ACTION: Automated Hardware-Software Codesign Framework for Low-precision Numerical Format SelecTION in TinyML

Hamed F. Langroudi(✉), Vedant Karia, Tej Pandit, Becky Mashaido, and Dhireesha Kudithipudi

Neuromorphic AI Lab, University of Texas at San Antonio, San Antonio, TX, USA
seyedhamed.fatemilangroudi@utsa.edu

Abstract. In this paper, a new low-precision hardware-software codesign framework is presented, to optimally select the numerical formats and bit-precision for TinyML models and benchmarks. The selection is performed by integer linear programming using constraints mandated by tiny edge devices. Practitioners can use the proposed framework to reduce design costs in the early stages of designing accelerators for TinyML models. The efficacy of various numerical formats is studied within a new low-precision framework, ACTION. Results assert that generalized posit and tapered fixed are suitable numerical formats for TinyML when the trade-off between accuracy and hardware complexity is desired.

Keywords: Deep neural networks · Low-precision arithmetic · Hardware-Software Codesign

1 Introduction

TinyML is an emerging machine learning (ML) field that aims to bring intelligence on ubiquitous tiny edge platforms with $\leq$ 1 MB memory footprint, 100 MOPS (million operations per second) throughput, and $\leq$ 1 mW power consumption [1]. The capability to perform ML inference on edge devices enabled by TinyML, can expand the scope of ML applications to new areas such as nature conservation [2], and STEM education [3]. Moreover, the on-device inference capabilities provided by TinyML bypass the latency and energy consumption of data transition between the device and cloud to enhance privacy and security. However, the resource limitations of edge devices introduce significant challenges to perform on-device ML inference on current TinyML models with thousands of parameters and millions of computations [4].

Quite often, to deploy TinyML models on tiny edge devices, the ML inference is performed with low-precision numerical formats [5–11]. The low-precision numerical format offers complexity reduction in multiple dimensions, such as computational resources, energy and memory footprint [5,11]. However, the benefits of

H.F. Langroudi and V. Karia—Equal contribution.

J. Gustafson and V. Dimitrov (Eds.): CoNGA 2022, LNCS 13253, pp. 50–65, 2022.
https://doi.org/10.1007/978-3-031-09779-9_4

low-precision numerical format come at the expense of model performance [12]. The trade-off between hardware complexity and accuracy loss differs between different numerical format configurations [13]. The non-uniform numerical format such as posit [14] has better accuracy and more hardware complexity compared to a hardware-oriented and equispaced numerical format such as fixed-point. This incongruence in accuracy and hardware complexity offered by various numerical formats introduces a broad and large design space for numerical format exploration. Tangential to this, the hardware and model performance constraints are varied from one edge device to another. The process of manual selection of numerical format is ad-hoc and sub-optimal due to the large design exploration space and variability in constraints. Therefore, the process of selecting the optimal numerical format for a TinyML target requires an automatic hardware-software co-design framework that considers model performance and hardware complexity constraints. Such a framework can be used by practitioners and startups as an Early-DSE [15] (early stage design space exploration) framework that generates the template for a suitable accelerator, including the numerical format specification (to aid in reducing the cost of the accelerator's design). Other frameworks that automatically select an appropriate low-precision numerical format based on constraints that are mandated by TinyML model performance and tiny edge platform limitations have been previously proposed in literature [5–7,11]. However, the scope of these existing hardware-software co-design frameworks have been limited to the selection of a bit-precision of the fixed-point numerical format for a particular layer of a TinyML model [5,6]. Moreover, the current frameworks to select a low-precision numerical format use computationally intensive reinforcement learning (RL) algorithms with high sensitivity to initial parameter selection [5,6].

Therefore, we propose a hardware-software co-design framework, called ACTION, that finds the optimal numerical format configuration through integer linear programming (ILP) inspired from recent studies in mixed-precision quantization [16,17]. Using ILP optimization instead of RL reduces the search time, bypasses the need for hyperparameter optimization, and reduces computational overheads [16]. Specifically, the optimal numerical format configuration achieved through the ILP solver minimizes or maximizes one of the objective metrics (e.g., accuracy) while the other subjective constraints (e.g., latency, memory footprint) are met. Unlike the existing frameworks, ACTION, supports a broad range of numerical formats including posit and generalized posit, summing up to a total of 60 possible numerical format configurations.

The key contributions of this work are as follows:

1. We develop a low-precision hardware-software co-design framework to constrain the early stage design space exploration which selects an appropriate numerical format based on the custom user defined constraints through integer linear programming optimization.
2. Various configurations and dataflows in a systolic array based architecture are studied to evaluate the performance of the numerical formats when incorporated in an accelerator.

