Fashion-MNIST dataset, achieving the similar performance of the 32-bit floating point baseline requires representing DNN parameters with 8-bit generalized posit numerical formats.

On Speech commands v2 dataset, amongst all five numerical formats, only generalized posit and posit numerical formats have shown promising results. For

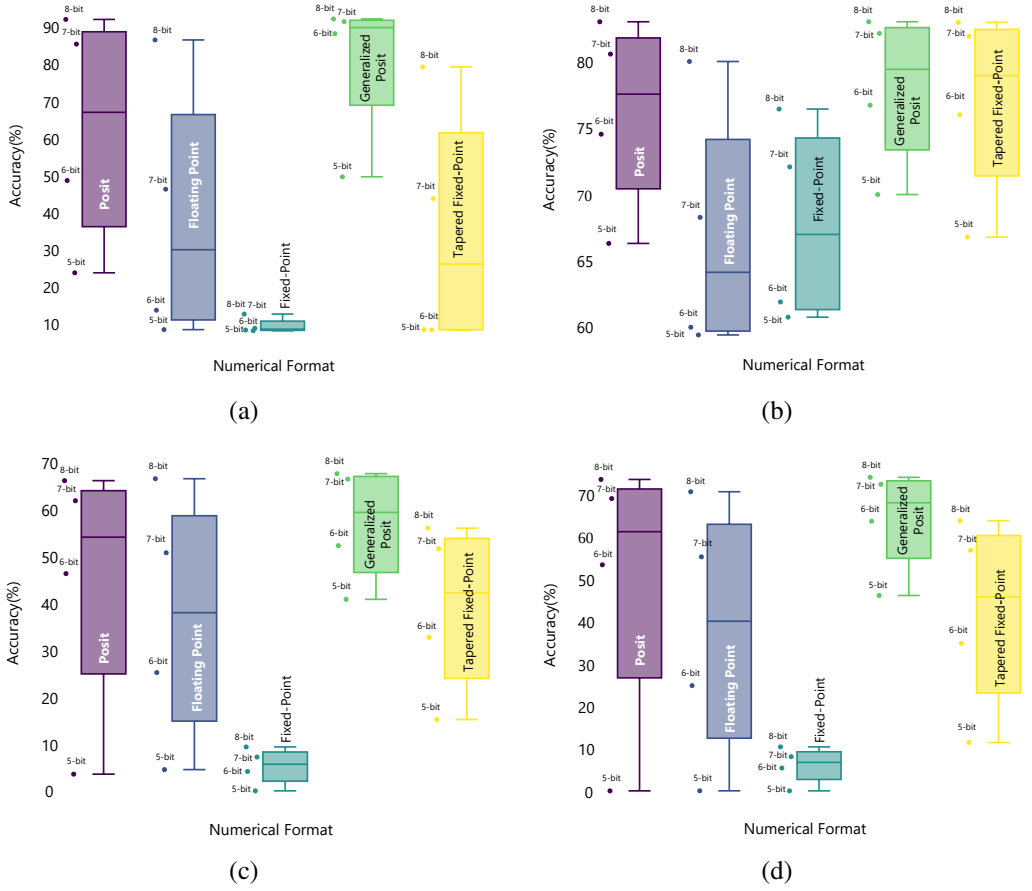

Figure 5.2: The DNN inference performance using the tapered-precision numerical formats on Speech commands v2, Visual Wake Word, and ImageNet datasets. (a) Speech commands v2 (DS-CNN); (b) Visual Wake Word (MobileNetv1); (c) ImageNet (ResNet18); (d) ImageNet (ResNet50).

instance, the 7-bit generalized posit DNN parameters representation is sufficient to achieve inference accuracy within 1% variation of 32-bit floating point. Moreover, the performance of a 6-bit low-precision generalized posit depthwise separable convolutional neural network on this dataset is boosted by 45.10%, compared to the float-based network. On Visual Wake Word dataset, generalized posit performs

uniformly well from 8- to 6-bit precision and demonstrates improvement over floating point. An interesting result is that generalized posit, posit, and tapered fixed-point at 8-bit precision improve upon the 32-bit floating point baseline performance for this task. These performance benefits can be intuitively explained by the quantization noise generated from the use of low-precision parameters in neural networks for binary image classification acts as a regularization method [102].

On ImageNet dataset, the findings show that the low-precision generalized posit outperforms other numerical formats on various benchmarks. For instance, the performance of a 7-bit low-precision generalized posit ResNet-18 network is improved by 4.6%, 15.7%, and 14.8% compared to the posit, float, and tapered fixed-point based ResNet-18 networks, respectively. Furthermore, we observed that the generalized posit shows greater benefits on DNN models whose parameters have a large dynamic range, such as ResNet model. These performance benefits can be intuitively explained by the auto-tuning capability of the Cheetah-V2 framework, which adapts the format to the dynamic range of the weights and activations, to reduce the quantization error. The best performance observed on all the benchmarks (when analyzed across the full 5- to 8-bit range) is achieved with generalized posit.

On temporal and spatio-temporal datasets such as PTB, and DogCentric datasets, the results demonstrate that ultra-precision generalized posit with (mostly $es = 1$) surpasses other numerical formats on most of the tasks and RNN models. For instance, the performance of deploying RNN inference using GRU on the Dog-Centric dataset is improved by 31.52%, and 10.11% with 5-bit generalized posit in

Table 5.4: The RNN inference performance using various numerical formats (P: posit, FP: floating point, FX: fixed-point, GP: generalized posit, TFX: tapered fixed-point).

Format	DogCentric (Vanila-RNN)				DogCentric (LSTM)			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	50.80±0.66%	50.78±0.71%	50.46±0.69%	48.59±0.78%	64.31±0.74%	64.25±0.77%	62.31±0.75%	55.64±0.81%
FP	50.78±0.72%	50.70±0.79%	50.07±0.83%	23.79±0.88%	64.39±0.74%	63.10±0.78%	59.17±0.83%	34.81±0.89%
FX	50.25±0.74%	43.91±0.79%	36.20±0.82%	26.86±0.84%	63.32±0.73%	62.66±0.68%	53.87±0.81%	45.98±0.90%
GP	**50.85±0.64%**	**50.82±0.60%**	**50.63±0.71%**	**49.06±0.76%**	**64.71±0.69%**	**64.40±0.76%**	**63.27±0.78%**	**57.92±0.82%**
TFX	50.70±0.70%	50.33±0.73%	49.72±0.77%	30.65±0.78%	63.98±0.72%	62.97±0.80%	58.23±0.84%	48.34±0.86%
32-bit FP	50.78±0.65%				63.79±0.76%			

Format	DogCentric (GRU)				PTB (Vanila-RNN)			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	62.03±0.73%	61.94±0.76%	61.86±0.88%	59.19±0.85%	164.92±1.25	164.98±1.30	170.49±1.32	191.58±1.28
FP	62.05±0.81%	61.99±0.78%	61.02±0.85%	29.35±0.92%	165.22±1.36	166.87±1.40	175.51±1.42	390.83±1.40
FX	60.46±0.83%	58.27±0.89%	50.50±0.94%	45.40±0.92%	167.42±1.40	183.19±1.43	238.05±1.46	375.18±1.51
GP	**62.05**±0.75%	**62.03**±0.81%	**61.94**±0.83%	**60.87**±0.78%	**164.40**±1.19	**164.92**±1.24	**166.22**±1.30	**175.41**±1.33
TFX	63.98±0.82%	62.97±0.85%	58.23±0.80%	48.34±0.88%	165.71±1.31	170.42±1.36	185.63±1.40	220.31±1.42
32-bit FP	61.97±0.80%				164.36±1.21			

Format	DogCentric (LSTM)				DogCentric (GRU)			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	115.52±0.88	117.98±0.91	128.37±0.87	181.99±0.95	117.64±0.86	118.02±0.95	122.86±1.01	164.41±1.00
FP	114.89±0.94	116.15±0.92	119.45±0.99	137.41±1.05	117.66±0.88	119.02±0.93	125.16±0.96	191.99±1.04
FX	134.70±1.00	167.46±1.06	288.90±1.11	524.69±1.15	127.10±0.98	151.37±1.04	188.50±1.06	460.62±1.13
GP	**114.36**±0.87	**115.32**±0.93	**117.91**±0.88	**128.43**±0.91	**117.66**±0.85	**117.98**±0.92	**120.86**±0.91	**140.27**±1.01
TFX	120.61±0.91	124.38±0.96	140.70±1.00	200.32±1.03	124.61±0.97	130.59±0.98	142.48±1.06	205.03±1.08
32-bit FP	114.25±0.91				117.64±0.87			

comparison to 5-bit floating point, and tapered fixed-point, respectively. However, as an interesting result for the same task, the performance of LSTM based RNN inference with floating point numerical format outperforms posit. The dynamic range of weights in LSTM($\sim$[-8,8]) could intuitively explain this observation where posit always shows better performance where most of RNN parameters are distributed near zero [6]. For instance, the posit performance gains are also observed on the DogCentric dataset over floating point and fixed-point, respectively.

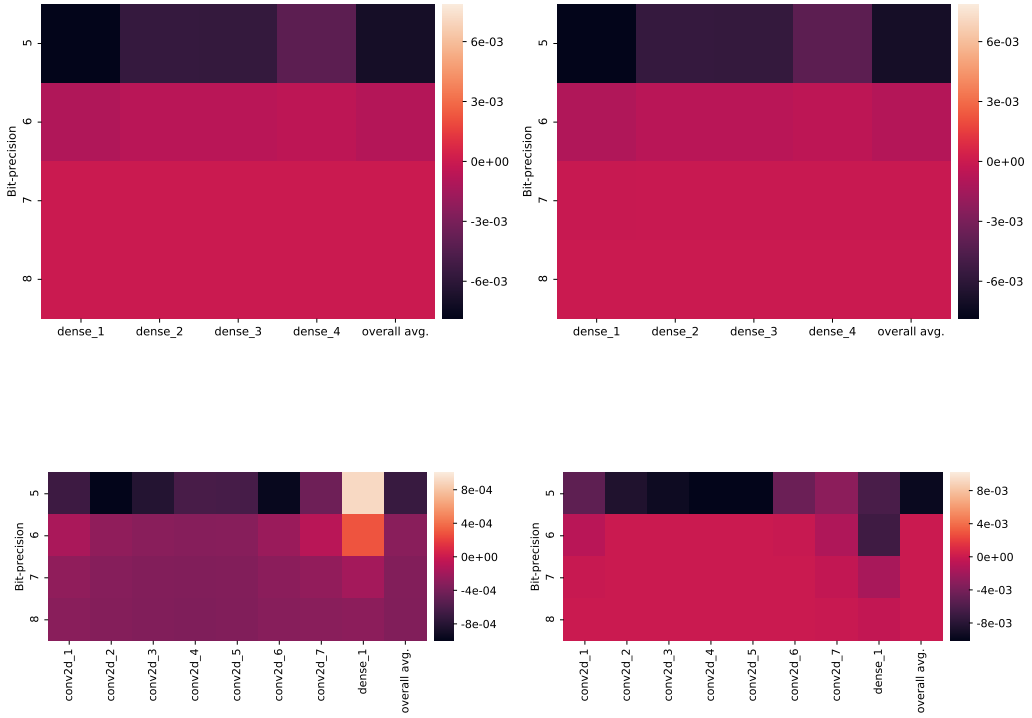

Figure 5.3: Layer-wise delta distortion rate $\Delta(d(R))$ heatmaps compare the precision (rates) of 5 to 8-bit numerical formats for representing 32-bit floating point DNN parameters. The average $\Delta(d(R))$ among all weights in a DNN is shown in the final column of each heatmap. (a) $\text{d(R)}_{posit} - \text{d(R)}_{fixed}$ for the Fashion-MNIST task; (b) $\text{d(R)}_{posit} - \text{d(R)}_{float}$ for the Fashion-MNIST task; (c) $\text{d(R)}_{posit} - \text{d(R)}_{fixed}$ for the Fashion CIFAR-10 task; (d) $\text{d(R)}_{posit} - \text{d(R)}_{float}$ for the CIFAR-10 task.;

5.3 Empirical quantization error analysis

The common metric to empirically analyze the quantization error generated during converting parameter values represented with a 32-bit floating point to low-precision numerical formats is the distortion rate ($d(R)$), which is defined by (5.1) as the average square distance between the actual parameters (P_i) and quantized

parameters ($Quant(P_i)$). This metric can be used to compare different numerical formats. Figure 5.3 demonstrates the distortion rates ($d(R)$) of weights of fully connected layers on Fashion-MNIST and convolutional layers on CIFAR-10 datasets with various formats. It is clear that posit has the least distortion rate, and thus it suffers the least consequences from quantization, which is especially noticeable at 5-bit precision.

$$d(R) = \ d(P, Quant(P)) = \frac{1}{n}\sum_{i}^{n} ||P_i, Quant(P_i)||_2 \tag{5.1}$$

5.4 Numerical analysis of quantization error

5.4.1 SQNR impact on DNN accuracy

As aforementioned in Chapter 4, the SQNR has a linear relationship with accuracy (as shown in Figure 5.4). The range of weights is mostly centered at zero and tapered to a dynamic range of two as mentioned in Table 5.1. Since posit has maximum SQNR in this range, the DNN accuracy using posit outperforms floating point 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 posit surpasses the floating point, and in a few cases floating point surpasses posit. Therefore, it is valuable to have a format, such as generalized posit, where the SQNR is variable and can be matched to the variability of activations.

5.4.2 Theoretical vs. experimental performance:

Figure 5.4(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 posit. This theoretical bound depends on the SQNR of weights and activations and is given by (4.19). Since the SQNR of generalized posit is related to the precision, Δ_{GP}, the misclassification grows exponentially when the precision is decreased (Figure 5.4(b)). Overall, the theoretical bound is shown to approximate the misclassification rate in most cases.

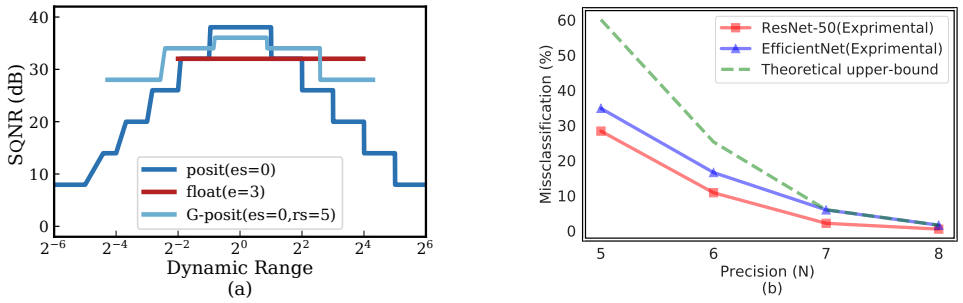

Figure 5.4: (a) The SQNR of 8-bit generalized posit compared to 8-bit posit and 8-bit floats. (b) ImageNet misclassification rate as a function of the generalized posit bit-precision with optimal *rs* for two DNNs and the theoretical upper bound (as formalized in (4.19)).

5.5 Hardware system results

5.5.1 EMAC hardware complexity vs performance accuracy

To expose the effectiveness of tapered-precision numerical formats over floating point and fixed-point when considering both classification accuracy and hardware cost, we evaluate the trade-off between the hardware complexity metric of the EMAC operation (as mentioned in Table 5.2) and average accuracy degradation from 32-bit floating point per bit-width across the multiple models on MNIST, Fashion-MNIST and CIFAR-10 datasets for the Cheetah-V2 framework. For instance, Figure 5.5 shows the trade-off between EDP and accuracy on across the multiple models on these datasets.

The results indicate that posit achieves up to 23% average accuracy improvement over fixed-point. However, this accuracy enhancement is gained at the cost of a 0.41×10^{-10} increase in energy-delay-product to implement the EMAC unit.

Posit also consistently shows better performance, especially at 5-bit width, compared to the floating point number system at a slight increase in the energy-delay-product. This slight cost of generalized posit numerical format is mainly due to the latency and power associated with decoding and encoding hardware. Overall, the 6-bit posit shows the best trade-off between energy-delay-product and average accuracy degradation from 32-bit floating point on the two benchmarks (when analyzed across the 5- to 8-bit range).

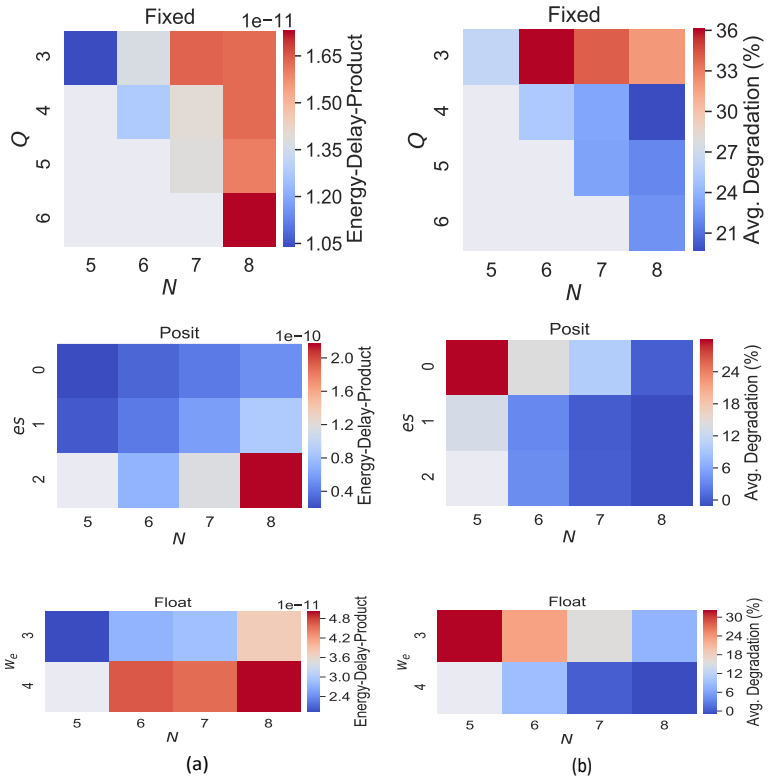

Figure 5.5: (a) The average accuracy degradation for 32-bit floating point across three classification tasks *vs.* the energy-delay-product of the respective multiplier and accumulator unit (MAC). (b) The average accuracy degradation for 32-bit floating point across three classification tasks vs. the latency of the respective MAC. N is total bits, Q is fractional bits for the fixed-point numerical format, es and w_e are the number of bits allocated for exponent in posit and floats, respectively.

5.5.2 Exploiting the posit es parameter

Experimental results in this chapter are evaluated by exploiting the performance of generalized posit and posit numerical formats by using $es \in \{0,1,2\}$ across benchmarks. As is shown in Figure 5.5, the energy-delay-product of the posit EMAC is dependent upon the es parameter. For instance, the energy-delay-product of the posit EMAC with $es = 0$, on average, is $3\times$ and $1.4\times$ less than the energy-

delay-product of the posit EMAC with $es = 2$ and $es = 1$, respectively. On the other hand, the average performance of DNN inference with $es = 1$ for the posit EMAC among the five datasets and 5- to 7-bit precision is 2% and 4% better than with $es = 2$ and $es = 0$, respectively. Thus, the posit ($es = 1$) EMAC has a better trade-off between energy-delay-product and accuracy for 5 to 7 bits. For 8-bit, the results suggest that $es = 1$ is a better fit for energy-efficient applications and $es = 2$ for accuracy-dependent applications.

5.5.3 DNN inference hardware complexity vs performance accuracy

The execution time of the DNN model is mainly governed by the dataflow and the PE array architecture. Output stationary dataflow has been shown to offer a 24% reduction in latency as compared to weight stationary dataflow in performing one inference. This is a significant improvement for a compute-bound DNN, as inference favors latency over throughput [184]. The homogeneous 16×16 PE configuration offers an improvement in computing efficiency from 89.58% to 91.82%, with a significant reduction in energy consumption. Figure 5.6 illustrates the energy-delay product (EDP) for ResNet-50 with posit while performing inference on the ImageNet dataset. It is worth noting that generalized posit offers in a 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 5.6 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 posit and posit.

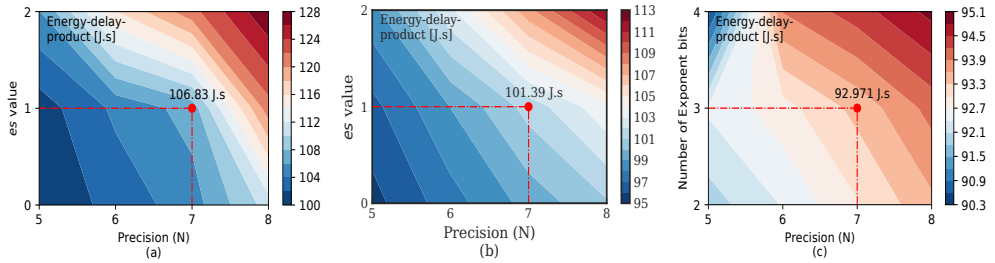

Figure 5.6: Energy-delay product of ResNet-50 benchmarked with ImageNet when using generalized posit (a), posit (b) and floating point(c). The performance of posit is evaluated for different bit widths, by changing number of exponent bits (*es*), to represent the weights and activations.

5.6 Numerical format identification based on user constraints through ILP

Figures 5.7 and 5.8 illustrate the performance of each numerical format incorporated into the different configurations of accelerator and dataflows. The ILP optimization identified the optimal numerical format much more quickly ($\leq$1s, performed on Intel i9-9960X) than the tedious and iterative process undertaken by reinforcement learning optimization algorithms (which can take several hours [185]). When constraints are selected in the region beyond the mean plus standard deviation of metrics (highlighted region), generalized posit was most frequently selected as the optimal numerical format. Note that except for a trade-off between the

accuracy and MAC frequency (Figures 5.7(b) and 5.8(b)), the numerical formats are selected in a way that to maximize accuracy when the accuracy and hardware constraints (e.g, EDP) are met. In the case of MAC frequency, the numerical formats are selected to maximize the frequency when the accuracy constraints are satisfied. In most cases, the generalized posit has shown better performance as compared to other numerical formats. When it comes to frequency performance, tapered fixed-point is selected as an appropriate numerical format. However, note that in some cases, the best possible result for a specific metric by identifying the optimal numerical format may not meet the user target. In these cases, the low-precision arithmetic approach needs to be combined with other compression optimizations such as pruning [186] and processing in memory [187] to meet user constraints.

5.7 Comparison with other posit frameworks

A summary of a comparison between the previous low-precision posit frameworks and the proposed frameworks (Cheetah-V2, and ALPS) is listed in Table 5.5. Several research groups have explored the efficacy of posit on the performance and hardware complexity of DNNs with multiple image classification tasks [46, 47, 51, 105, 107, 125]. However, none of these works analyze the appropriateness of the generalized posit and tapered fixed-point numerical format for both DNN inference and training. Additionally, previous works do not offer insight into the impact of the numerical format on both accuracy and hardware complexity, as described in

this chapter. In addition, none of the previous frameworks proposed a numerical analysis approach to explain the reasons behind the high performance accuracy of generalized posit. Finally, the Cheetah-V2 framework provides an approach to identify the appropriate numerical format based on edge devices constraints which are lacking in the previous frameworks.

Table 5.5: High-level summary of Cheetah-V2, ALPS and other low-precision posit frameworks. All datasets are image classification tasks. FMNIST: Fashion MNIST; FP: floating point; FX: fixed-point; P: posit; GP: Generalized Posit; TFX: Tapered Fixed-point; SW: software; HW: hardware.

	Cococcioni *et al.* [146]	murillo *et al.* [145]	tambe *et al.* [123]	wong *et al.* [158]	Johnson *et al.* [58]	Cheetah-V2	ALPS
Dataset	MNIST, FMNIST, ImageNet	CIFAR-10	WMT, LIBRISPEECH, ImageNet	COCO, WMT ImageNet	ImageNet	Table 5.1	CIFAR-10, ImageNet
Numerical Format	P	FP, FX, P	FP, FX P	PS	FP, FX P	FX, FP P,GP,TFX	GP, FP P
Bit-precision	[16..8]	16,8	[4..8]	8,6	8	[5..8]	[5..8]
Models	Inference	Inference/Training	Inference	Inference/Training	Inference	Inference	Inference
Implementation	SW	SW	SW & HW	SW	SW & HW	SW & HW	SW & HW
DNN library	Home Suite	Tensorflow	Pytorch	-	Pytorch	Keras/TensorFlow	Keras/TensorFlow
Device	-	-	ASIC	-	ASIC	ASIC	ASIC/FPGA
Technology Node	-	-	16 nm	-	28 nm	32/28 nm	32/28 nm

5.8 Summary

In this chapter, the efficacy of the tapered-precision numerical formats is evaluated within new low-precision, Cheetah-V2, and ALPS frameworks on 20 benchmarks for classification and prediction tasks. When it comes to performance accuracy, the generalized posit constantly outperforms other numerical formats such as posit, tapered fixed-point, float, and fixed-point. The reason behind the high performance of generalized posit is that it auto-adjusts to the dynamic range and distribution of the weights and activations, reducing quantization error which is demonstrated by the distortion rates of weights in various neural networks. In addition, the

benefit of generalized posit is also demonstrated through the numerical analysis of the quantization approach. Through this analysis, we found that the generalized posit provides a higher SQNR value as compared to posit and float. Moreover, we found that the theoretical bound on the misclassification rate for generalized posit approximately replicates the experimental result. This shows that the generalized posit configuration is selected optimally for each layer. Furthermore, the results assert that generalized posit can achieve substantial performance improvements with a relatively moderate increase in energy consumption over posit.

In the second part of this chapter, the Cheetah-V2 framework is evaluated to identify the optimal numerical format based on edge device constraints. The results show that no single numerical format can perform well for all edge devices, as these devices exhibit extremely heterogeneous in terms of hardware constraints. This is analogous to the "No free lunch theorem" in machine learning, where there is no guarantee that a model that has shown significant performance in one class of applications; can perform well in other classes. For instance, the tapered fixed-point is suitable for edge devices, providing a high frequency. Finally, the Cheetah-V2 and ALPS frameworks are compared with the existing low-precision posit-based framework for DNN. Among different frameworks, only Cheetah-V2 and ALPS frameworks support the adaptive tapered-precision numerical format such as generalized posit and tapered fixed-point. In addition, the Cheetah-V2 framework has the ability to identify the appropriate numerical format based on targeted application accuracy and edge device constraints.

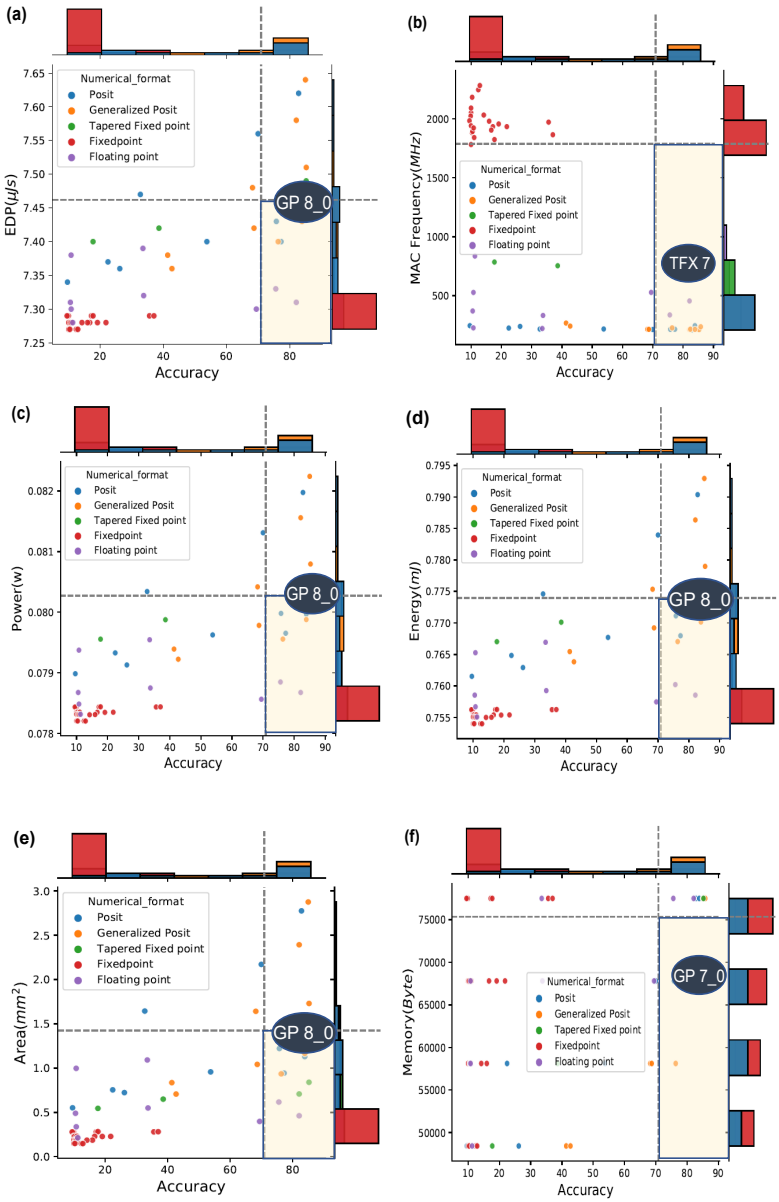

Figure 5.7: (a) EDP vs Accuracy (b) MAC Frequency vs. Accuracy (c) Power vs. Accuracy (d) Energy vs. Accuracy (e) Area vs Accuracy (f) Memory vs Accuracy for an image classification task with an accelerator configured with PEs arranged in a 16x16 systolic array and output stationary dataflow. The constraint was derived by adding the mean with the standard deviation of the metric.

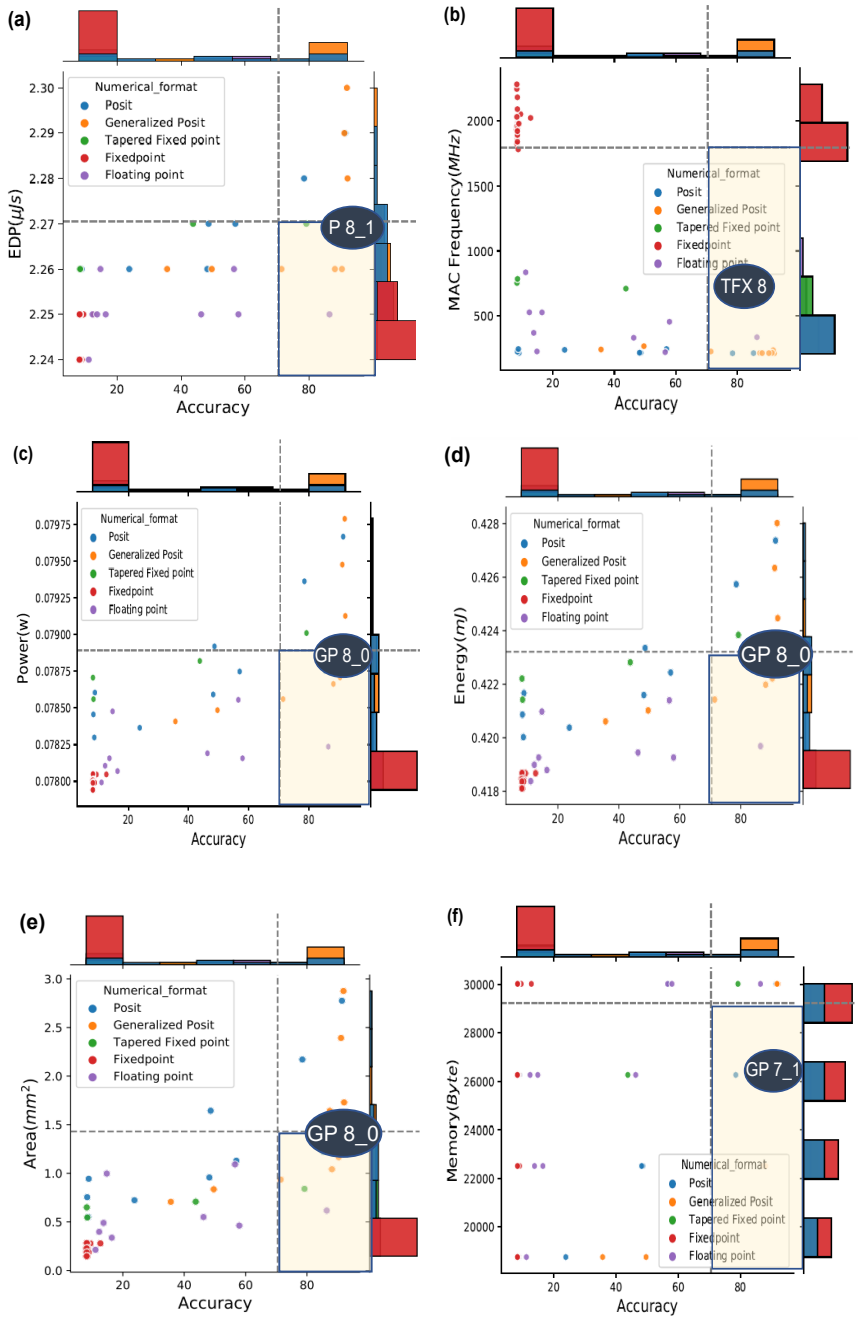

Figure 5.8: (a) EDP vs Accuracy (b) MAC Frequency vs. Accuracy (c) Power vs. Accuracy (d) Energy vs. Accuracy (e) Area vs Accuracy (f) Memory vs Accuracy for keyboard spotting task with an accelerator configured with PEs arranged in a 16x16 systolic array and output stationary dataflow. The constraint was derived by adding the mean with the standard deviation of the metric. The numerical format selected by the ILP optimizer (marked by the large dark blue oval) in the highlighted region identifies the format for which the best accuracy and metric combination is achieved. GP n_es is n-bit generalized posit with es-bit exponent.

6. Tapered-precision numerical formats for deep learning training

Over the last decade, the computational complexity and knowledge capacity requirements for training deep learning models have grown exponentially [188]. Unfortunately, the computational capabilities and memory footprints of AI accelerators have only boosted quadratically [189]. This growth-rate mismatch prevents progress on the development and deployment of new deep learning models due to the long DNN training time [18]. In addition, training current DNN models on AI accelerators requires a high bandwidth memory and consumes high energy, which consequently has a negative environmental impact [18, 21].

Similar to DNN inference, training of DNN models using the low-precision numerical format is a prominent approach to reduce the complexity of DNN training without accuracy degradation [111]. It can also reduce memory footprint by curtailing the bit-precision of DNN training parameters and can decrease latency and energy consumption associated with MAC units. For instance, training DNN models with 16-bit brain floating point (bfloat16) instead of the 32-bit floating

point reduces energy consumption by 3.4x [190]. However, attempts at reducing the bit-precision to $\leq$ 8-bit have resulted in notable accuracy degradation (e.g., 4% and 10% error for ResNet50 training on ImageNet with naive 8-bit and 4-bit training respectively [38, 48]).