2 Background

A non-zero finite real number y is represented by Eq. (1) where s is the sign, L is the bit array, ϕ is a function mapping the bit array $f \in [0, 1)$ as a fraction to a real value, and $\star$ is an arbitrary function between the integer and the fraction (in this study $\star \in \{\times, +\}$)

$$y = (-1)^s \times \psi(L) \star \phi(f) \tag{1}$$

The numerical format used in this study is summarized in Table 1, based on Eq. (1). Note that all numerical formats use a two's complement representation to represent a negative number except for the floating point numerical format, which uses a sign-magnitude representation. The main difference between these numerical formats is the way that the bit array L is encoded. In traditional numerical formats such as fixed and floating point, L is binary(B) and offset-binary(OB) encoded respectively while in recent numerical formats such as tapered fixed-point (taper [18]), L is signed unary encoded or regime encoded (RE) where the *runlength* m of identical bits $(l...l)$ is terminated by either an compliment bit $\bar{l}$ where $m \leq n$ or by a final bit. Hence the value R in regime encoding is computed as (2).

$$R = \begin{cases} -m, & l = 0 \\ m - 1, & l = 1 \end{cases} \tag{2}$$

In posit and generalized posit numerical formats the bit string l is divided into two parts, the regime and the exponent. The regime bit array is singed unary encoded and the exponent bit array is binary encoded. The signed unary encoding is a variable encoding that adds a tapered accuracy attribute to the posit, generalized posit and tapered fixed-point formats. In numerical formats with **tapered-accuracy**, the density of values is highest near 0 and then tapers towards the maximum-representable number as shown in Fig. 1.

Table 1. Description of numerical formats that are explored in this study.

Format	L Encode	$\psi(L)$	$\star$	$\phi(f)$	Parameters
Fixed-point	B	L	$+$	f	–
Tapered fixed-point	RE	R	$+$	f	Is, sc
Floating point	OB	$2^{e-2^{es-1}-1}$	$\times$	$1+f$	–
Posit	RE, B	$2^{2^{es}R+e}$	$\times$	$1+f$	–
Generalized posit	RE, B	$2^{2^{es}R+e+e_b}$	$\times$	$1+f$	rs,e_b

The numerical formats with a tapered-accuracy characteristic are more appropriate to represent TinyML model parameters (weights) due to their bell-shaped distribution [9].

Among numerical formats with the tapered-accuracy characteristic, only generalized posit and tapered fixed-point can accommodate the variability observed in a layer's parameter distribution by assigning two additional hyperparameters that can modify it's dynamic range and tapered precision [7,19]. Ordinarily, the maximum accuracy is located at 1 in posit, generalized posit and tapered fixed-point formats. The Exponent bias (e_b) and scaling factor (sc) can re-center the location of maximum accuracy from 1 to 2^{e_b} or 2^{sc}. The dynamic range and shape of the numerical format values' distribution (maximum tapered to uniform) is controlled by a maximum regime/integer run-length (rs/Is) parameter.

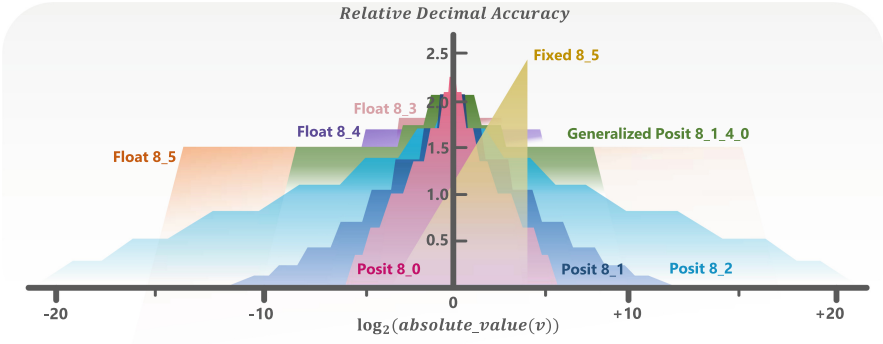

Fig. 1. The relative decimal accuracy [14] for various 8-bit numerical formats Float 8_5 , Float 8_4, Float 8_3 are 8-bit floating format with 5, 4 and 3 exponent bits, respectively, and Posit 8_0, Posit 8_1, and Posit 8_2 are 8-bit posit format with 0, 1, and 2 exponent bits respectively. The Fixed 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 $e_b = 0$.

3 Related Work

In recent years, the impact of various low-precision numerical formats on deep learning inference accuracy has been studied thoroughly [12,13,20–22]. For instance, Gysel *et al.* proposed the Ristretto framework to explore the effect of fixed-point, minifloat (8-bit floating point format with arbitrary exponent and fraction bit-width), and block floating point (where each block of floating point numbers used a shared exponent) on classification accuracy [20]. The outcome of this study on the CIFAR-10 corpus shows negligible accuracy difference between DNN inference with an AlexNet model using 8-bit and 32-bit floating point format parameters.