To compensate the accuracy degradation, the current $\leq$ 8-bit training requires to be combined with approaches such as hybrid numerical format [191], mixed-precision training [7], stochastic rounding [51], adaptive exponent bias [192], dynamic loss scaling [38], analytical clipping [88], 2-phase rounding [48], and log quantization [45], increasing hardware complexity overhead while reducing the benefit of training with low-precision numerical format. For instance, the training throughput (training time) of MobileNet-V2 models with hybrid 8-bit floating point (HBF8: FP(1,5,2) for the backward pass and FP(1,4,3) for the forward pass) on ImageNet is only improved by 1.1x (far below 2x expectation) as compared to 16-bit IEEE standard floating point [192]. In addition, the overhead of training a ResNet50 model on ImageNet with 4-bit radix-4 numerical format is 10 Flops per gradient [48] (without considering the hyperparameter tuning overhead).

To mitigate the aforementioned challenges in performing DNN training with the low-precision numerical formats, we extend the use case of tapered precision numerical formats for deep learning training through the Jappi framework in this chapter. In addition, this study introduces the asymmetric generalized posit numerical format for $\leq$ 8-bit DNN training. It provides a high dynamic range ($\approx 2^{14}$ more than $\leq$ 8-bit floating point and fixed-point to capture the dynamic range of gradients (Figures 6.1(a) and 6.1(b)). The high dynamic range of asymmetric

Figure 6.1: Comparison DNN parameter's range and distributions with numerical formats value range and distribution. (a) 8-bit numerical formats absolute value range, fixed-point: FX(integer,fraction), floating point: FP(sign,exponent,fraction), posit:P(es), asymmetric generalized posit:AGP(es,rs_d,rs_u).(b) DNN Parameters absolute value range. The range is asymmetric toward small values.

generalized posit comes from a high-radix for regime values (e.g., radix-16 for es=2) [171]. Moreover, it is well known that the DNN parameters can be approximated with normal distribution for weights and activations [39] and log-normal for gradients [88]. The asymmetric generalized posit values can inherently match the distribution of DNN parameters due to its tapered-accuracy attribute (high accuracy near zero tapering toward the maximum representable number).

Note that the asymmetric generalized posit is a promising alternative to HFP8 since it exhibits similar accuracy to FP(1,4,3) for values in the range of [-16,16], and it could provide more than FP(1,5,2) dynamic range, as shown in Figure 6.2. Moreover, the DNN parameters have asymmetric distribution with respect to origin (Figure 6.1(b). The maximum positive value of DNN parameters is mostly bounded at 2^5 (especially for activations); however, the value of gradients heavily tailed toward zero (mostly less than 2^{-5} values). The asymmetric generalized numerical

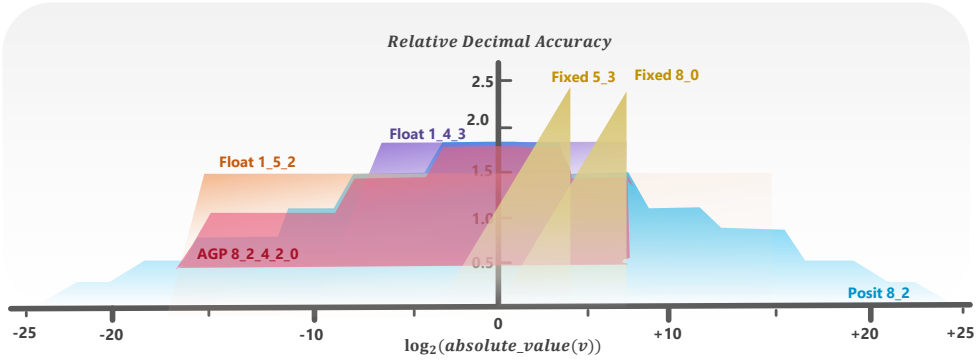

Figure 6.2: The relative decimal accuracy [6] for various 8-bit numerical formats Float 1_5_2 , Float 1_4_3, are 8-bit floating format with 5, 4 exponent bits, respectively, and Posit 8_2 are 8-bit posit format with 2 exponent bits respectively. The Fixed 5_3 and Fixed 8_0 indicates fixed-point numerical format with 5-bit integer and 8-bit integer respectively, and Generalized posit 8_2_4_2_0 is 8-bit generalize posit numerical format with $es = 2$, rs_d=4, rs_u=2, and $e_b = 0$.

format can represent asymmetric distribution by allocating different regime bit widths for values less than 1 and bigger than 1. Finally, the asymmetric generalized posit can accommodate the variability observed in DNN parameter distributions and dynamic range across layers and training epochs by optimization of three hyperparameters. These benefits motivate us to develop a low-precision deep learning framework to demonstrate the efficacy of asymmetric generalized posit in DNN training.

6.1 Problem formulation

As mentioned in Chapter 2, the aim of training a DNN model in supervised setting is learning the DNN model parameters (W) through the empirical risk minimization problem as in (6.1), where A represents the activation vector, α is the learning rate,

m indicates batch size, $Y_i(A,W)$ is output prediction, L is the loss function, and $\hat{Y}_i$ is desired output (corresponding labels).

$$\Delta W = -\alpha \nabla_W \big(\frac{1}{m}\sum_{i=1}^{m} L(Y_i(A,W),\hat{Y}_i)\big) \tag{6.1}$$

The procedure of minimizing the empirical risk, and thus learning weights, is performed by training DNN models through mini-batch stochastic gradient descent using backpropagation. The gradients of weights and activations are computed for each DNN layer in the backpropagation algorithm. The training part of the Jaapi framework is shown in Figure 6.3 for generalized posit numerical format. The main aim of the framework at each training iteration (epoch) is to identify the tapered-precision numerical format hyperparameters (rs_d, rs_u, e_b) for $C_l = \{W, A, \nabla_W, \nabla_A\}$ a set of the DNN parameter configurations. These parameters are selected either through statistical or numerical analysis approaches. Note that the energy overhead to compute (rs_d, rs_u, e_b) parameters is negligible since the statistics of parameters and gradients are estimated from the previous epoch which is inspired by the In-Hindsight quantization range estimation approach for low-precision DNN training [193].

Moreover, the selection of generalized posit parameters is briefly explained for weights, and the same equations can be used for other parameters such as activations and gradients.

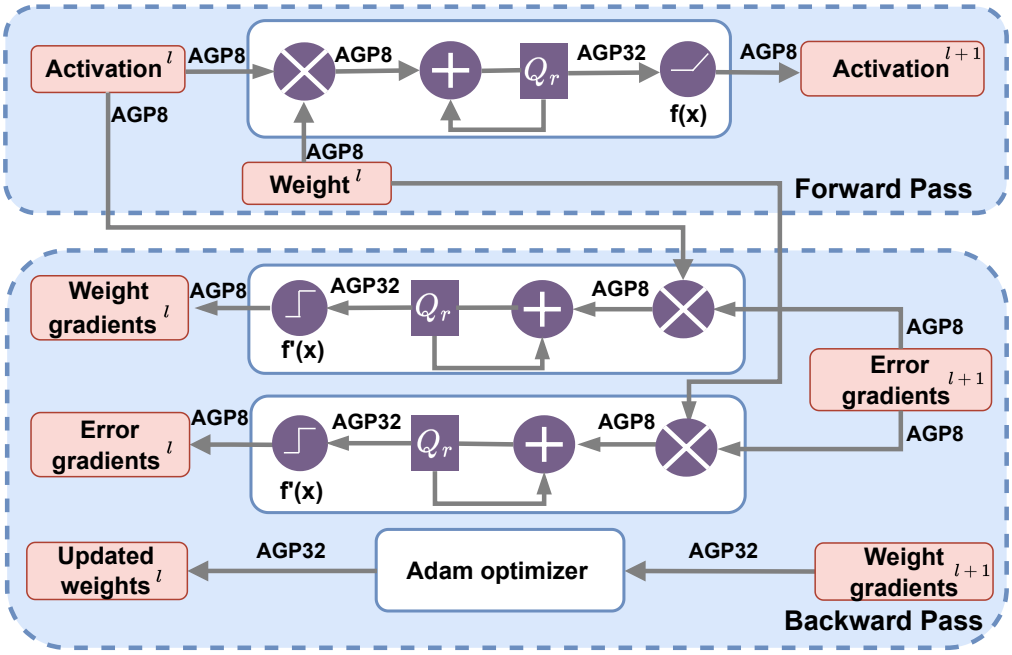

Figure 6.3: The training unit of Jaapi low-precision framework for DNN training

6.1.1 AGP parameter selection (statistical approach)

The *es* can recenter the maximum accuracy from 2^0 to 2^{es}. Therefore, it is desired to match the maximum accuracy of asymmetric generalized posit by selecting optimal e_b with the mode of weights in log_2 domain, as given by (6.2) where m and v are the mean and variance, respectively.

$$e_b(w) = \frac{m}{(\frac{v}{m^2}+1)^{3/2}} \tag{6.2}$$

To compute rs_d the difference between the excess kurtosis (EK) of DNN parameters, in the range of $[-1/e_b(w), +1/e_b(w)]$, and the excess kurtosis of asymmetric generalized posit values, with varied *rs*, is calculated. Then the asymmetric gen-

eralized posit numerical format configuration with the closest excess kurtosis to that of the DNN parameters is selected, as represented in (6.3) where the P_W and $Q_{AGP(rs_d,e_b(w))}$ are probability density function of weight distribution and asymmetric generalized posit value distribution.

$$rs_d(w) = \underset{rs_d \in [1,n-1]}{\arg\min}\ EK_d(P_W||Q_{AGP(rs_d,e_b(w))}) \tag{6.3}$$

To compute rs_u, the range of DNN parameters requires to be captured by the asymmetric generalized posit numerical format. Therefore, the difference between the maximum absolute value of DNN parameters and the maximum absolute value of asymmetric generalized posit with varied *rs* is calculated. Then, the asymmetric generalized posit numerical format configuration with the closest maximum absolute value to that of the DNN parameters is selected as represented in (6.4). Note that the asymmetric generalized posit numerical format should capture the dynamic range of DNN parameters $(max(|W|) - max(|AGP(rs_u, e_b(w))|) > 0)$.

$$rs_u(w) = \underset{rs_u \in [1,n-1]}{\arg\min}\ (max(|W|) - max(|AGP(rs_u, e_b(w))|) \tag{6.4}$$

6.1.2 AGP parameter selection (numerical analysis approach)

The output of each DNN layer (A_{l+1}) in the forwardpass is computed, as given in (6.5) through a sequence of operations. The multiply-accumulate (MAC) operation is primarily followed by an activation function F where $A_1 = x$ and $A_{n+1} = y$ are the input and output of DNN model, respectively. The errors generated in the

calculation of w_i and A_i are represented as ε_{w_i} and ε_{A_i}.

$$\begin{aligned} A_{l+1} + \varepsilon_{A_{l+1}} =& F_l(...(F_2(F_1(w_1 + \varepsilon_{w_1} \times A_1 + \varepsilon_{A_1}) + \\ & \varepsilon_{A_2}) \times (w_2 + \varepsilon_{w_2})) \end{aligned} \tag{6.5}$$

The error $\varepsilon_{A_{l+1}}$ can be estimated as in (6.6) using the chain rule of partial derivatives and first-order Taylor series approximation in neural network models with a non-linear activation function as F_i.

$$\varepsilon_{A_{l+1}} \approx \sum_{i=1}^{l} \varepsilon_{w_i} \frac{\partial F_l}{\partial w_i} + \sum_{i=1}^{l} \varepsilon_{A_i} \frac{\partial F_l}{\partial A_i} \tag{6.6}$$

In the backward pass, the activation gradients are computed as a series of operations, as given in (6.7).

$$\begin{aligned} \nabla_{A_l} + \varepsilon_{\nabla_{A_l}} =& G_l(\nabla_{A_{l+1}}, ...\nabla_{A_n}, w_l, ...w_n) = \\ & F'_{l+1} \times \big(...\times (F'_{n-1} \times \big(F'_n \times (\nabla_{A_n} + \varepsilon_{\nabla_{A_n}}) \\ & \times (w_n + \varepsilon_{w_n}) + \varepsilon_{\nabla_{A_{n-1}}}) \times (w_{n-1} \\ & + \varepsilon_{w_{n-1}})...\big) \times \big(w_l + \varepsilon_{w_l}\big)\big) \end{aligned} \tag{6.7}$$

The error $\varepsilon_{\nabla_{A_l}}$ can be estimated as in (6.8) using the same approach to compute the error $\varepsilon_{A_{l+1}}$.

$$\varepsilon_{\nabla_{A_l}} \approx \sum_{i=n}^{l+1} \varepsilon_{w_i} \frac{\partial G_l}{\partial w_i} + \sum_{i=n}^{l+1} \varepsilon_{\nabla_{A_i}} \frac{\partial G_l}{\partial \nabla_{A_i}} \tag{6.8}$$

The layerwise error in computation on weight gradients ($\varepsilon_{\nabla_{A_i}}$) and updating the weights (ε_{w_i}) depend on the error of the parameter for the same and either previous

or subsequent layers. These errors can be computed as in (6.9) and (6.10).

$$\begin{aligned} \varepsilon_{\nabla w_l} \approx & T(\varepsilon_{\nabla A_{l+1}}, \nabla_{A_{l+1}}, \varepsilon_{A_l}, A_l) = \\ & F'_{l+1} \times (\varepsilon_{\nabla A_{l+1}} \times A_l + \varepsilon_{A_l} \times \nabla_{A_{l+1}}) \end{aligned} \tag{6.9}$$

$$\begin{aligned} \varepsilon_{w_l} \approx & T(\varepsilon_{\nabla w_n^m}, \alpha, w_l) = \\ & \varepsilon_{w_l} + \alpha \sum_{i=0}^{i=m} \varepsilon_{\nabla w_n^m} \end{aligned} \tag{6.10}$$

The objective of the framework is to select optimal *rs* and *sc* for layerwise parameters to minimize the conversion errors such as ε_{w_l}, ε_{A_l}, $\varepsilon_{\nabla w_l}$, and $\varepsilon_{\nabla A_l}$ as in 6.11. Formally, we drive the conversion error from 32-bit asymmetric generalized posit numerical format with rs=15 and e_b=0 to low-precision 8-bit asymmetric generalized posit with varied *rs* and e_b.

$$\begin{aligned} rs_u, rs_d, e_b = & \arg\min_{rs_u \in [1, n-1]} \text{AGP}_\text{h}\text{-AGP}_\text{l} \\ & , \; rs_d \in [1, n-1] \\ & , \; e_b \in [-4, 3] \end{aligned} \tag{6.11}$$

6.1.3 Low-precision asymmetric generalized posit dot product

The asymmetric generalized posit dot product is presented in Algorithm 13 in Appendix A. In the first step, a set of quantized weights and activations is decoded to the asymmetric generalized posit format, and the scaling factor is computed (lines 2–5). Then, the product of the asymmetric 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* [6], for m multipliers with size $w_{quire} = \lceil \log_2(m) \rceil + 2 \times \lceil \log_2(\frac{\text{Max}_{GP}}{\text{Min}_{GP}}) \rceil + 2$ (lines 11–14). The stored products are then converted and accumulated using fixed-point arithmetic. Finally, the accumulated result is converted back to the asymmetric generalized posit numerical format (lines 15–17).

6.2 Benchmark specification

In this section, we discuss the four benchmarks to evaluate the efficacy of different tapered-precision formats in the Jaapi framework. In addition, we summarize the performance of different tapered-precision numerical formats including (posit, generalized posit, and asymmetric generalized posit) in these benchmarks. The specifications of the tasks and training performance with 32-bit floats DNNs are summarized in Table 6.1.

6.2.1 Datasets

CIFAR-10 [173]: The CIFAR-10 dataset contains images of 10 various categories such as airplanes and automobiles which are collected from the web. In this dataset, 60000 images are assigned to the training set. The training set is normalized in a range of 0 to 1. The ResNet models are trained on this dataset for 200 epochs with cross-entropy loss function and Adam optimizer. In this study, the 0.001 learning rate is gradually decreased after 80, 120, 160, and 180 epochs. To improve the

accuracy, the images are augmented.

Table 6.1: The DNN models and benchmarks using 32-bit float parameters description.

Dataset	DNN Model	# Parameters	Performance
CIFAR-10	ResNet-18	0.27 M	91.54±0.23%
	ResNet-50	0.86 M	92.10±0.24%
Tiny-ImageNet	ResNet-18	0.32 M	60.60±0.36%
	ResNet-50	0.91 M	65.00±0.41%
ImageNet	ResNet-18	11.70 M	68.10±0.52%
	ResNet-50	25.00 M	74.60±0.68%

Tiny ImageNet [174]: The Tiny ImageNet dataset contains images of 200 categories, each with 500 images. These categories are selected from the 1000 categories of ImageNet. The images in the training set are resized to 64x64 pixels with cubic spline interpolation. Afterward, the images are center cropped and augmented. The ResNet models are trained on this dataset for 24 epochs with cross-entropy loss function and Adam optimizer. In this study, the cyclic learning rate approach is used (baseline learning rate equals 0.001).

ImageNet [174]: The ImageNet dataset comprises images from 1000 different categories, created using WordNet's hierarchical system. This dataset designates 50,000 images for its test set. These test set images are first resized to 256x256 pixels using cubic spline interpolation, and then they are center cropped to a dimension of 224x224 pixels. The models are trained for 200 epochs using a single RTX 2080 Ti Nvidia GPU, with a batch size of 200 and an initial learning rate of 0.0001. The dropout rate is set at 0.2.

6.2.2 Experiment setup

The Jaapi framework is implemented in the C++ language and the CUDA platform, and extended to the TensorFlow framework [180]. To demonstrate the efficacy of the Jaapi empirical framework, the performance of asymmetric generalized posits with rs_d, rs_u, and sc is evaluated across three training tasks and compared to both generalized posits and posit with two different DNN architectures. The specifications of the tasks and performance accuracy with 32-bit float DNNs are summarized in Table 6.1. In the evaluation of each format, $es = 2$, $rs_d \in [1..n-1]$, $rs_u \in [1..n-1]$, and $sc \in [-3,3]$ are considered for the asymmetric generalized posits. The $es = 2$ is selected for posits and $es = 2$, $rs \in [1..n-1]$, and $sc \in [-3,3]$ are chosen for generalized posit.

6.3 Tapered-precision numerical formats performance

The efficacy of the Jaapi framework is evaluated for DNN training using the asymmetric generalized posits, as shown in Table 6.2.

Table 6.2: The DNN training performance using the posit, generalized posit, posit, and asymmetric generalized posit formats on various benchmarks.

Dataset	Bit Precision	Posit		Generalized Posit		Asymmetric Generalized Posit	
		ResNet-18	ResNet-50	ResNet-18	ResNet-50	ResNet-18	ResNet-50
CIFAR-10	8-bit	89.71±0.26%	90.12±0.22%	90.46±0.15%	91.02±0.20%	**91.43±0.09%**	**91.79±0.14%**
	7-bit	85.21±0.11%	83.11±0.10%	87.81±0.12%	88.92±0.26%	**89.06±0.12%**	**90.21±0.32%**
	6-bit	74.32±0.33%	50.23±0.32%	76.64±0.21%	75.43±0.34%	**79.43±0.32%**	**82.31±0.34%**
	5-bit	18.23±0.35%	11.05±0.14%	45.91±0.29%	40.31±0.15%	**57.34±0.43%**	**58.23±0.36%**
Tiny-ImageNet	8-bit	58.03±0.31%	62.54±0.18%	58.96±0.13%	63.76±0.17%	**60.43±0.22%**	**64.36±0.17%**
	7-bit	53.61±0.17%	54.11±0.09%	55.61±0.11%	59.31±0.23%	**57.38±0.19%**	**62.31±0.17%**
	6-bit	42.35±0.27%	30.26±0.31%	46.53±0.25%	48.11±0.36%	**50.36±0.31%**	**56.45±0.20%**
	5-bit	17.48±0.18%	15.41±0.20%	30.43±0.15%	32.69±0.14%	**40.61±0.43%**	**41.96±0.31%**
ImageNet	8-bit	60.21±0.65%	64.71±0.76%	61.69±0.54%	65.03±0.60%	**63.02±0.59%**	**66.53±0.55%**
	7-bit	54.33±0.70%	55.69±0.78%	56.23±0.75%	61.42±0.81%	**59.91±0.70%**	**64.70±0.63%**
	6-bit	46.64±0.78%	48.31±0.85%	50.11±0.81%	55.77±0.86%	**54.48±0.80%**	**58.01±0.70%**
	5-bit	25.61±0.76%	30.22±0.83%	35.69±0.76%	41.33±0.76%	**44.37±0.72%**	**48.71±0.77%**

On CIFAR-10, the findings indicate that an 8-bit asymmetric posit can achieve training performance comparable to a 32-bit floating format. Both the posit and generalized posit also exhibit acceptable performance with 8-bit precision in this dataset. A notable advantage of the asymmetric posit over other numerical formats is evident in the 5-bit representation, where the asymmetric generalized posit shows improvements of 39.11% and 11.43% compared to the posit and generalized posit, respectively. This leap in performance is attributed to the gradients' distribution, which follows a near-logarithmic distribution, akin to an asymmetric generalized posit distribution. The posit and generalized posit are the most appropriate numerical formats for DNN inference when the DNN follows a normal distribution. Opting for a symmetric distribution leads to a loss in the balance between encoding efficiency (how many bit strings of the numerical format are used to represent DNN parameters) and accuracy for representing DNN parameters. For example, in a 5-bit DNN inference, if rs is set to 4, 25% of the encoding becomes unusable, as the maximum value of gradients, activation, and weights is 8, while the maximum for generalized posit and posit is 4096. This excess encoding could be more accurately utilized to represent numbers less than 4 in an asymmetric generalized posit by selecting rs_u as 2. However, choosing $rs = 2$ renders numbers less than 2^{-3} unrepresentable in a generalized format, leading to significant quantization error. Conversely, with $rs = 4$, numbers up to 2^{-12} can be represented, reducing the quantization error.

When it comes to the Tiny-ImageNet and ImageNet datasets, the dynamic range of gradients increases significantly (e.g., 2^{36} for the ResNet model on ImageNet

datasets). This dynamic range is roughly covered only with 8-bit precision, and when this bit precision is reduced, the quantization error increases due to the dynamic range of DNN parameters exceeding that of the number system. For instance, there is a 30.23% loss in inference performance when the number of bits is reduced to 5-bit for the ResNet-18 model on the ImageNet dataset. In a comparison between numerical formats, similar to the trend in Fashion-MNIST, the asymmetric generalized posit shows better results due to its better adaptation to the logarithmic distribution of gradients. Overall, the results of this study assert that the asymmetric generalized posit performs best among the tested numerical formats. For example, the representation of DNN parameters with an 8-bit asymmetric generalized posit is sufficient to achieve training accuracy within a 1% variation of 32-bit floating-point. Moreover, the performance of a 6-bit low-precision asymmetric generalized posit ResNet-50 model on the Tiny ImageNet dataset is boosted by 26.19%, compared to the float-based network. These performance benefits can be intuitively explained by the auto-tuning capability of the Jaapi framework, which adapts the format to the dynamic range and distribution of the weights, activations, and gradients, thereby reducing error and improving accuracy. Finally, to achieve similar performance to the 32-bit floating-point, 10-bit, and 14-bit precision are required for the Tiny ImageNet and ImageNet datasets, respectively

As a future study, the performance of DNN training with asymmetric posit could be enhanced by allocating more bits to the scaling factor. In this study, 3 bits were allocated for the scaling factor; however, current research in low-precision arithmetic [191, 194, 195] recommends using 6 bits. Although the scaling factor

is currently the only practical solution to address and accurately represent DNN parameters, the design of a new numerical format that represents the logarithmic distribution without using a scaling factor is desirable.

6.3.1 Comparison with state-of-the-art low-precision training approaches

A summary of previous studies that proposed low-precision training using ResNet-18 and ResNet-50 models on the CIFAR-10 dataset is shown in Table 6.3. Several research groups have explored the efficacy of 8-bit floating point and fixed-point numerical format on the performance of these models on this dataset. A few works also analyze the suitability of the posit numerical format for DNN training. The results indicate that the asymmetric generalized posit is an appropriate candidate for training DNN models.

6.4 Summary

In this chapter, the proposed Jaapi framework is presented as a tool to study the performance of tapered-precision numerical formats for deep learning training. To capture the dynamic range of gradients, a new numerical format, asymmetric generalized posit, has been introduced. Interestingly, generalized posit can reproduce the values that are representable with state-of-the-art hybrid floating point (FP(1,4,3) for forward-pass and FP(1,5,2) for backward bass) without additional hardware to support both numerical formats. The process of selecting the optimal

Table 6.3: The DNN training performance using variants of IEEE-754 standard floats format, on CIFAR-10.

Numerical Format	Bit-Configuration[1]								Parameter[2]		CIFAR-10	
	W	A	∇_W	∇_A	Acc	UP	BN	Act	e_b/sc	rs	ResNet-20	ResNet-50
FX8 [130]	8	8	8	8/16	FP32	FP32	8	32	✓	-	-	-
FX8 [133]	8	8	8	8	32	32	32	32	✓	×	91.95(-0.37)%	-
FX8 [196]	8	8	8	8	FP32	FP32	FP32	32	✓	×	-	93.22(-0.13)%
MFX8 [7]	8/16	8/16	8/16	8/16	32	FP32	FP32	32	✓	×	90.86(+0.78)%	-
FX8 [132]	8	8	8	8	32	32	32	32	✓	×	92.76(+0.41)%	-
FP8 [117]	8	8	8	8	16	16	16	16	×	×	-	-
FP8 [119]	8	8	8	8	32	16	32	32	×	×	-	-
HFP8 [38]	8	8	8	8	DF16	DF16	DF16	DF16	✓	×	-	-
S2FP8 [115]	8	8	8	8	32	32	32	32	✓	×	91.10(-0.40)%	93.2(+0.20)%
LFP8 [116]	8	8	8	8	24	16	32	16	×	×	93.41(-0.10)%	-
FP8-B [194]	8	8	8	8	32	32	32	32	✓	×	-	-
FP8-B [191]	8	8	8	8	32	32	32	32	✓	×	-	-
HBFP8 [118]	8	8	8	8	32	32	32	32	10	×	-	-
HBFP8 [197]	8	8	16	8	FP32	FP32	FP32	FP32	✓	×	-	-
P8_1 [148]	8	8	8	8	32	16	16	32	✓	×	92.07(-0.14)%	-
LP8_2 [151]	8	8	8	8	20	32	32	32	✓	×	93.10(-0.30)%	-
P8_2 [158]	8	8	8	8	32	32	32	32	✓	×	91.60(0.60)%	-
Ours	8	8	8	8	32	32	32	32	✓	✓	**91.43(-0.11)**%	**91.79(-0.31)**%

[1] W: Weights; A: Activations, G: Gradients, E: Error, Acc: Accumulation, UP: Update, BN: Batch Normalization, Act: Activation

[2] e_b/sc: exponent bias or scaling factor, rs: regime bit width

asymmetric generalized posit configuration through the statistical approach and numerical analysis is explained. For instance, the excess kurtosis and maximum absolute values of DNN parameters are used as a metric to select the asymmetric generalized posit parameters. Through the Jaapi framework, the efficacy of asymmetric generalized posit on deep learning training is evaluated in comparison to posit and generalized posit. The findings show that the asymmetric generalized posit outperforms posit on most benchmarks and deep learning models.

7. Case study

7.1 Case study: Surveillance video analysis

Surveillance cameras have been used in public places to improve public safety and reduce crimes by detecting abnormal behavior, exposing hostile intent, and identifying the human subject [198]. Though previous studies showed that using surveillance cameras reduces the crime rate [199, 200], there is still a considerable gap in the performance of a surveillance camera vs. human monitors [201]. Therefore, the surveillance system is mostly utilized as a passive system. The video captured by the surveillance system is transferred to the cloud for human inspection post-event [202]. To develop more efficient, active surveillance, this study proposes the use of the Cheetah-V2 framework to automatically and instantaneously perform analyses on live video streams to detect crime-related behaviors as shown in Figure 7.1. In the process of performing training, the DNN learns new activities from videos captured by the surveillance camera and performs inference. The DNN model with learned weights classifies the abnormal activities in each video. The following subsections describe the DNN model that is applied in this

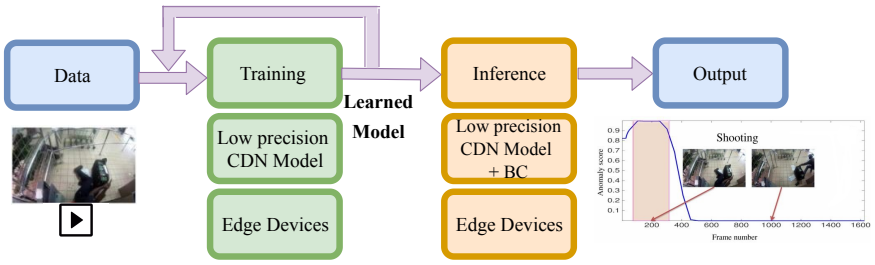

Figure 7.1: The high-level deep learning flow for abnormal activity detection on surveillance video for the edge. BC: Binary Classification

study for anomaly activity recognition in the candidate video datasets and the ideal edge-devices that are compatible with the framework.

7.1.1 DNN model & datasets

To analyze activity in a video, we propose the application of the Convolutional Drift Network (CDN), as shown in Figures. 7.2 [8]. CDNs greatly minimize training costs in a video by combining deep learning for visual feature extraction and Echo State Networks [203] for efficient temporal feature extraction. In a CDN, features extracted from a pre-trained deep Convolutional Neural Network are pushed into the ESN reservoir, where they propagate naturally, or drift through a fading memory representation provided by randomly initialized recurrent connections. They provide near state-of-the-art performance on challenging video analysis tasks and carry only minimal training complexity (one feed-forward neural network layer). In addition, CDN training does not require costly back-propagation through time algorithms common to other recurrent neural networks.

To evaluate the performance of low-precision CDN model training and in-

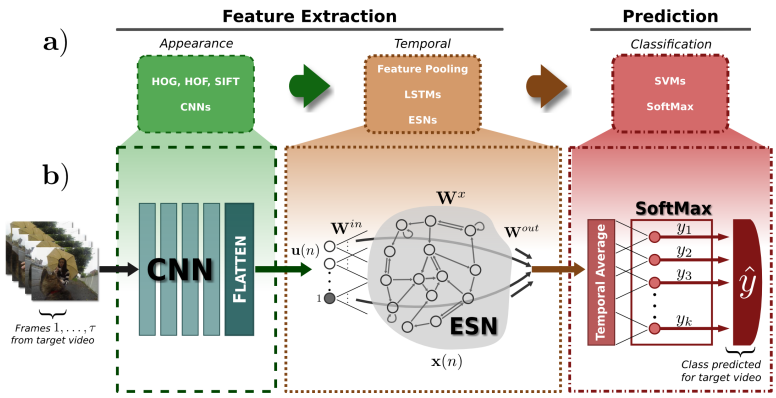

Figure 7.2: High level overview of the CDN architecture. Each frame $n = 1,\ldots,\tau$ from video v are passed through the CNN feature extractor to produce $\mathbf{u}(\mathbf{n}) = \mathbf{u}_n^{(v)}$. All ESN responses $\mathbf{U}^{(v)}$ are collected (i.e. $\mathbf{U}^{(v)} = \{\mathbf{u}_1^{(v)},\ldots,\mathbf{u}_\tau^{(v)}\} = \mathbf{x}(n)$ for $n = 1,\ldots,\tau$). Temporal averaging is performed on $\mathbf{U}^{(v)}$. Finally the averaged responses are passed into the SoftMax layer for video activity class prediction $\mathbf{y}(v)$ [8].

ference, the Avenue dataset is selected [204]. The specification of this dataset is summarized in Table 7.1. The training accuracy using the floating point in this dataset for Resnet-18 and Resnet-50 models on CDN is 65+0.43% and $68\pm0.35\%, respectively.$

Table 7.1: Specifications of the anomaly detection video dataset

Dataset	# of videos	Average # of frames	Dataset length	Example anomalies
Avenue [204]	37	839	30 minutes	Throwing objects, loitering

7.1.2 Edge device constraints

To realize which edge-device characteristics are suitable for this task, a comprehensive list of systems is summarized in Table 7.2. It is important to note that all these

systems have a built-in camera to capture the video at 30 frames per second (fps).

The aim of this study is to select the optimal numerical format to perform abnormal activity detection on surveillance video with a constraint requirement of the edge devices mentioned in Tables 7.2 and 7.3).

Table 7.2: Specifications of the SOC with edge co-processors

SOC	CPU	co-processor	SDRAM	DRAM	Throughput	Power Consumption
NVIDIA Jetson Nano	4 cores A57	128 cores Maxwell GPU	4 GB	8 GB	472 GFLOPS	6 W
Intel Neural NCS2	4 cores A53	Myrid X VPU	1 GB	8 GB	1 TFLOPS	4.5 W
Google Coral	4 cores A53	Edge TPU	1 GB	8 GB	2 TFLOPS	4.8 W
Snapdragon 855	8 cores A55 & A76	Adreno 640 GPU	6 GB	16 GB	727 GFLOPS	5 W
HiSilicon Kirin 980	8 cores A76 & A55	2 cores NPU	8 GB	16 GB	-	5 W
MediaTek Helio P70	8 cores A53 & A73	NPU	6 GB	16 GB	560 GFLOPS	5 W
Exynos 9820 Octa	8 cores M4 & A76 & A55	Adreno 640 GPU	6 GB	16 GB	727 GFLOPS	5 W

Table 7.3: Specifications of the Embedded Microcontoller chip

ARM Mbed Platform	Processor	SDRAM	DRAM	Throughput	Power Consumption
MBED LPC11U24	Cortex-M0	8 KB	32 KB	48 MCPS	0.1 W
NORDIC nRF51-DK	Cortex-M0	32 KB	256 KB	16 MCPS	0.25 W
Mbed LPC1768	Cortex-M3	32 KB	512 KB	96 MCPS	0.5 W
Nucleo F103RB	Cortex-M3	20 KB	120 KB	72 MCPS	0.5 W
Nucleo L476RG	Cortex-M4	128 KB	1 MB	80 MCPS	0.7 W
Nucleo F411RE	Cortex-M4	128 KB	512 KB	100 MCPS	0.6 W
FRDM-K64F	Cortex-M4	256 KB	1 MB	120 MCPS	0.6 W
Nucleo F746ZG	Cortex-M7	320 KB	1 MB	216 MCPS	1.5 W

7.2 Tapered precision numerical format's performance

The findings show that the asymmetric low-precision generalized posit outperforms other numerical formats in terms of accuracy. In particular, it is possible to perform abnormal detection with an 8-bit generalized posit and less than 1% accuracy

Table 7.4: The DNN inference performance using the generalized posit, posit, and asymmetric generalized posit formats on CIFAR-10.