However, a few works proposed empirical frameworks that demonstrate the effect of numerical formats on the trade-off between performance and hardware complexity. For instance, Hashemi *et al.* demonstrate that DNN inference with 8-bit fixed-point using AlexNet (on CIFAR-10 dataset) results in a 6.8× improvement in energy consumption with <2% accuracy degradation compared to a

DNN inference with 32-bit floating point [13]. Following this work, Langroudi *et al.* introduce the Cheetah framework where the trade-off between inference accuracy and hardware complexity (e.g., energy-delay product (EDP)) is provided for DNN inference with [5, 8] precision posit, float and fixed-point formats [12]. The optimal bit-precision for each numerical format in this framework is obtained through a top-down iterative process where the accuracy and hardware complexity achieved by a numerical format is compared with the specified design constraints provided by practitioners. Through this study, posit shows better accuracy and EDP trade-off as compared to float and fixed-point numerical formats. Recently, Thierry Tambe *et al.* [22] and Langroudi *et al.* [21] introduce two novel numerical formats (adaptive float and adaptive posit) to represent DNN parameters. With negligible hardware overhead, these numerical formats are able to adapt to the dynamic range and distribution of DNN parameters and thus improve inference accuracy [21,22].

The efficacy of numerical formats in terms of hardware complexity and inference accuracy is also evaluated on TinyML models and benchmarks [5–8,10]. However, the variants of fixed-point numerical formats used for these studies and other numerical formats is not evaluated on TinyML models and benchmarks. For instance, Rusci *et al.* demonstrate a mixed 2-, 4-, 8-bits precision fixed-point numerical format to perform TinyML inference on MCU devices with low-memory constraints (e.g., 2 MB) [6]. In this study, the automatic bit precision assignment policy for parameters across layers are selected through a reinforcement learning algorithm. On MobileNet V2 and ImageNet dataset, the aforementioned mixed-precision quanization approach results in about 1.3% inference accuracy degradation as compared to inference accuracy with 32-bit floats. Recently, Langroudi *et al.* introduce an efficient method of quantizing TinyML models using a novel tapered fixed-point numerical format that leverages the benefit of both posit (in terms of accuracy performance) and fixed-point (in terms of hardware efficiency) [7]. The tapered fixed-point has shown better EDP and accuracy trade-off over fixed-point on various benchmarks [7].

This research proposes the ACTION, framework for TinyML models where the numerical format and bit-precision of model parameters is automatically selected through ILP optimization. A notable difference between this work and previous works is that the ACTION framework supports a broad range of numerical formats and its search space exploration time is an order of magnitude faster than previous RL approach [6].

4 ACTION Framework

The goal of ACTION framework is to automatically and swiftly select the appropriate numerical format based on constraints required by TinyML benchmarks and tiny edge devices. This platform can be generalized for use on other DNN models and edge devices since it provides the ability for practitioners to choose their own constraints. This framework comprises of four key aspects as shown in Fig. 2: User Interface, Initialization, Optimizer, and Evaluator.

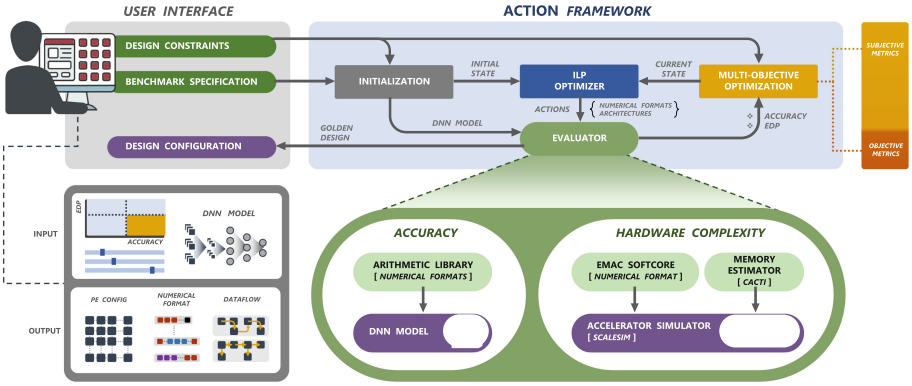

Fig. 2. The ACTION high-level low-precision hardware & software co-design framework for TinyML models on tiny edge platforms

4.1 User Interface

The goal of the user interface is to preselect the metrics and parameters given as input to the framework. These include benchmark specifications, models, datasets (e.g., TinyML v0.5 benchmark), metrics, constraints, and variables (summarized in Table 2, and 3). The framework then generates specifications of the accelerator such as numerical format configuration, PE configurations, memory requirements and data flows, that are summarized in an output file. The input and output of the user interface are specified in YAML format.

4.2 Initialization

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. The specification of the TinyML model that is used by the accuracy and hardware complexity evaluator is automatically generated.

4.3 Evaluator

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 operation clearly, although it can be generalized for any TinyML model.