Dataset	Bit Precision	Posit		Generalized Posit		Asymmetric Generalized Posit	
		RESNET-18	ResNet-50	RESNET-18	ResNet-50	RESNET-18	ResNet-50
Avenue	8-bit	61.32±0.21%	64.71±0.44%	63.79±0.16%	65.49+0.24%	**64.88±0.36%**	**67.81+0.32%**

degradation as compared to the performance of the same network with a 32-bit floating point. Moreover, this finding emphasizes that the asymmetric generalized posit is a good candidate for applications and models in which a trade-off between performance and hardware complexity is desired.

7.3 Summary

In this chapter, the efficacy of the proposed Cheetah-V2 framework is evaluated for real anomaly detection on videos captured by surveillance cameras. We summarized the different edge devices that can be used for this task and also described the high-level deep learning flow to perform this task. The Convolution Drift Network (CDN) is combined with Echo State Networks to minimize the training cost for edge devices. We found that among different low-precision numerical formats, the asymmetric generalized posit is the appropriate numerical format for this case study.

8. Conclusion and Future work

This research addresses the challenges of choosing an appropriate numerical format that enable the development and deployment of deep learning models on edge devices. The first challenge is to reduce the quantization error and improve the performance of DNN inference and training. This challenge is addressed by introducing low-precision deep learning frameworks (e.g., PositNN, and Cheetah) with posit numerical formats. Through these frameworks, we demonstrate that the posit numerical format has a high affinity for deep neural network inference at 5- to 8-bit precision as compared to fixed-point and floating point. In addition, the posit EMAC hardware is competitive with the floating point counterpart in terms of resource utilization and EDP. Moreover, the posit EMAC hardware offers a higher maximum operating frequency over that of floating point.

A limitation of working with posits with fewer than 8-bits is that it cannot accommodate the variability observed in inter- and intra-layer parameter distributions of DNNs. This can lead to a catastrophic drop in accuracy for six and five-bit width models. We addressed this by proposing novel adaptive quantization algorithms and low-precision frameworks (e.g, ALPS and TENT) that adjust the distribution

and dynamic range of tapered-precision numerical formats (e.g, generalized posits and tapered fixed-point) and match it with that of the DNN parameters for each layer. This adaptation of the numerical format configurations to best represent the layerwise dynamic range and distribution of parameters is performed with different approaches. For example, through the ALPS framework, we proposed a numerical analysis of the quantization error to discover the optimal generalized posit configurations. To accomplish this, we defined a novel SQNR formulation for generalized posits that uses a metric to select an appropriate generalized posit configuration. This adaptive quantization approach yields an improvement in the average classification accuracy during inference by 14% compared to the posits.

The performance of DNN inference and training is improved by representing DNN parameters in tapered-precision numerical format. However, identifying the optimal tapered-precision numerical format to achieve the best trade-offs between performance accuracy and hardware complexity introduces a large design space to explore for numerical format and bit-precision. Moreover, the hardware constraints are different across edge devices. We address this challenge using an automatic hardware-software co-design framework (Cheetah-V2). This framework identifies an appropriate numerical format based on the custom user-defined constraints through integer linear programming optimization. This framework can also be used by practitioners and startups as an Early-DSE (early stage design space exploration) framework that identifies the numerical format specification to reduce the cost of the deep learning accelerator's design.

The efficacy of the tapered-precision numerical format is demonstrated on

different benchmarks such as image classification and natural language processing. The generalized posit numerical format outperformed the floating point, fixed-point, posit, and tapered fixed-point in terms of accuracy while using fewer bits to represent weights and activations. When it comes to hardware complexity and performance trade-off, identifying the optimal numerical format depends on the specific edge device constraints that can be realized with the proposed Cheetah-V2 framework. Furthermore, the benchmark results suggest that the tapered precision numerical format is more appropriate for neural networks such as CNNs with a weight distribution range between -8 and 8. However, this is not particularly suitable for transformer models, where the weight values can potentially exceed 8.

Future work may involve further exploration of the following:

In the process of deploying a model on edge devices to perform DNN inference, in some cases even employing the optimal numerical format is not enough to meet the edge device resource constraints. Therefore, as a future study, the tapered-precision numerical formats need to be combined with other hardware/software optimizations such as pruning [205] and processing in memory [206] for deep learning inference and training.

Recent studies have shown that compressed networks through various approaches such as low-precision numerical formats are significantly vulnerable to adversarial attacks [207]. Therefore, in future work, the development of theory and algorithms that address the high compute and memory demands of deep learning models (through model compression approaches) is necessary. Ensuring the performance and robustness of the deep learning model against adversarial attacks

during deployment is also crucial to pave a path toward secure and efficient AI services at the edge.

The current generation of intelligent machines is reaching an astonishing level of performance (comparable to that of the human brain) on a wide range of challenging tasks such as image classification and natural language processing [78]. However, these intelligent machines have not demonstrated the same level of proficiency in lifelong learning. Lifelong learning involves the ability to continuously learn new tasks without forgetting previous ones, reusing the learned knowledge, exploiting similarities between multiple tasks, tolerating noise, and performing tasks in a resource-efficient way [208]. Designing an intelligent machine with lifelong learning capabilities remains a major challenge, and addressing this challenge requires a holistic approach with cross-layer optimization. Despite numerous optimizations for lifelong learning [209, 210], only a few studies focus on the low-precision numerical format for lifelong learning [211, 212]. Therefore, the tapered-precision numerical format could be an appropriate numerical format for lifelong learning.

In the future, the existence of an AI accelerator with posit arithmetic support will enable us to study the full-scale implications of deployment-phase deep learning. This would allow us to go beyond synthesizing only the EMAC operation on hardware and combine the results with emulation frameworks, such as SCALE-Sim, to measure hardware metrics for DNN models [183].

Finally, adding a new low-precision numerical format to deep learning libraries is a tedious and non-trivial task [213]. This thesis offers different frameworks to

utilize tapered-precision numerical formats for deep learning. However, a range of opportunities are available to improve the portability and user-friendly nature of these frameworks.

A. Appendix

A.1 Algorithms

Algorithm 1 posit decoder for converting an n-bit input with es exponent bits into a real value.
Input: posit (n, es)
Output: real value

1: **procedure** DECODE(in) ▷ Data extraction of input
2: $\texttt{nzero} \leftarrow |\texttt{in}$ ▷ '1' if in is nonzero
3: $\texttt{sign} \leftarrow \texttt{in}[n\text{–}1]$ ▷ Extract sign
4: $\texttt{twos} \leftarrow (\{n\text{–}1\{\texttt{sign}\}\} \oplus \texttt{in}[n\text{–}2:0]) + \texttt{sign}$
5: $\texttt{rc} \leftarrow \texttt{twos}[n\text{–}2]$ ▷ Regime check
6: $\texttt{inv} \leftarrow \{n\text{–}1\{\texttt{rc}\}\} \oplus \texttt{twos}$ ▷ Invert 2's

Extract regime value
7: $\texttt{zc} \leftarrow \text{LZD}(\texttt{inv})$
8: $\texttt{reg} \leftarrow \texttt{rc}\ ?\ \texttt{zc}\text{–}1 : \text{–}\texttt{zc}$

Extract exponent and fraction
9: $\texttt{tmp} \leftarrow \texttt{twos}[n\text{–}4:0] \ll (\texttt{zc} - 1)$
10: $\texttt{exp} \leftarrow \texttt{tmp}[n\text{–}2 : n\text{–}es\text{–}1]$
11: $\texttt{frac} \leftarrow \{\texttt{nzero}, \texttt{tmp}[n\text{–}es\text{–}2:0]\}$
12: **return** $(-1)^{\texttt{sign}} \times 2^{2^{\texttt{es}} \times \texttt{reg}} \times 2^{\texttt{exp}} \times \left(1 + \frac{\texttt{frac}}{2^{\texttt{fs}}}\right)$
13: **end procedure**

Algorithm 2 posit encoder for converting a real value to *n*-bit outputs with *es* exponent bits.
Input: real value
Output: posit (n,es)

```
1:  procedure ENCODE(in)                          ▷ Data extraction of input
2:      nzero ← |in                               ▷ '1' if in is nonzero
3:      sign ← in > 0 ? 0 : 1
4:      Abs ← Abs(in)
        Encode the regime bit
5:      if in > 1 then
6:          RunLength ← 1
7:          while in > 2^(2^es) & RunLength > n − 1 do
8:              in ← in/2^(2^es)
9:              RunLength ← RunLength + 1
10:         end while
11:         reg ← RunLength − 1                   ▷ set Regime bit
12:     else
13:         RunLength ← 0
14:         while in < 1 & RunLength > n − 1 do
15:             in ← in×2^(2^es)
16:             RunLength ← RunLength + 1
17:         end while
18:         reg ← −1 × RunLength                  ▷ set Regime bit
19:     end if
        Encode the exponent bit
20:     e ← 2^(es−1)
21:     while e > 0.5 do
22:         if in < 2^e then
23:             in ← in/2^e
24:             exp ← exp + 1
25:         end if
26:         e ← e/2
27:     end while
        Encode the fraction
28:     frac ← RNE(in − 1)                        ▷ Round-Tie-Even
29:     return sign, reg, exp, frac
30: end procedure
```

Algorithm 3 Generalized posit decoder for converting an n-bit input with es exponent bits, rs with $\lceil \log_2(n-1) \rceil$ bit-width, and e_b with 3 bit-width, into a real value.

Input: Generalized posit (n, es, rs, e_b)

Output: real value

procedure DECODE(in) ▷ Data extraction of `input`
  $\texttt{nzero} \leftarrow |\texttt{in}$ ▷ '1' if `in` is nonzero
  $\texttt{sign} \leftarrow \texttt{in}[n-1]$ ▷ Extract sign
  $\texttt{twos} \leftarrow (\{n-1\{\texttt{sign}\}\} \oplus \texttt{in}[n-2:0]) + \texttt{sign}$
  $\texttt{rc} \leftarrow \texttt{twos}[n-2]$ ▷ Regime check
  $\texttt{inv} \leftarrow \{n-1\{\texttt{rc}\}\} \oplus \texttt{twos}$ ▷ Invert 2's

  Extract regime value with rs
  $\texttt{zc} \leftarrow \text{LZD}(\texttt{inv}[\texttt{n}-1:\texttt{n}-1-\texttt{rs}])$
  $\texttt{reg} \leftarrow \texttt{rc}\ ?\ \texttt{zc}-1 : -\texttt{zc}$

  Extract exponent and fraction
  $\texttt{tmp} \leftarrow \texttt{twos}[n-2:0] \ll (\texttt{zc}+1)$
  $\texttt{exp} \leftarrow \texttt{tmp}[n-2:n-es-1]$
  $\texttt{frac} \leftarrow \{\texttt{nzero}, \texttt{tmp}[n-es-2:0]\}$
  return $(-1)^{\texttt{sign}} \times 2^{(2^{\texttt{es}} \times \texttt{reg}) + \texttt{exp} + \texttt{e}_\texttt{b}} \times \left(1 + \frac{\texttt{frac}}{2^{\texttt{fs}}}\right)$
end procedure

Algorithm 4 Generalized posit encoder for converting a real value to n-bit outputs with es exponent bits, rs with $\lceil \log_2(n-1) \rceil$ bit-width, and e_b with 3 bit-width.
Input: real value
Output: Generalized posit (n, es, rs, e_b)

procedure ENCODE(in) ▷ Data extraction of input
    nzero $\leftarrow$ |in ▷ '1' if in is nonzero
    sign $\leftarrow$ in $>$ 0 ? 0 : 1
    Abs $\leftarrow$ Abs(in)

Encode the regime bit
    if $in > 1$ **then**
        RunLength $\leftarrow$ 1
        while $in > 2^{2^{es}}$ & $RunLength > rs$ **do**
            in$\leftarrow$ in$/2^{2^{\text{es}}}$
            RunLength $\leftarrow$ RunLength $+ 1$
        end while
        reg $\leftarrow$ RunLength $- 1$ ▷ set Regime bit
    else
        RunLength $\leftarrow$ 0
        while $in < 1$ & $RunLength > rs$ **do**
            in$\leftarrow$ in$\times 2^{2^{\text{es}}}$
            RunLength $\leftarrow$ RunLength $+ 1$
        end while
        reg $\leftarrow -1 \times$ RunLength ▷ set Regime bit
    end if

Encode the exponent bit
    e $\leftarrow 2^{\text{es}-1}$
    while $e > 0.5$ **do**
        if $in < 2^e$ **then**
            in $\leftarrow$ in$/2^{\text{e}}$
            exp $\leftarrow$ exp $+ 1$
        end if
        e $\leftarrow$ e$/2$
    end while

Encode the fraction
    frac $\leftarrow$ RNE(in $- 1$) ▷ Round-Tie-Even
    return sign, reg, exp, frac
end procedure

Algorithm 5 Tapered fixed-point decoder for converting an n-bit input with IS with $\lceil \log_2(n) \rceil$ bit-width, and SC with 3 bit-width into a real value.
Input: Tapered fixed point (n, IS, SC)
Output: real value

1: **procedure** DECODE(in) ▷ Data extraction of input
2: $\texttt{nzero} \leftarrow |\texttt{in}$ ▷ '1' if in is nonzero
3: $\texttt{sign} \leftarrow \texttt{in}[n-1]$ ▷ Extract sign
4: $\texttt{twos} \leftarrow (\{n-1\{\texttt{sign}\}\} \oplus \texttt{in}[n-2:0]) + \texttt{sign}$
5: $\texttt{inv} \leftarrow \{n-1\{\overline{\texttt{sign}}\}\} \oplus \texttt{twos}$ ▷ Invert 2's

Extract Integer value with IS
6: $\texttt{zc} \leftarrow \text{LZD}(\texttt{inv}[\texttt{n}-1:\texttt{n}-1-IS]) + 1$
7: $\texttt{Int} \leftarrow \overline{\texttt{sign}}\ ?\ \texttt{zc}-1 : -\texttt{zc}$

Extract fraction
8: $\texttt{tmp} \leftarrow \texttt{twos}[n-2:0] \ll (\texttt{zc}+1)$
9: $\texttt{frac} \leftarrow \{\texttt{nzero}, \texttt{tmp}[n-2:0]\}$
10: **return** $(\texttt{Int} + (\frac{\texttt{frac}}{2^{\texttt{fs}}})) \times 2^{SC}$
11: **end procedure**

Algorithm 6 Tapered fixed point encoder for converting a real value to n-bit outputs with IS with $\lceil \log_2(n) \rceil$ bit-width, and SC with 3 bit-width into a real value.
Input: real value
Output: Tapered fixed point (n,IS,SC)

procedure ENCODE(in) ▷ Data extraction of `input`
    $\texttt{nzero} \leftarrow |\texttt{in}$ ▷ '1' if `in` is nonzero
    $\texttt{sign} \leftarrow \texttt{in} > 0 \ ? \ 0 : 1$

Encode the integer bit
    $\texttt{RunLength} \leftarrow 1$
    while $0 > in > 1$ & $RunLength > IS$ **do**
        $\texttt{in} \leftarrow \texttt{in} + (-1)^{\texttt{sign}}$
        $\texttt{RunLength} \leftarrow \texttt{RunLength} + 1$
    end while
    $\texttt{Int} \leftarrow (-1)^{\texttt{sign}} \times \texttt{RunLength}$ ▷ set Regime bit

Encode the fraction
    $\texttt{frac} \leftarrow \texttt{RNE}(\texttt{in})$ ▷ Round-Tie-Even
    return `sign`, `Int`, `frac`
end procedure

Algorithm 7 Compute the maximum regime bit-width (*rs*) and scaling factor (*sc*) of generalized posit for DNN parameters
Input: layers weights (W_l), layers activations (A_l)
Output: rs_{w_l},rs_{A_l},sc_{w_l},sc_{A_l} generalized posit parameters

procedure RS-SC SELECTION (W_l,A_l)
$\mathtt{w_{log}} \leftarrow \mathtt{RNE}(\log_2(|\mathtt{w_l}|),2)$
$\mathtt{A_{log}} \leftarrow \mathtt{RNE}(\log_2(|\mathtt{A_l}|),2)$
$\mathtt{w_{max}} \leftarrow \max(\mathtt{w_{log}})$
$\mathtt{A_{max}} \leftarrow \max(\mathtt{A_{log}})$
$\mathtt{w_{min}} \leftarrow \min(\mathtt{w_{log}})$
$\mathtt{A_{min}} \leftarrow \min(\mathtt{A_{log}})$

Compute the sc_{w_l}, sc_{A_l}
$\mathtt{sc_A} \leftarrow |\mathtt{Mean}(\mathtt{W_{log}})|$
$\mathtt{sc_W} \leftarrow |\mathtt{Mean}(\mathtt{A_{log}})|$