Software Design and Exploration: A computational node in a single hidden layer convolutional neural network computes (3) 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 values 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: the number of filter channels, the filter heights, and

the filter weights respectively. The computation in (3) is performed N times, where N is a product of batch size, output activation size (height and width) and the number of filters.

$$Y_j = \theta(Q(B_j) + \sum_{i=0}^{M} Q(A_i) \times Q(W_{ij})) \tag{3}$$

In this work, each 32-bit floating point TinyML parameter (x_i) is mapped to a l-bit low-precision numerical format value (x_i') through the quantization function as defined in (4), where s and z are the scaling factor and zero point, respectively. Large magnitude 32-bit floating point numbers that are not expressible in $[l, u]$ (low-precision numerical format values range) are clipped either to the format lowerbound (l) and upperbound (u). Moreover, the clipped values that lie in interval $[a, b]$ (the two consecutive low-precision numerical format values) are rounded to the nearest even number.

$$x_i' = Q(x_i, q, l, u, s, z) = \text{Round}(\text{Clip}(s \times x_i + z, l, u)) \tag{4}$$

The product of quantized activations and weights are computed with a low-precision numerical format multiplier without rounding the end products. The products are then accumulated over wide signed fixed-point register, the *quire* [14]. Note that this MAC operation for m operands is error free since quire size, as shown in (5), is selected in a way that the dynamic range of partial accumulated values are captured. The D_l in (5) represents the dynamic range of the low-precision numerical format.

$$w_{quire} = \lceil \log_2(m) \rceil + 2 \times \lceil \log_2(D_l) \rceil + 2 \tag{5}$$

Hardware System Design and Architecture: To evaluate the area, energy and latency of the hardware accelerator, analyzing only the hardware complexity introduced by various numerical formats can be misleading. Existing frameworks that evaluate the performance of the numerical formats compare only the energy consumption of the MAC operations which overlooks the constraints imposed by memory and dataflow in the accelerator.

In order to evaluate the area, and power of an accelerator which incorporates a particular numerical format, we limit the evaluation to an architecture comprising of Processing Elements (PE) arranged in a 2D systolic array configuration. The compute efficiency of the systolic array architectures depends highly on the dataflow and the PE array size since the matrix multiplication operation of a TinyML model is mapped to the PEs arranged in the 2D matrix structure. To estimate the latency of the system we bridge our framework with the SCALE-Sim tool [23] by simulating the TinyML models for various configurations of PE and dataflows as illustrated in Fig. 3.

To analyze the energy consumption and the area of the system, each PE is replaced with the multiply and accumulate (MAC) unit of the numerical formats stated in Table 3. The MAC units for various numerical formats and different configurations were synthesized on the Synopsys 32 nm CMOS technology node. Power consumption and the area of individual MAC units were combined with the cycle count and memory access details obtained from SCALE-Sim to analyze the hardware complexity of the accelerators with various incorporated numerical format configurations.

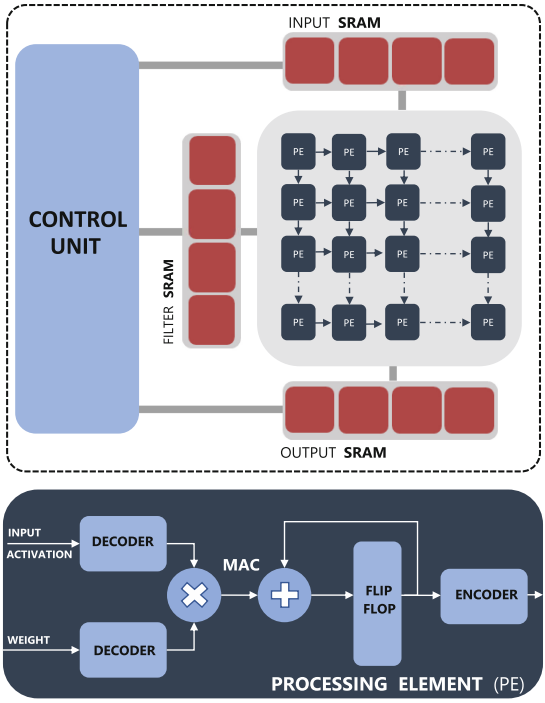

Fig. 3. The overview of the accelerator's system architecture used to evaluate the efficacy of the numerical formats on accelerators.

4.4 Optimizer

Integer Linear Programming (ILP): The ILP in this work is defined as (6) where y_i and x_i are objective and subjective metrics selected from Table 2, A as configuration sets from Table 3, C_j are constraints with respect to subjective x_j.

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

For instance the accuracy degradation (ACC_d) is selected as an objective, and area, EDP, memory footprint, and MAC frequency are chosen as subjectives, the (6) as in (7). In this study, we set the maximum number of subjective metrics to 4. On occasion, the subjective metrics have some overlap, such as power and EDP.

$$\begin{aligned} \min_{ACC_d} \quad & ACC_d(A) \\ \text{s.t.} \quad & \text{EDP}(A) < \text{EDP Constraint} \\ & \text{Area}(A) < \text{Area Constraint} \\ & \text{Memory footprint}(A) < \text{Memory footprint. Constraint} \\ & \text{MAC Frequency}(A) < \text{MAC Frequency Constraint} \end{aligned} \tag{7}$$

5 Experimental Setup, Results and Analysis

The ACTION framework is implemented in TensorFlow [24]. A summary of the metrics and constraints specifications are presented in Table 2. The current version of ACTION framework supports 8 metrics crucial for TinyML applications. For specific metrics, constraints are selected at 3 intervals between the best and worst performance yielding values. Note that in some cases, the best possible result for a specific metric may not meet the TinyML target. This shows that the low-precision arithmetic needs to combine with other hardware/software optimizations such as pruning [25] and processing in memory [26] to meet that specific metric for TinyML. The specification of key variables and their configurations are summarized in Table 3. Note that the generalized posit hyperparameters ($rs \in [1..n-1]$ and $e_b \in [-3, 3]$) and tapered fixed-point ($Is \in [1..n]$ and $sc \in [-3, 3]$) are not mentioned in Table 3 since these values are fixed and predetermined based on the dynamic range and distribution of parameters [7,19]. The specifications of the tasks and inference performance with 32-bit floats are summarized in Table 4. To estimate latency, we bridge our framework with the SCALE-Sim tool [23]. SCALE-Sim, however, does not consider the cycles consumed while 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 the energy estimation analysis, execution time, and power consumption, we consider the use of the 32-nm CMOS technology node.

5.1 Numerical Formats' Performance on TinyML Benchmark

The Table 5 are summarized the performance of various numerical formats on TinyML v0.5 benchmark that evaluated using ACTION framework. Amongst the evaluated numerical formats, generalized posit shows the best performance. For instance, the inference accuracy on the image classification benchmark using generalized posit is improved by an average of 6.70%, 19.92%, 8.66%, 30.02%, as compared to posit, float, tapered fixed-point and fixed-point respectively. The

Table 2. The metrics and constraints specification.

Categ	Metrics	Constraints	TinyML target
1	EDP	[mean-std,mean,mean+std]	–
	Energy		–
	Power		≤ 1 mw [1]
2	MAC Frequency		10–100 MHz [27]
3	PE utilization		–
4	Area		< 20 mm^2 [27]
6	Accuracy Degradation		1–6% [1]
7	Memory footprint		≤ 100 KB + 0.5 MB [28]

Table 3. The key variable specification (P:Posit, FP:Floating point, FX: Fixed-point, GP: generalized posit, and TFX: tapered fixed-point).

Variable	Configuration	Search space
Formats	P($n \in [5..8]$, $es = [0..2]$)	60
	FP($n \in [5..8]$, $e = [3..n-2]$)	
	FX($n \in [5..8]$, $f = [1..n-1]$)	
	GP($n \in [5..8]$, $es = [0..2]$)	
	TFX($n \in [5..8]$)	
PE	32×32, 32×16, 16×16, 12×14, 8×8	5
Data-flow	OS, WS, IS	3
Total	–	900

Table 4. The TinyML v0.5 [4] models and benchmarks using 32-bit float parameters description.

Application	Dataset	DNN Model	# Parameters	# Ops	Performance
Keyword spotting	Speech commands v2	DS-CNN	24.91 K	5.54 M	92.15%
Visual wake words	VWW dataset	MobileNetV1	221.79 K	15.69 M	82.72%
Image classification	CIFAR10	ResNet-8	78.67 K	25.27 M	86.26%

high performance of the generalized posit numerical format on TinyML benchmarks can be credited to the capability of this numerical format to auto-adjust to the dynamic range and distribution of the weights and activations. Moreover, we observed that the performance of tapered fixed-point is not only better than fixed-point, but also, on average, comparable with floats and posit formats, which has not been previously observed [7]. This finding emphasizes that tapered fixed-point is a good candidate for TinyML models and applications. Moreover, as the number of bits is decreased to 7-bits and below, the float, fixed-point and tapered fixed-point formats show poor accuracy performance. This can be attributed to

Table 5. The TinyML inference performance using various numerical formats on TinyML v0.5 benchmark (P:Posit, FP:Floating point, FX: Fixed-point, GP: Generalized posit, and TFX: Tapered fixed-point).