Compute the rs_{w_l}, rs_{A_l}
for $k \leftarrow n-1$ to 1 do
$\mathtt{rs_{max}}, \mathtt{rs'_{max}}, \mathtt{rs_{min}}, \mathtt{rs'_{min}} \leftarrow \mathtt{n}-1$
$\mathtt{GP_{log}} \leftarrow \mathtt{RNE}(\log_2(|\mathtt{GP}(\mathtt{n},\mathtt{k},\mathtt{sc_w})|),2)$
$\mathtt{GP'_{log}} \leftarrow \mathtt{RNE}(\log_2(|\mathtt{GP}(\mathtt{n},\mathtt{k},\mathtt{sc_a})|),2)$
$\mathtt{GP_{max}} \leftarrow \max(\mathtt{GP_{log}})$
$\mathtt{GP'_{max}} \leftarrow \max(\mathtt{GP'_{log}})$
$\mathtt{GP_{min}} \leftarrow \min(\mathtt{GP_{log}})$
$\mathtt{GP'_{min}} \leftarrow \min(\mathtt{GP'_{log}})$
if $\mathtt{GP_{max}} > \mathtt{w_{max}}$ then
$\mathtt{rs_{max}} \leftarrow \mathtt{k}$
end if
if $\mathtt{GP'_{max}} > \mathtt{A_{max}}$ then
$\mathtt{rs'_{max}} \leftarrow \mathtt{k}$
end if
if $\mathtt{GP_{min}} < \mathtt{w_{min}}$ then
$\mathtt{rs_{min}} \leftarrow \mathtt{k}$
end if
if $\mathtt{GP'_{min}} < \mathtt{A_{min}}$ then
$\mathtt{rs'_{min}} \leftarrow \mathtt{k}$
end if
end for
$\mathtt{rs_w} \leftarrow \max(\mathtt{rs_{max}}, \mathtt{rs_{min}})$
$\mathtt{rs_A} \leftarrow \max(\mathtt{rs'_{max}}, \mathtt{rs'_{min}})$
end procedure

Algorithm 8 Compute the maximum integer bit width (*IS*) and scaling factor (*SC*) of tapered fixed-point for DNN parameters
Input: layers weights (W_l), layers activations (A_l)
Output: IS_{w_l},IS_{A_l},SC_{w_l} tapered fixed-point parameters

procedure *IS*, *SC* SELECTION (W_l,A_l)
    $\mathtt{w_{amax}} \leftarrow \max(|\mathtt{w_l}|)$
    $\mathtt{A_{amax}} \leftarrow \max(|\mathtt{A_l}|)$

    Compute the IS_{w_l},IS_{A_l}
    if $\lfloor \mathtt{W_{amax}} \rfloor + 1 \leq \mathtt{n_{bit}}$ then
        $\mathtt{IS_w} \leftarrow \lfloor \mathtt{W_{amax}} \rfloor + 1$
    else
        $\mathtt{IS_w} \leftarrow \mathtt{n_{bit}}$
    end if
    if $\lfloor \mathtt{A_{amax}} \rfloor + 1 \leq \mathtt{n_{bit}}$ then
        $\mathtt{IS_A} \leftarrow \lfloor \mathtt{A_{amax}} \rfloor + 1$
    else
        $\mathtt{IS_A} \leftarrow \mathtt{n_{bit}}$
    end if

    Compute the SC_{w_l}
    $\mathtt{SC_{w_l}} \leftarrow 0$
    if $\mathtt{W_{amax}} < 0.5$ then
        $\mathtt{SC_W} \leftarrow \lfloor \log_2(\mathtt{W_{amax}}) \rfloor + 1$
    end if
    return $\mathtt{IS_{w_l}}, \mathtt{IS_{A_l}}, \mathtt{SC_{w_l}}$
end procedure

Algorithm 9 Posit EMAC operations for vector elements each with n bits, es exponent bits.
Input: layers quantized weights (W_{l_q}), layers quantized activations (A_{l_q}),
Output: R as a dot product result

procedure GENERALIZED POSIT EMAC (W_{l_q}, A_{l_q})
  $\mathtt{sign_w}, \mathtt{reg_w}, \mathtt{exp_w}, \mathtt{frac_w} \leftarrow$ DECODE(W_{l_q})
  $\mathtt{sign_a}, \mathtt{reg_a}, \mathtt{exp_a}, \mathtt{frac_a} \leftarrow$ DECODE(A_{l_q})

Gather total scale factors
  $\mathtt{sf_w} \leftarrow 2^{\mathtt{es}\times\mathtt{reg_w}} + \mathtt{exp_w}$
  $\mathtt{sf_a} \leftarrow 2^{\mathtt{es}\times\mathtt{reg_a}} + \mathtt{exp_a}$

Multiplication
  $\mathtt{sign_{mult}} \leftarrow \mathtt{sign_w} \oplus \mathtt{sign_a}$
  $\mathtt{frac_{mult}} \leftarrow \mathtt{frac_w} \times \mathtt{frac_a}$
  $\mathtt{normfrac_{mult}} \leftarrow \mathtt{frac_{mult}} \gg \mathtt{frac_{mult}}[\mathtt{MSB}]$
  $\mathtt{sf_{mult}} \leftarrow \mathtt{sf_w} + \mathtt{sf_a} + \mathtt{frac_{mult}}[\mathtt{MSB}]$
  $\mathtt{p_{mult}} \leftarrow (-1)^{\mathtt{sign}} \times 2^{\mathtt{sf_{mult}}} \times (1 + \mathtt{frac_{mult}})$

Accumulation
  $\mathtt{fracs_{mult}} \leftarrow \mathtt{sign_{mult}}\ ?\ -\mathtt{frac_{mult}} : \mathtt{frac_{mult}}$
  $\mathtt{sf_{biased}} \leftarrow \mathtt{sf_{mult}} + 2^{es+1} \times (n-2)$
  $\mathtt{fracs_{fixed}} \leftarrow \mathtt{fracs_{mult}} \ll \mathtt{sf_{biased}}$
  $\mathtt{sum_{quire}} \leftarrow \mathtt{fracs_{fixed}} + \mathtt{sum_{quire}}$

Rounding & Encode
  $\mathtt{R} \leftarrow$ ROUNDING & ENCODING($\mathtt{sum_{quire}}$)
  return R
end procedure

Algorithm 10 Generalized posit EMAC 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

procedure GENERALIZED POSIT DP (W_{l_q}, A_{l_q})
  $\mathtt{sign_w}, \mathtt{reg_w}, \mathtt{exp_w}, \mathtt{frac_w} \leftarrow$ DECODE(W_{l_q}, rs_w)
  $\mathtt{sign_a}, \mathtt{reg_a}, \mathtt{exp_a}, \mathtt{frac_a} \leftarrow$ DECODE(A_{l_q}, rs_a)

  Gather total scale factors
  $\mathtt{sf_w} \leftarrow 2^{\mathtt{es} \times \mathtt{reg_w}} + \mathtt{exp_w} + \mathtt{sc_w}$
  $\mathtt{sf_a} \leftarrow 2^{\mathtt{es} \times \mathtt{reg_a}} + \mathtt{exp_a} + \mathtt{sc_a}$

  Multiplication
  $\mathtt{sign_{mult}} \leftarrow \mathtt{sign_w} \oplus \mathtt{sign_a}$
  $\mathtt{frac_{mult}} \leftarrow \mathtt{frac_w} \times \mathtt{frac_a}$
  $\mathtt{normfrac_{mult}} \leftarrow \mathtt{frac_{mult}} \gg \mathtt{frac_{mult}}[\mathtt{MSB}]$
  $\mathtt{sf_{mult}} \leftarrow \mathtt{sf_w} + \mathtt{sf_a} + \mathtt{frac_{mult}}[\mathtt{MSB}]$
  $\mathtt{p_{mult}} \leftarrow (-1)^{\mathtt{sign}} \times 2^{\mathtt{sf_{mult}}} \times (1 + \mathtt{frac_{mult}})$

  Accumulation
  $\mathtt{fracs_{mult}} \leftarrow \mathtt{sign_{mult}}\ ?\ -\mathtt{frac_{mult}} : \mathtt{frac_{mult}}$
  $\mathtt{sf_{biased}} \leftarrow \mathtt{sf_{mult}} + 2^{es+1} \times (n-2)$
  $\mathtt{fracs_{fixed}} \leftarrow \mathtt{fracs_{mult}} \ll \mathtt{sf_{biased}}$
  $\mathtt{sum_{quire}} \leftarrow \mathtt{fracs_{fixed}} + \mathtt{sum_{quire}}$

  Rounding & Encode
  $\mathtt{R} \leftarrow$ ROUNDING & ENCODING($\mathtt{sum_{quire}}$)
  return R
end procedure

Algorithm 11 Tapered fixed-point EMAC operations for n-bit inputs each with $\lceil \log n \rceil$ bits for *RS*, 3 bits for *SC*.
Input: layers quantized weights (W_{l_q}), layers quantized activations (A_{l_q})
Output: R as Dot Product result

procedure TAPERED FIXED-POINT DP (W_{l_q}, A_{l_q})
  $\texttt{sign}_\texttt{w}, \texttt{Int}_\texttt{w}, \texttt{frac}_\texttt{w} \leftarrow$ DECODE(W_{l_q}, IS_w)
  $\texttt{sign}_\texttt{a}, \texttt{Int}_\texttt{a}, \texttt{frac}_\texttt{a} \leftarrow$ DECODE(A_{l_q}, IS_a)

Multiplication
  $\texttt{sign}_\texttt{mult} \leftarrow \texttt{sign}_\texttt{w} \oplus \texttt{sign}_\texttt{a}$
  $\texttt{Value}_\texttt{w} \leftarrow (\texttt{Int}_\texttt{w} + \texttt{frac}_\texttt{w}) \ll \texttt{frac}_{\texttt{bit}_\texttt{w}} + |\texttt{SC}_\texttt{w}|$
  $\texttt{Value}_\texttt{a} \leftarrow (\texttt{Int}_\texttt{a} + \texttt{frac}_\texttt{a}) \ll \texttt{frac}_{\texttt{bit}_\texttt{a}}$
  $\texttt{p}_\texttt{mult} \leftarrow \{\texttt{sign}, \texttt{Value}_\texttt{w} \times \texttt{Value}_\texttt{a}\}$

Accumulation & Normalize
  $\texttt{sum}_\texttt{quire} \leftarrow \texttt{p}_\texttt{mul} + \texttt{sum}_\texttt{quire}$ ▷ Accumulate
  $\texttt{sum}_\texttt{nquire} \leftarrow \texttt{sum}_\texttt{quire} \gg \texttt{frac}_\texttt{a} + \texttt{frac}_\texttt{w} + |\texttt{SC}_\texttt{w}|$

Rounding & Encode
  $\texttt{result} \leftarrow$ ROUNDING & ENCODING($\texttt{sum}_\texttt{nquire}$)
  return result
end procedure

Algorithm 12 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

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

    Compute the rs_{w_r}
    $\mathtt{K}_{\mathtt{GP}_0} \leftarrow \mathtt{Kurt}(\mathtt{GP}(\mathtt{n}, \mathtt{es}, \mathtt{rs}_{\mathtt{r}_0}, \mathtt{sc}_{\mathtt{w}_0}), -\mathtt{W}_{\mathtt{amax}}, \mathtt{W}_{\mathtt{amax}})$
    $\mathtt{K\_diff}^{\mathtt{w}}_{0} \leftarrow |\mathtt{K}_{\mathtt{w}} - \mathtt{K}_{\mathtt{GP}_0}|$
    for $i \leftarrow 3$ to $n-2$ **do**
        $\mathtt{K}_{\mathtt{GP}_i} \leftarrow \mathtt{Kurt}(\mathtt{GP}(\mathtt{n}, \mathtt{es}, \mathtt{i}, \mathtt{sc}_{\mathtt{w}_r}), -\mathtt{W}_{\mathtt{amax}}, \mathtt{W}_{\mathtt{amax}})$
        $\mathtt{K\_diff}^{\mathtt{w}}_{\mathtt{i}} \leftarrow |\mathtt{K}_{\mathtt{w}} - \mathtt{K}_{\mathtt{GP}_i}|$
        if $\mathtt{K\_diff}^{\mathtt{w}}_{\mathtt{i}} < \mathtt{K\_diff}^{\mathtt{w}}_{0}$ **then**
            $\mathtt{rs}_{\mathtt{w}_r} \leftarrow \mathtt{i}$
            $\mathtt{K\_diff}^{\mathtt{w}}_{0} \leftarrow \mathtt{K\_diff}^{\mathtt{w}}_{\mathtt{i}}$
        end if
    end for

    Compute the sc_{w_r}
    $\mathtt{M}_{\mathtt{GP}_0} \leftarrow \mathtt{mean}(\mathtt{GP}(\mathtt{n}, \mathtt{es}, \mathtt{rs}_{\mathtt{r}_w}, \mathtt{sc}_{\mathtt{r}_0}))$
    $\mathtt{M\_diff}^{\mathtt{W}}_{0} \leftarrow |\mathtt{M}_{\mathtt{W}} - \mathtt{M}_{\mathtt{GP}_0}|$
    for $i \leftarrow 1$ to 3 **do**
        $\mathtt{M}_{\mathtt{GP}_i} \leftarrow \mathtt{mean}(\mathtt{GP}(\mathtt{n}, \mathtt{es}, \mathtt{rs}_{\mathtt{r}_0}, \mathtt{i}))$
        $\mathtt{M\_diff}^{\mathtt{W}}_{\mathtt{i}} \leftarrow |\mathtt{M}_{\mathtt{W}} - \mathtt{M}_{\mathtt{GP}_i}|$
        if $\mathtt{M\_diff}^{\mathtt{A}}_{\mathtt{i}} < \mathtt{M\_diff}^{\mathtt{W}}_{0}$ **then**
            $\mathtt{sc}_{\mathtt{W}_r} \leftarrow -\mathtt{i}$
            $\mathtt{M\_diff}^{\mathtt{W}}_{0} \leftarrow \mathtt{M\_diff}^{\mathtt{W}}_{\mathtt{i}}$
        end if
    end for
end procedure

Algorithm 13 asymmetric generalized posit dot product operations for vector elements each with n bits, es exponent bits, rs_d with $\lfloor \log_2(n-1) \rfloor$ bit-width, rs_u 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

procedure GENERALIZED POSIT DP (W_{l_q}, A_{l_q})
    $\mathtt{sign_w}, \mathtt{reg_w}, \mathtt{exp_w}, \mathtt{frac_w} \leftarrow$ DECODE$(W_{l_q}, rs_{u_w}, rs_{d_w})$
    $\mathtt{sign_a}, \mathtt{reg_a}, \mathtt{exp_a}, \mathtt{frac_a} \leftarrow$ DECODE$(A_{l_q}, rs_{u_a}, rs_{d_w})$

Gather total scale factors
    $\mathtt{sf_w} \leftarrow 2^{\mathtt{es} \times \mathtt{reg_w}} + \mathtt{exp_w} + \mathtt{sc_w}$
    $\mathtt{sf_a} \leftarrow 2^{\mathtt{es} \times \mathtt{reg_a}} + \mathtt{exp_a} + \mathtt{sc_a}$

Multiplication
    $\mathtt{sign_{mult}} \leftarrow \mathtt{sign_w} \oplus \mathtt{sign_a}$
    $\mathtt{frac_{mult}} \leftarrow \mathtt{frac_w} \times \mathtt{frac_a}$
    $\mathtt{normfrac_{mult}} \leftarrow \mathtt{frac_{mult}} \gg \mathtt{frac_{mult}}[\mathtt{MSB}]$
    $\mathtt{sf_{mult}} \leftarrow \mathtt{sf_w} + \mathtt{sf_a} + \mathtt{frac_{mult}}[\mathtt{MSB}]$
    $\mathtt{p_{mult}} \leftarrow (-1)^{\mathtt{sign}} \times 2^{\mathtt{sf_{mult}}} \times (1 + \mathtt{frac_{mult}})$

Accumulation
    $\mathtt{fracs_{mult}} \leftarrow \mathtt{sign_{mult}} \; ? \; -\mathtt{frac_{mult}} : \mathtt{frac_{mult}}$
    $\mathtt{sf_{biased}} \leftarrow \mathtt{sf_{mult}} + 2^{es+1} \times (n-2)$
    $\mathtt{fracs_{fixed}} \leftarrow \mathtt{fracs_{mult}} \ll \mathtt{sf_{biased}}$
    $\mathtt{sum_{quire}} \leftarrow \mathtt{fracs_{fixed}} + \mathtt{sum_{quire}}$

Rounding & Encode
    $\mathtt{R} \leftarrow$ ROUNDING & ENCODING$(\mathtt{sum_{quire}})$
    return R
end procedure

A.2 Derivation of Equation 4.11

The quantization error of the generalized posit numerical format $\varepsilon_p(x_i, x_i') = x_i - x_i'$ 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 (A.1) where $\varepsilon_{fx}(x_i, x_i')$ presents the fixed-point quantizer error, y' is the input of the expander function, x' is the output of the expander function computed in (A.2), Δ is quantization step size and γ is a real number. Note that to derive the Equation 4.11, we follow [168].

$$\varepsilon_p(x_i, x_i') = \varepsilon_{fx}(x_i, x_i') \frac{dx'}{dy'} \tag{A.1}$$

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

By obtaining the derivative ($\frac{dx'}{dy'}$) using (A.2) and replace result in (A.1), we have (A.3)

$$\varepsilon_p(x_i, x_i') = \varepsilon_{fx}(x_i, x_i') \frac{\gamma}{2\theta} \left(e^{(\gamma y')} + e^{-(\gamma y')}\right) \tag{A.3}$$

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 $\varepsilon_p(x_i, x_i')$ is correlated to x' as (A.4)

$$\begin{aligned}\varepsilon_p(x_i,x_i') &= \varepsilon_{fx}(x_i,x_i')\frac{\gamma}{2\theta}\left(e^{(\gamma y')}+e^{-(\gamma y')}\right)\\ &\approx \varepsilon_{fx}(x_i,x_i')\gamma\left|x'\right|\end{aligned} \tag{A.4}$$

The generalized posit quantization error can be expressed in terms of x rather than x' as in (A.5) where the generalized posit quantization error $\varepsilon_p(x_i,x_i')$ is much smaller than the inputs (and underflow and overflow does not occur).

$$\begin{aligned}\varepsilon_p(x_i,x_i') &\approx \varepsilon_{fx}(x_i,x_i')\gamma|x_i+\varepsilon_p(x_i,x_i')|\\ &\approx \varepsilon_{fx}(x_i,x_i')\gamma|x_i|\end{aligned} \tag{A.5}$$

From (A.5), the generalized posit SQNR can be expressed as (A.6) and (A.7). $\Delta_{GP}^2 = u^2$ and $\varepsilon_{fx}^2(x_i,x_i') = \frac{u^2}{12}$ where u is the smallest value that can be represented by the generalized posit numerical format.

$$\text{SQNR}_{GP} = \frac{\mathbb{E}\{x^2\}}{\mathbb{E}\{\varepsilon_p^2(x_i,x_i')\}} \approx \frac{12}{\gamma^2}\times(\Delta_{GP}^{-1})^2 \tag{A.6}$$

$$\text{SQNR}_{GP}(dB) \approx (10.79-20\log(\gamma))+20\log(\Delta_{GP}^{-1}) \tag{A.7}$$

$$\Delta_{GP} = \begin{cases} 2^{-(2^{es}rs-2^{es-(n-rs-1)})+sc}, & \text{if } (n-rs\leq es+1)\\ 2^{-(2^{es}rs-es+(n-rs-1))+sc}, & \text{otherwise}\end{cases} \tag{A.8}$$

Table A.1: The DNN inference performance using the tapered-precision numerical formats on CIFAR-10 dataset (P: posit, FP: floating point, FX: fixed-point, GP: generalized posit, TFX: Tapered fixed-point).