Format	Keyword Spotting				Image classification			
	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
P	91.97%	85.33%	48.62%	23.79%	85.31%	77.28%	53.78%	26.19%
FP	86.45%	46.29%	13.72%	8.50%	82.10%	69.45%	11.28%	10.79%
FX	12.70%	8.87%	8.39%	8.22%	37.00%	21.80%	15.97%	12.90%
GP	**92.10%**	**91.39%**	**88.14%**	**49.62%**	**85.81%**	**83.90%**	**76.36%**	**42.72%**
TFX	79.20%	43.75%	8.52%	8.43%	85.24%	82.10%	38.60%	17.72%
32-bit FP	92.15%				86.26%			

Format	Visual Wake Words			
	8-bit	7-bit	6-bit	5-bit
P	83.02%	80.58%	74.53%	66.28%
FP	80.02%	68.25%	59.95%	59.37%
FX	76.43%	72.06%	61.86%	60.71%
GP	**83.02%**	**82.14%**	**76.72%**	**69.97%**
TFX	82.97%	81.92%	76.00%	66.76%
32-bit FP	82.26%			

discrepancy between the dynamic range provided by these numerical formats and the actual dynamic range of weights and activations.

5.2 ACTION Framework Results

Figures 4 and 5 illustrate the performance of each numerical format incorporated on the various configurations of accelerator and dataflows. The ILP optimization identified the optimal numerical format much quicker ($\leq$1 s, performed on Intel i9-9960X) than the tedious and iterative process undertaken by reinforcement learning optimization algorithms (which can take several hours [5]). 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 the accuracy vs. MAC frequency trade-off (Figs. 4.b and 5.b), the numerical formats are selected in 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 frequency when the accuracy constraints are satisfied.

To evaluate the efficacy of the numerical formats on a custom accelerator, the framework uses the SCALE-Sim simulator which outputs the estimated cycle

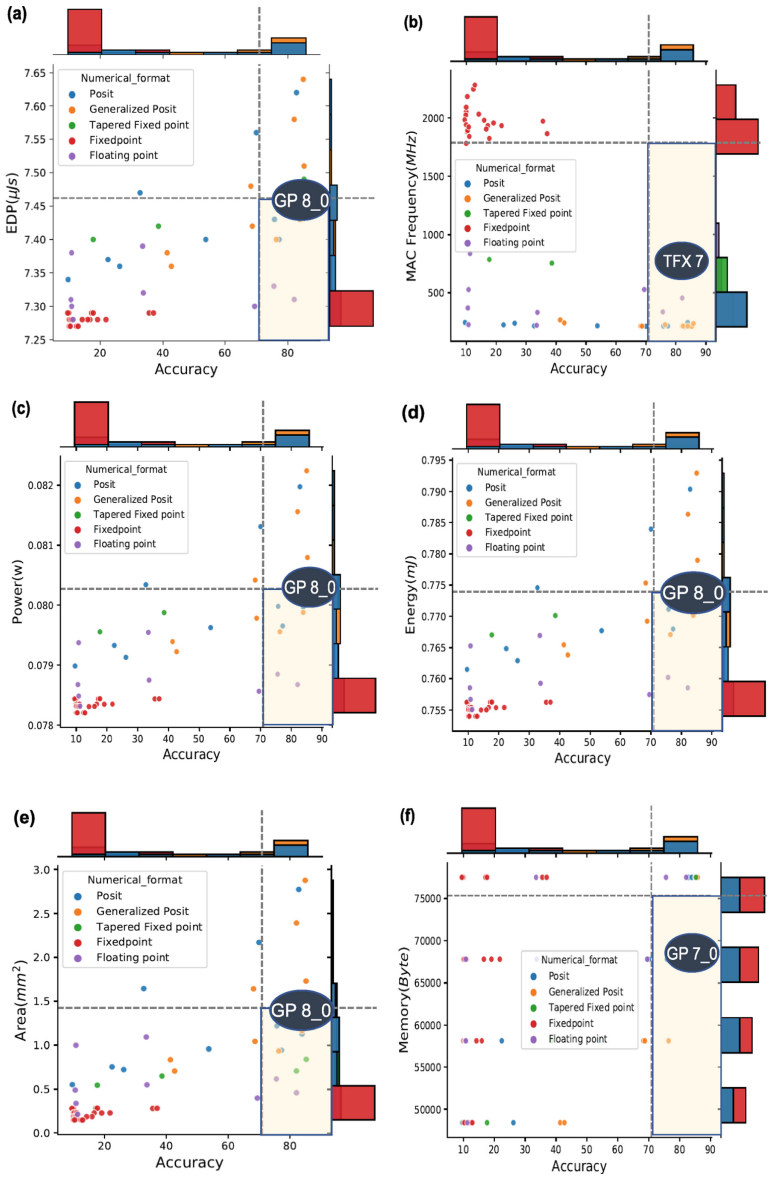

Fig. 4. (a) EDP vs Accuracy (b) MAC frequency vs. Accuracy (c) Power vs. Accuracy (d) Energy vs. Accuracy (e) Area vs Accuracy (f) Memory vs Accuracy for an image classification task with an accelerator configured with PEs arranged in a 16 × 16 systolic array and output stationary dataflow. The constraint 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.

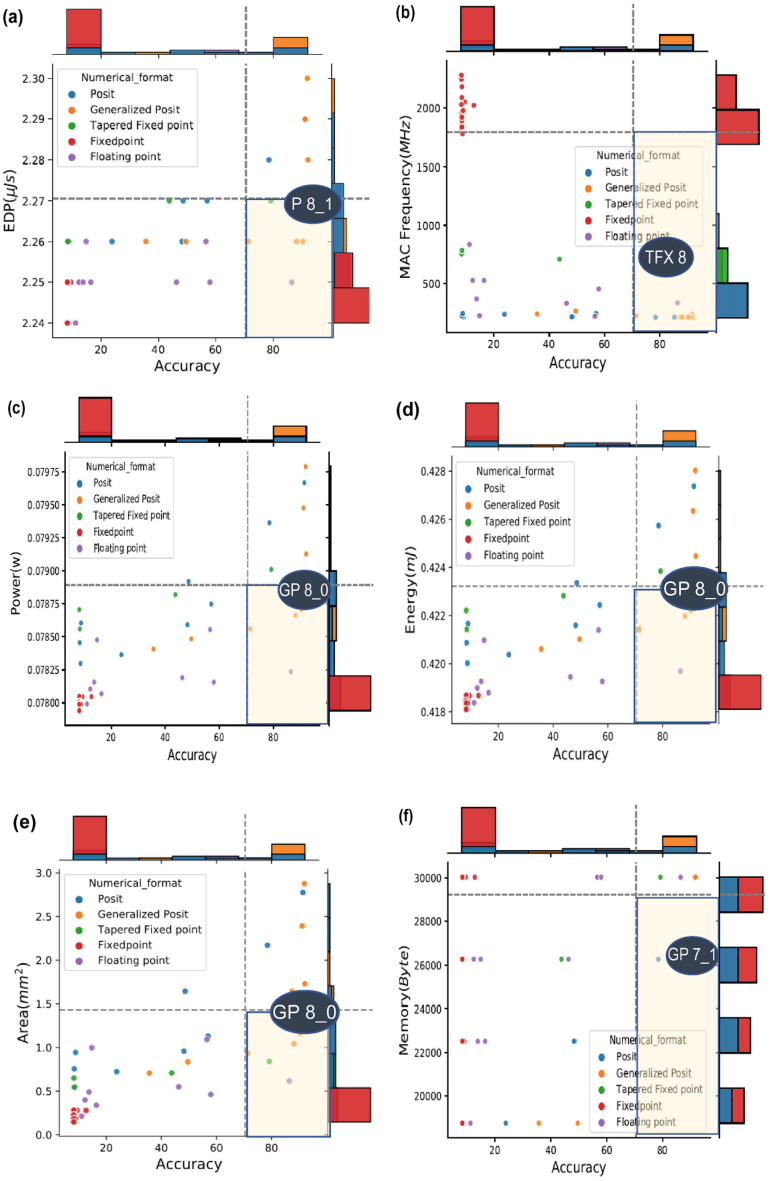

Fig. 5. (a) EDP vs Accuracy (b) MAC frequency vs. Accuracy (c) Power vs. Accuracy (d) Energy vs. Accuracy (e) Area vs Accuracy (f) Memory vs Accuracy for keyboard spotting task with an accelerator configured with PEs arranged in a 16 × 16 systolic array and output stationary dataflow. The constraint 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.

count and the memory data movement sequences for executing a CNN model on a custom configuration. The cycle count and the data movement are combined with results obtained by synthesizing the MAC unit of each numerical format to generate the EDP and latency of the accelerator. Various dataflows and PE matrix array configurations were compared against the output stationary dataflow, which outperformed the other dataflows and offered a 24% reduction in latency as compared to the weight stationary dataflow in particular, for a single inference cycle. It has also shown significant improvement in EDP and utilization factor as compared to the input stationary and weight stationary dataflows. Moreover, generalized posit has outperformed all the other formats in Keyword Spotting and Image Classification tasks with minor overhead in EDP.

6 Conclusions

Through the ACTION framework, we propose a hardware-software co-design framework for early stage design space exploration to discover the optimal numerical formats and accelerator configurations based on custom user defined constraints. The configuration selection problem is solved by integer linear programming (ILP), which allows us to identify the optimal numerical format and accelerator configuration faster than reinforcement learning approaches. We show that generalized posit yields a 16% improvement in the average inference accuracy over the other numerical formats that are considered in this paper.

Acknowledgement. This research was supported by the Matrix AI Consortium for Human Well-Being at UTSA. The authors would like to thank Dr. John Gustafson, who is the inventor of Posit, Generalized posit and Tapered Fixed-point and has provided valuable insights over the years. The authors would also like to express gratitude to NUAI lab members at RIT and UTSA who supported this research study.