Format	CIFAR-10 (ResNet-8)				CIFAR-10 (ResNet-18)			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	85.31±0.19%	77.28±0.22%	53.78±0.28%	26.19±0.32%	91.03±0.18%	90.04±0.17%	78.00±0.27%	19.91±0.36%
FP	82.10±0.24%	69.45±0.19%	11.28±0.23%	10.79±0.35%	91.05±0.21%	88.66±0.25%	58.31±0.20%	10.02±0.32%
FX	37.00±0.33%	21.80±0.30%	15.97±0.28%	12.90±0.35%	54.20±0.31%	24.05±0.29%	12.96±0.35%	11.11±0.40%
GP	**85.81±0.21%**	**83.90±0.17%**	**76.36±0.25%**	**42.72±0.30%**	**91.31±0.17%**	**90.64±0.20%**	**86.52±0.23%**	**52.23±0.30%**
TFX	85.24±0.22%	82.10±0.26%	38.60±0.27%	17.72±0.33%	81.66±0.24%	75.90±0.23%	46.79±0.26%	23.44±0.31%
32-bit FP	86.26±0.30%				91.54±0.23%			

Format	CIFAR-10 (ResNet-50)				CIFAR-10 (EfficientNet-B0)			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	91.15±0.25%	88.66±0.27%	58.31±0.31%	10.02±0.36%	96.89±0.30%	91.27±0.22%	60.09±0.34%	10.00±0.40%
FP	89.70±0.30%	74.73±0.28%	27.63±0.37%	10.00±0.45%	94.61±0.32%	78.22±0.26%	25.96±0.39%	10.05±0.46%
FX	12.03±0.32%	10.44±0.31%	10.28±0.35%	10.01±0.42%	17.65±0.29%	14.36±0.27%	10.54±0.36%	10.03±0.45%
GP	**91.75**±0.22%	**90.62±0.20%**	**76.00±0.25%**	**51.65±0.29%**	**97.37±0.28%**	**92.91±0.25%**	**70.64±0.33%**	**53.66±0.40%**
TFX	80.52±0.28%	74.32±0.29%	45.33±0.32%	22.56±0.41%	87.45±0.30%	79.23±0.28%	50.61±0.35%	28.83±0.42%
32-bit FP	92.10±0.24%				98.00±0.36%			

Table A.2: The DNN inference performance using the tapered-precision numerical formats on Speech commands v2, Visual Wake Word, and ImageNet datasets (P: posit, FP: floating point, FX: fixed-point, GP: generalized posit, TFX: Tapered fixed-point).

Format	Keyword Spotting				Visual Wake Word			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	91.97±0.35%	85.33±0.40%	48.62±0.46%	23.79±0.49%	83.02±0.38%	80.58±0.36%	74.53±0.45%	66.28±0.51%
FP	86.45±0.40%	46.29±0.42%	13.72±0.48%	8.50±0.53%	80.02±0.41%	68.25±0.40%	59.95±0.49%	59.37±0.52%
FX	12.70±0.38%	8.87±0.43%	8.39±0.50%	8.22±0.58%	76.43±0.37%	72.06±0.41%	61.86±0.52%	60.71±0.60%
GP	**92.10±0.32%**	**91.39±0.39%**	**88.14±0.45%**	**49.62±0.48%**	**83.02±0.35%**	**82.14±0.44%**	**76.72±0.45%**	**69.97±0.49%**
TFX	79.20±0.37%	43.75±0.35%	8.52±0.44%	8.43±0.52%	82.97±0.34%	81.92±0.40%	76.00±0.48%	66.76±0.55%
32-bit FP	92.15±0.41%				82.72±0.47%			

Format	ImageNet (ResNet-18)				ImageNet (ResNet-50)			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	66.23±0.55%	61.95±0.58%	46.41±0.59%	3.66±0.63%	73.61±0.65%	69.10±0.70%	53.46±0.73%	0.10±0.75%
FP	66.64±0.61%	50.85±0.64%	25.31±0.60%	4.63±0.65%	70.71±0.66%	55.34±0.63%	24.95±0.70%	0.11±0.78%
FX	9.46±0.65%	7.31±0.67%	4.23±0.73%	0.10±0.79%	10.48±0.72%	8.17±0.68%	5.49±0.67%	0.10±0.85%
GP	**67.72±0.54%**	**66.55±0.57%**	**52.37±0.63%**	**40.93±0.62%**	**74.11±0.64%**	**72.46±0.68%**	**63.76±0.70%**	**46.22±0.72%**
TFX	56.11±**0.29%**	51.72±0.59%	32.84±0.64%	15.31±0.71%	63.86±0.75%	56.88±0.70%	34.93±0.64%	11.51±0.69%
32-bit FP	68.10±0.52%				74.60±0.68%			

Bibliography

[1] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," in *Advances in Neural Information Processing Systems*, 2018, pp. 1963–1972.

[2] C. Bailas, M. Marsden, D. Zhang, N. E. O'Connor, and S. Little, "Performance of video processing at the edge for crowd-monitoring applications," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. IEEE, 2018, pp. 482–487.

[3] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang *et al.*, "Streaming end-to-end speech recognition for mobile devices," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6381–6385.

[4] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," *CoRR*, vol. abs/1812.03443, 2018. [Online]. Available: http://arxiv.org/abs/1812.03443

[5] S. Kodali, P. Hansen, N. Mulholland, P. Whatmough, D. Brooks, and G.-Y. Wei, "Applications of deep neural networks for ultra low power iot," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 589–592.

[6] 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.

[7] A. Rajagopal, D. Vink, S. Venieris, and C.-S. Bouganis, "Multi-precision policy enforced training (muppet): A precision-switching strategy for quantised fixed-point training of cnns," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7943–7952.

[8] D. Graham, S. H. F. Langroudi, C. Kanan, and D. Kudithipudi, "Convolutional drift networks for video classification," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2017, pp. 1–8.

[9] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[10] M. Haque, S. Marwaha, C. K. Deb, S. Nigam, A. Arora, K. S. Hooda, P. L. Soujanya, S. K. Aggarwal, B. Lall, M. Kumar *et al.*, "Deep learning-based approach for identification of diseases of maize crop," *Scientific reports*, vol. 12, no. 1, pp. 1–14, 2022.

[11] D. Tuia, B. Kellenberger, S. Beery, B. R. Costelloe, S. Zuffi, B. Risse, A. Mathis, M. W. Mathis, F. van Langevelde, T. Burghardt *et al.*, "Perspectives in machine learning for wildlife conservation," *Nature communications*, vol. 13, no. 1, pp. 1–15, 2022.

[12] M. Taddeo, T. McCutcheon, and L. Floridi, "Trusting artificial intelligence in cybersecurity is a double-edged sword," *Nature Machine Intelligence*, vol. 1, no. 12, pp. 557–560, 2019.

[13] A. S. Chakravarthy, R. Roy, and P. Ravirathinam, "Mrscatt: A spatio-channel attention-guided network for mars rover image classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1961–1970.

[14] R. Evans *et al.*, "De novo structure prediction with deep-learning based scoring," in *In Thirteenth Critical Assessment of Techniques for Protein Structure Prediction*, 2018.

[15] I. P. Waldmann and C. A. Griffith, "Mapping saturn using deep learning," *Nature Astronomy*, p. 1, 2019.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[17] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[18] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[19] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute trends across three eras of machine learning," *arXiv preprint arXiv:2202.05924*, 2022.

[20] C. Li, "Openai's gpt-3 language model: A technical overview," *https://lambdalabs.com/blog/demystifying-gpt-3/*, 2020.

[21] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," *arXiv preprint arXiv:2104.10350*, 2021.

[22] "2021 electricity rates by state," *https://standards.ieee.org/project/3109.html*.

[23] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[25] A. Wongpanich, H. Pham, J. Demmel, M. Tan, Q. Le, Y. You, and S. Kumar, "Training efficientnets at supercomputer scale: 83% imagenet top-1 accuracy in one hour," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 947–950.

[26] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training. arxiv 2021," *arXiv preprint arXiv:2104.00298*.

[27] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[28] "Global edge ai hardware market (2022-2027) by component, device, power consumption, function, end user, geography, competitive analysis, and the impact of covid-19 with ansoff analysis," *Research and Market*, p. 1, 2022.

[29] A. Kaul, "Mapping saturn using deep learning," *NVIDIA's Inference Push for AI*, 2018.

[30] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[31] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[32] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 517–532, 2021.

[33] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Memory-efficient patch-based inference for tiny deep learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2346–2358, 2021.

[34] J. H. Shin, A. Shafiee, A. Pedram, H. Abdel-Aziz, L. Li, and J. Hassoun, "Griffin: Rethinking sparse optimization for deep learning architectures," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 861–875.

[35] B. Zhou, Y. Sun, D. Bau, and A. Torralba, "Revisiting the importance of individual units in cnns via ablation," *arXiv preprint arXiv:1806.02891*, 2018.

[36] W. Park, D. Jin, and C.-S. Kim, "Eigencontours: Novel contour descriptors based on low-rank approximation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 2667–2675.

[37] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "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.

[38] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, X. Cui, W. Zhang, K. Gopalakrishnan *et al.*, "Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks," 2019.

[39] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 7950–7958.

[40] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," *arXiv preprint arXiv:2004.09602*, 2020.

[41] B. Darvish Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner *et al.*, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 271–10 281, 2020.

[42] V. Popescu, A. Venigalla, D. Wu, and R. Schreiber, "Representation range needs for 16-bit neural network training," *arXiv preprint arXiv:2103.15940*, 2021.

[43] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. Mahoney *et al.*, "Hawq-v3: Dyadic neural network quantization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 875–11 886.

[44] S. Q. Zhang, B. McDanel, and H. Kung, "Fast: Dnn training under variable precision block floating point with stochastic rounding," *Computer*, 2022.

[45] B. Chmiel, R. Banner, E. Hoffer, H. B. Yaacov, and D. Soudry, "Logarithmic unbiased quantization: Practical 4-bit training in deep learning," *arXiv preprint arXiv:2112.10769*, 2022.

[46] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1474–1479.

[47] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[48] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, "Ultra-low precision 4-bit training of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[49] P3109, "Standard for arithmetic formats for machine learning," *https://standards.ieee.org/project/3109.html*, 2021.

[50] S. H. Langroudi, T. Pandit, and D. Kudithipudi, "Deep learning inference on embedded devices: Fixed-point vs posit," *arXiv preprint arXiv:1805.08624*, 2018.

[51] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Advances in neural information processing systems*, 2018, pp. 7686–7695.

[52] S. Hashemi, N. Anthony, H. Tann, R. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," *arXiv preprint arXiv:1612.03940*, 2016.

[53] Z. Li and C. De Sa, "Dimension-free bounds for low-precision training," 2018.

[54] Y. Ding, J. Liu, J. Xiong, and Y. Shi, "On the universal approximability and complexity bounds of quantized relu neural networks," *arXiv preprint arXiv:1802.03646*, 2018.

[55] S. H. F. 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)*, March 2018, pp. 19–23.

[56] 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.

[57] ——, "Performance-efficiency trade-off of low-precision numerical formats in deep neural networks," *arXiv preprint arXiv:1903.10584*, 2019.

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

[59] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv preprint arXiv:1808.04752*, 2018.

[60] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," *arXiv preprint arXiv:1612.01543*, 2016.

[61] H. F. Langroudi, Z. Carmichael, J. L. Gustafson, and D. Kudithipudi, "Positnn framework: Tapered precision deep learning inference for the edge," in *2019 IEEE Space Computing Conference (SCC)*. IEEE, 2019, pp. 53–59.

[62] H. F. Langroudi, Z. Carmichael, D. Pastuch, and D. Kudithipudi, "Cheetah: Mixed low-precision hardware & software co-design framework for dnns on the edge," *arXiv preprint arXiv:1908.02386*, 2019.

[63] H. F. Langroudi, V. Karia, J. L. Gustafson, and D. 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*, 2020, pp. 726–727.

[64] H. F. Langroudi, V. Karia, T. Pandit, and D. Kudithipudi, "Tent: Efficient quantization of neural networks on the tiny edge with tapered fixed point," *arXiv preprint arXiv:2104.02233*, 2021.

[65] H. F. Langroudi, V. Karia, Z. Carmichael, A. Zyarah, T. Pandit, J. L. Gustafson, and D. Kudithipudi, "Alps: Adaptive quantization of deep neural networks with generalized posits," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3100–3109.

[66] H. F. Langroudi, "ACTION: Automated hardware-software Codesign framework for low-precision numerical format selecTION in tinyml," in *CoNGA 2022: Conference on Next Generation Arithmetic (CoNGA)*, March 1, 2022.

[67] H. F. Langroudi, Z. Carmichael, and D. Kudithipudi, "Deep learning training on the edge with low-precision posits," *arXiv preprint arXiv:1907.13216*, 2019.

[68] 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.

[69] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[70] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[71] A. Ivakhnenko, "Cybernetic predicting devices," Tech. Rep.

[72] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron," *IEICE Technical Report, A*, vol. 62, no. 10, pp. 658–665, 1979.

[73] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[74] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[75] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[77] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on fpga," *arXiv preprint arXiv:1511.05552*, 2015.

[78] Y. Bengio, Y. Lecun, and G. Hinton, "Deep learning for ai," *Communications of the ACM*, vol. 64, no. 7, pp. 58–65, 2021.

[79] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *arXiv preprint arXiv:2106.04554*, 2021.

[80] A. Iwata, Y. Yoshida, S. Matsuda, Y. Sato, and N. Suzumura, "An artificial neural network accelerator using general purpose 24 bits floating point digital signal processors," in *IJCNN*, vol. 2, 1989, pp. 171–182.

[81] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[82] R. Barnes, E. Cooke-Yarborough, and D. Thomas, "An electronic digital computor using cold cathode counting tubes for storage," *Electronic Engineering*, 1951.

[83] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[84] S. Migacz, "8-bit inference with tensorrt," in *GPU Technology Conference*, 2017.

[85] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein, "Training quantized nets: A deeper understanding," in *Advances in Neural Information Processing Systems*, 2017, pp. 5811–5821.

[86] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[87] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[88] B. Chmiel, L. Ben-Uri, M. Shkolnik, E. Hoffer, R. Banner, and D. Soudry, "Neural gradients are near-lognormal: improved quantized and sparse training," *arXiv preprint arXiv:2006.08173*, 2020.

[89] R. Morris, "Tapered floating point: A new floating-point representation," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1578–1579, 1971.

[90] H. Hamada, "Urr: Universal representation of real numbers," *New Generation Computing*, vol. 1, no. 2, pp. 205–209, 1983.

[91] ——, "A new real number representation and its operation," in *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH)*. IEEE, 1987, pp. 153–157.

[92] A. M. Azmi and F. Lombardi, "On a tapered floating point system," in *Proceedings of 9th Symposium on Computer Arithmetic*. IEEE, 1989, pp. 2–9.

[93] H. Yokoo, "Overflow/underflow-free floating-point number representations with self-delimiting variable-length exponent field," *IEEE transactions on computers*, vol. 41, no. 8, pp. 1033–1039, 1992.

[94] N. Akinari and H. Hagiwara, "On the real-number representation with variable-length exponent field," *Information processing letters*, vol. 52, no. 1, pp. 1–6, 1994.