References

1. Banbury, C., et al.: Mlperf tiny benchmark. In: Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (2021)
2. Curnick, D.J., et al.: Smallsats: a new technological frontier in ecology and conservation? Remote Sens. Ecol. Conserv. **8**(2), 139–150 (2021)
3. Reddi, V.J., et al.: Widening access to applied machine learning with tinyml. arXiv preprint arXiv:2106.04008 (2021)
4. Banbury, C.R., et al.: Benchmarking tinyml systems: Challenges and direction. arXiv preprint arXiv:2003.04821 (2020)
5. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: Haq: hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8612–8620 (2019)
6. Rusci, M., Fariselli, M., Capotondi, A., Benini, L.: Leveraging automated mixed-low-precision quantization for tiny edge microcontrollers. In: Gama, J., et al. (eds.) ITEM/IoT Streams -2020. CCIS, vol. 1325, pp. 296–308. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-66770-2_22

7. Langroudi, H.F., Karia, V., Pandit, T., Kudithipudi, D.: Tent: Efficient quantization of neural networks on the tiny edge with tapered fixed point. arXiv preprint arXiv:2104.02233 (2021)
8. Ghamari, S., et al.: Quantization-guided training for compact tinyml models. arXiv preprint arXiv:2103.06231 (2021)
9. Fahim, F., et al.: hls4ml: an open-source codesign workflow to empower scientific low-power machine learning devices. arXiv preprint arXiv:2103.05579 (2021)
10. Ravaglia, L., Rusci, M., Nadalini, D., Capotondi, A., Conti, F., Benini, L.: A tinyml platform for on-device continual learning with quantized latent replays. IEEE J. Emerg. Sel. Top. Circuits Syst. **11**(4), 789–802 (2021)
11. Lin, J., Chen, W.M., Cai, H., Gan, C., Han, S.: Mcunetv2: memory-efficient patch-based inference for tiny deep learning. arXiv e-prints pp. arXiv-2110 (2021)
12. Langroudi, H.F., Carmichael, Z., Pastuch, D., Kudithipudi, D.: Cheetah: mixed low-precision hardware & software co-design framework for dnns on the edge. arXiv preprint arXiv:1908.02386 (2019)
13. Hashemi, S., Anthony, N., Tann, H., Bahar, R.I., Reda, S.: Understanding the impact of precision quantization on the accuracy and energy of neural networks. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1474–1479. IEEE (2017)
14. Gustafson, J.L., Yonemoto, I.T.: Beating floating point at its own game: posit arithmetic. Supercomputing Front. Innov. **4**(2), 71–86 (2017)
15. Brumar, I., Zacharopoulos, G., Yao, Y., Rama, S., Wei, G.Y., Brooks, D.: Early DSE and automatic generation of coarse grained merged accelerators. arXiv preprint arXiv:2111.09222 (2021)
16. Yao, Z., et al.: Hawq-v3: Dyadic neural network quantization. In: International Conference on Machine Learning, pp. 11875–11886. PMLR (2021)
17. Hubara, I., Nahshan, Y., Hanani, Y., Banner, R., Soudry, D.: Improving post training neural quantization: layer-wise calibration and integer programming. arXiv preprint arXiv:2006.10518 (2020)
18. Gustafson, L.J.: A generalized framework for matching arithmetic format to application requirements. https://posithub.org/ (2020)
19. Langroudi, H.F., et al.: Alps: adaptive quantization of deep neural networks with generalized posits. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3100–3109 (2021)
20. Gysel, P., Pimentel, J., Motamedi, M., Ghiasi, S.: Ristretto: a framework for empirical study of resource-efficient inference in convolutional neural networks. IEEE Trans. Neural Netw. Learn. Syst. **29**(11), 5784–5789 (2018)
21. Langroudi, H.F., Karia, V., Gustafson, J.L., Kudithipudi, D.: 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, pp. 726–727 (2020)
22. Tambe, T., et al.: Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. In: 2020 57th ACM/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE (2020)
23. Samajdar, A., Zhu, Y., Whatmough, P., Mattina, M., Krishna, T.: Scale-sim: Systolic CNN accelerator simulator. arXiv preprint arXiv:1811.02883 (2018)
24. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/, software available from tensorflow.org
25. Li, S., Romaszkan, W., Graening, A., Gupta, P.: Swis-shared weight bit sparsity for efficient neural network acceleration. arXiv preprint arXiv:2103.01308 (2021)

26. Zhou, C., et al.: Analognets: Ml-hw co-design of noise-robust tinyml models and always-on analog compute-in-memory accelerator. arXiv preprint arXiv:2111.06503 (2021)
27. Gousev, E.: Recent progress on tinyml technologies and opportunities. https://www.youtube.com/ (2021)
28. Banbury, C., et al.: Micronets: neural network architectures for deploying tinyml applications on commodity microcontrollers. In: Proceedings of Machine Learning and Systems, vol. 3 (2021)