[95] J. L. Gustafson, "A generalized framework for matching arithmetic format to application requirements," in *https://posithub.org/*, 2020.

[96] ——, "A generalized framework for matching arithmetic format to application requirements," in *https://posithub.org/*, 2020.

[97] J. Gustafson, *The End of Error: Unum Computing*, ser. Chapman & Hall/CRC Computational Science. Taylor & Francis, 2015. [Online]. Available: https://books.google.com/books?id=W2ThoAEACAAJ

[98] U. W. Kulisch, "Computer arithmetic in theory and practice," Tech. Rep., 1981.

[99] J. L. Gustafson *et al.*, "Standard for posit™ arithmetic," in *https://posithub.org/*, 2021, pp. 1–12.

[100] J. Gustafson, "Posit arithmetic," 2017.

[101] D. Hammerstrom, "A vlsi architecture for high-performance, low-cost, on-chip learning," in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*. IEEE, 1990, pp. 537–544.

[102] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural computation*, vol. 7, no. 1, pp. 108–116, 1995.

[103] N. Morgan, *Experimental determination of precision requirements for back-propagation training of artificial neural networks*, 1991.

[104] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *CoRR*, vol. abs/1502.02551, 2015. [Online]. Available: http://arxiv.org/abs/1502.02551

[105] M. Courbariaux, Y. Bengio, and J. David, "Low precision arithmetic for deep learning," *CoRR*, vol. abs/1412.7024, 2014. [Online]. Available: http://arxiv.org/abs/1412.7024

[106] *IEEE standard for binary floating-point arithmetic*. New York: Institute of Electrical and Electronics Engineers, 1985, note: Standard 754-1985.

[107] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[108] N. P. Jouppi, D. H. Yoon, G. Kurian, S. Li, N. Patil, J. Laudon, C. Young, and D. Patterson, "A domain-specific supercomputer for training deep neural networks," *Communications of the ACM*, vol. 63, no. 7, pp. 67–78, 2020.

[109] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen *et al.*, "A study of bfloat16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

[110] W. Kahan, "Further remarks on reducing truncation errors, commun," *Assoc. Comput. Mach*, vol. 8, p. 40, 1965.

[111] P. Zamirai, J. Zhang, C. R. Aberger, and C. De Sa, "Revisiting bfloat16 training," *arXiv preprint arXiv:2010.06192*, 2020.

[112] A. Agrawal, S. M. Mueller, B. M. Fleischer, X. Sun, N. Wang, J. Choi, and K. Gopalakrishnan, "Dlfloat: A 16-b floating point format designed for deep learning training and inference," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 92–95.

[113] "Ai-float™ - mixed precision arithmetic for ai: A hardware perspective," *https://docs.graphcore.ai/*, 2022.

[114] A. Nannarelli, "Variable precision 16-bit floating-point vector unit for embedded processors," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2020, pp. 96–102.

[115] L. Cambier, A. Bhiwandiwalla, T. Gong, M. Nekuii, O. H. Elibol, and H. Tang, "Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks," *arXiv preprint arXiv:2001.05674*, 2020.

[116] J. Zhao, S. Dai, R. Venkatesan, M.-Y. Liu, B. Khailany, B. Dally, and A. Anandkumar, "Low-precision training in logarithmic number system using multiplicative weight update," *arXiv preprint arXiv:2106.13914*, 2021.

[117] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," *Advances in neural information processing systems*, vol. 31, 2018.

[118] M. Drumond, T. Lin, M. Jaggi, and B. Falsafi, "Training dnns with hybrid block floating point," *arXiv preprint arXiv:1804.01526*, 2018.

[119] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul, "Mixed precision training with 8-bit floating point," *arXiv preprint arXiv:1905.12334*, 2019.

[120] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[121] J. Choi, P. I.-J. Chuang, Z. Wang, S. Venkataramani, V. Srinivasan, and K. Gopalakrishnan, "Bridging the accuracy gap for 2-bit quantized neural networks (qnn)," *arXiv preprint arXiv:1807.06964*, 2018.

[122] Z. Deng, C. Xu, Q. Cai, and P. Faraboschi, "Reduced-precision memory value approximation for deep learning," *Hewlett Packard Labs, HPL-2015-100*, 2015.

[123] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. J. Reddi, A. Rush, D. Brooks, and G.-Y. Wei, "Adaptivfloat: A floating-point based data type for resilient deep learning inference," *arXiv preprint arXiv:1909.13271*, 2019.

[124] J. Wilkinson, "Rounding errors in algebraic processes," 1965.

[125] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 1742–1752.

[126] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas *et al.*, "Mixed precision training of convolutional neural networks using integer operations," *arXiv preprint arXiv:1802.00930*, 2018.

[127] J. Park, S. Lee, and D. Jeon, "A neural network training processor with 8-bit shared exponent bias floating point and multiple-way fused multiply-add trees," *IEEE Journal of Solid-State Circuits*, 2021.

[128] H. Fan, G. Wang, M. Ferianc, X. Niu, and W. Luk, "Static block floating-point quantization for convolutional neural networks on fpga," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 28–35.

[129] Z. Song, Z. Liu, and D. Wang, "Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[130] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *arXiv preprint arXiv:1805.11046*, 2018.

[131] D. Yin, A. Pananjady, M. Lam, D. Papailiopoulos, K. Ramchandran, and P. Bartlett, "Gradient diversity: a key ingredient for scalable distributed learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 1998–2007.

[132] K. Zhao, S. Huang, P. Pan, Y. Li, Y. Zhang, Z. Gu, and Y. Xu, "Distribution adaptive int8 quantization for training cnns," in *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021.

[133] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1969–1979.

[134] S. Dai, R. Venkatesan, M. Ren, B. Zimmer, W. Dally, and B. Khailany, "Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference," *Proceedings of Machine Learning and Systems*, vol. 3, 2021.

[135] C. Sakr, Y. Kim, and N. Shanbhag, "Analytical guarantees on numerical precision of deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3007–3016.

[136] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302.

[137] Z. Dong, Z. Yao, Y. Cai, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq-v2: Hessian aware trace-weighted quantization of neural networks," *arXiv preprint arXiv:1911.03852*, 2019.

[138] M. Kerner, K. Tammemäe, J. Raik, and T. Hollstein, "Triple fixed-point mac unit for deep learning," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1404–1407.

[139] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, "Post-training 4-bit quantization of convolution networks for rapid-deployment," *arXiv preprint arXiv:1810.05723*, 2018.

[140] C. Te Ewe, P. Y. Cheung, and G. A. Constantinides, "Dual fixed-point: An efficient alternative to floating-point computation," in *International Conference on Field Programmable Logic and Applications*. Springer, 2004, pp. 200–208.

[141] M. Cococcioni, E. Ruffaldi, and S. Saponara, "Exploiting posit arithmetic for deep neural networks in autonomous driving applications," in *2018 International Conference of Electrical and Electronic Technologies for Automotive*. IEEE, 2018, pp. 1–6.

[142] E. de Haan, A. Podobas, and S. Matsuoka, "Towards accelerating deep neural network training on fpgas: Facilitating the use of variable precision," 2018.

[143] Z. Wan, E. Mibuari, E.-Y. Yang, and T. Tambe, "Study of posit numeric in speech recognition neural inference," *Harvard Univ., Cambridge, MA, USA, Tech. Rep. CS247r*, 2018.

[144] A. Guntoro, C. De La Parra, F. Merchant, F. De Dinechin, J. L. Gustafson, M. Langhammer, R. Leupers, and S. Nambiar, "Next generation arithmetic for edge computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1357–1365.

[145] R. Murillo, A. A. Del Barrio, and G. Botella, "Deep pensieve: A deep learning framework based on the posit number system," *Digital Signal Processing*, vol. 102, p. 102762, 2020.

[146] M. Cococcioni, F. Rossi, E. Ruffaldi, S. Saponara, and B. D. de Dinechin, "Novel arithmetics in deep neural networks signal processing for autonomous driving: Challenges and opportunities," *IEEE Signal Processing Magazine*, vol. 38, no. 1, pp. 97–110, 2020.

[147] 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.

[148] J. Lu, C. Fang, M. Xu, J. Lin, and Z. Wang, "Evaluations on deep neural networks training using posit number system," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 174–187, 2020.

[149] G. Raposo, P. Tomás, and N. Roma, "Positnn: Training deep neural networks with mixed low-precision posit," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 7908–7912.

[150] N.-M. Ho, D.-T. Nguyen, H. De Silva, J. L. Gustafson, W.-F. Wong, and I. J. Chang, "Posit arithmetic for the training and deployment of generative adversarial networks," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1350–1355.

[151] Y. Wang, D. Deng, L. Liu, S. Wei, and S. Yin, "Lpe: Logarithm posit processing element for energy-efficient edge-device training," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, pp. 1–4.

[152] S. Nambi, S. Ullah, S. S. Sahoo, A. Lohana, F. Merchant, and A. Kumar, "Expan (n) d: Exploring posits for efficient artificial neural network design in fpga-based systems," *IEEE Access*, vol. 9, pp. 103 691–103 708, 2021.

[153] V. Gohil, S. Walia, J. Mekie, and M. Awasthi, "fixed-posit: A floating-point representation for error-resilient applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 10, pp. 3341–3345, 2021.

[154] N. Shah, L. I. G. Olascoaga, S. Zhao, W. Meert, and M. Verhelst, "9.4 piu: A 248gops/w stream-based processor for irregular probabilistic inference networks using precision-scalable posit arithmetic in 28nm," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 150–152.

[155] A. Y. Romanov, A. L. Stempkovsky, I. V. Lariushkin, G. E. Novoselov, R. A. Solovyev, V. A. Starykh, I. I. Romanova, D. V. Telpukhov, and I. A. Mkrtchan, "Analysis of posit and bfloat arithmetic of real numbers for machine learning," *IEEE Access*, vol. 9, pp. 82 318–82 324, 2021.

[156] M. Zolfagharinejad, M. Kamal, A. Afzali-Khusha, and M. Pedram, "Posit process element for using in energy-efficient dnn accelerators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2022.

[157] Y. Nakahara, Y. Masuda, M. Kiyama, M. Amagasaki, and M. Iida, "A posit based multiply-accumulate unit with small quire size for deep neural networks," *IPSJ Transactions on System LSI Design Methodology*, vol. 15, pp. 16–19, 2022.

[158] W.-F. Wong, "Qtorch+: Next generation arithmetic for pytorch machine learning," in *Next Generation Arithmetic: Third International Conference,*

CoNGA 2022, Singapore, March 1–3, 2022, Revised Selected Papers, vol. 13253. Springer Nature, 2022, p. 31.

[159] J. L. Holi and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 281–290, 1993.

[160] C. Lauter and A. Volkova, "A framework for semi-automatic precision and accuracy analysis for fast and rigorous deep learning," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2020, pp. 103–110.

[161] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[162] C. Sakr and N. 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)*. IEEE, 2018, pp. 1090–1094.

[163] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[164] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[165] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 58–68.

[166] N. S. Eliezer, R. Banner, E. Hoffer, H. Ben-Yaakov, and T. Michaeli, "Energy awareness in low precision neural networks," *arXiv preprint arXiv:2202.02783*, 2022.

[167] Y.-H. Chen, T.-J. Yang, J. Emer, and V. 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, vol. 9, no. 2, pp. 292–308, 2019.

[168] B. Widrow and I. Kollár, "Quantization noise," *Cambridge University Press*, vol. 2, p. 5, 2008.

[169] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE transactions on information theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

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

[171] P. Lindstrom, "Variable-radix coding of the reals," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2020, pp. 111–116.

[172] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[173] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[174] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[175] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[176] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau *et al.*, "Mlperf tiny benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[177] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," *arXiv preprint arXiv:1906.05721*, 2019.

[178] Y. Iwashita, A. Takamine, R. Kurazume, and M. S. Ryoo, "First-person animal activity recognition from egocentric videos," in *2014 22nd International Conference on Pattern Recognition*. IEEE, 2014, pp. 4310–4315.

[179] M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Using Large Corpora*, vol. 273, 1994.

[180] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[181] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[182] D. Graham, S. H. F. Langroudi, C. Kanan, and D. Kudithipudi, "Convolutional drift networks for video classification," in *proceedings of the IEEE International Conference on Rebooting Computing (ICRC)*, 2017, pp. 1–8.

[183] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[184] D. A. Patterson, "Latency lags bandwith," *Communications of the ACM*, vol. 47, no. 10, pp. 71–75, 2004.

[185] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[186] B. Chmiel, I. Hubara, R. Banner, and D. Soudry, "Optimal fine-grained n: M sparsity for activations and neural gradients," *arXiv preprint arXiv:2203.10991*, 2022.

[187] H. Kim, J. Mu, C. Yu, T. T.-H. Kim, and B. Kim, "A 1-16b reconfigurable 80kb 7t sram-based digital near-memory computing macro for processing neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[188] T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray *et al.*, "Scaling laws for autoregressive generative modeling," *arXiv preprint arXiv:2010.14701*, 2020.

[189] A. Gholami, Z. Yao, S. Kim, M. W. Mahoney, and K. Keutzer, "Ai and memory wall," *RiseLab Medium Post*, vol. 1, p. 6, 2021.

[190] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.

[191] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, and H. Wu, "Fp8 formats for deep learning," 2022. [Online]. Available: https://arxiv.org/abs/2209.05433

[192] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, H. Tran *et al.*, "Rapid: Ai accelerator for ultra-low precision training and inference," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 153–166.

[193] M. Fournarakis and M. Nagel, "In-hindsight quantization range estimation for quantized training," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3063–3070.

[194] B. Noune, P. Jones, D. Justus, D. Masters, and C. Luschi, "8-bit numerical formats for deep neural networks," *arXiv preprint arXiv:2206.02915*, 2022.

[195] S. P. Perez, Y. Zhang, J. Briggs, C. Blake, J. Levy-Kramer, P. Balanca, C. Luschi, S. Barlow, and A. W. Fitzgibbon, "Training and inference of large language models using 8-bit floating point," *arXiv preprint arXiv:2309.17224*, 2023.

[196] J. Chen, Y. Gai, Z. Yao, M. W. Mahoney, and J. E. Gonzalez, "A statistical framework for low-bitwidth training of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 883–894, 2020.

[197] S. Fox, S. Rasoulinezhad, J. Faraone, P. Leong *et al.*, "A block minifloat representation for training deep neural networks," in *International Conference on Learning Representations*, 2020.

[198] T. Ko, "A survey on behavior analysis in video surveillance for homeland security applications," in *2008 37th IEEE Applied Imagery Pattern Recognition Workshop*. IEEE, 2008, pp. 1–8.

[199] G. Alexandrie, "Surveillance cameras and crime: a review of randomized and natural experiments," *Journal of Scandinavian Studies in Criminology and Crime Prevention*, vol. 18, no. 2, pp. 210–222, 2017.

[200] E. L. Piza, "The crime prevention effect of cctv in public places: a propensity score analysis," *Journal of Crime and Justice*, vol. 41, no. 1, pp. 14–30, 2018.

[201] W. Sultani, C. Chen, and M. Shah, "Real-world anomaly detection in surveillance videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6479–6488.

[202] L. Valentín, S. A. Serrano, R. O. García, A. Andrade, M. A. Palacios-Alonso, and L. E. Sucar, "A cloud-based architecture for smart video surveillance." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 42, 2017.

[203] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[204] C. Lu, J. Shi, and J. Jia, "Abnormal event detection at 150 fps in matlab," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 2720–2727.

[205] S. Li, W. Romaszkan, A. Graening, and P. Gupta, "Swis–shared weight bit sparsity for efficient neural network acceleration," *arXiv preprint arXiv:2103.01308*, 2021.

[206] C. Zhou, F. G. Redondo, J. Büchel, I. Boybat, X. T. Comas, S. Nandakumar, S. Das, A. Sebastian, M. L. Gallo, and P. N. Whatmough, "Analognets: Ml-hw co-design of noise-robust tinyml models and always-on analog compute-in-memory accelerator," *arXiv preprint arXiv:2111.06503*, 2021.

[207] M. Isakov, V. Gadepally, K. M. Gettings, and M. A. Kinsy, "Survey of attacks and defenses on edge-deployed neural networks," in *Proceeding of*

the IEEE High Performance Extreme Computing Conference (HPEC), 2019, pp. 1–8.

[208] D. Kudithipudi, M. Aguilar-Simon, J. Babb, M. Bazhenov, D. Blackiston, J. Bongard, A. P. Brna, S. C. Raja, N. Cheney, J. Clune *et al.*, "Biological underpinnings for lifelong learning machines," *Nature Machine Intelligence*, vol. 4, no. 3, pp. 196–210, 2022.

[209] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[210] H. Li, A. Krishnan, J. Wu, S. Kolouri, P. K. Pilly, and V. Braverman, "Lifelong learning with sketched structural regularization," in *Asian Conference on Machine Learning*. PMLR, 2021, pp. 985–1000.

[211] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, and L. Benini, "A tinyml platform for on-device continual learning with quantized latent replays," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 789–802, 2021.

[212] V. Karia, F. T. Zohora, N. Soures, and D. Kudithipudi, "Scolar: A spiking digital accelerator with dual fixed point for continual learning," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 1372–1376.

[213] R. Zhao, W. Luk, C. Xiong, X. Niu, and K. H. Tsoi, "On the challenges in programming mixed-precision deep neural networks," in *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2020, pp. 20–28.