Field-Configurable Multi-resolution Inference: Rethinking Quantization

Sai Qian Zhang [1], Bradley McDanel [2], H.T. Kung [1], and Xin Dong [1]

[1]Harvard University
[2]Franklin and Marshall College

Abstract

Departing from traditional quantization for a fixed quantization resolution, we describe a novel architecture approach to support inference at multiple resolution deployment points. A single meta multi-resolution model with a small footprint can select from multiple resolutions at runtime to satisfy given resource constraints. The proposed scheme relies on term quantization to enable flexible bit annihilation at any position for a value in a context of a group of values. This is in contrast to conventional uniform quantization which always truncates the lowest-order bits. We present multi-resolution training of the meta model and field-configurable multi-resolution Multiplier-Accumulator (mMAC) design. We compare our design against a traditional MAC design and evaluate the inference performance on a variety of datasets including ImageNet, COCO, and WikiText-2.

1. Introduction

Deep Neural Network (DNN) quantization has received much attention in recent years due to its potential to address the growing computational costs of DNN inference. By quantizing a 32-bit full-precision models to a lower resolution (*e.g.,* 8-bit fixed-point), quantized DNNs can be implemented using more efficient hardware, such as 8-bit Multiplier-Accumulator (MAC) units instead of floating-point units, leading to significant improvements in energy efficiency, latency, and storage requirements. However, there is an inherent trade-off between the performance (*e.g.,* classification accuracy) and the precision of a DNN under a fixed quantization regime. Due to this trade-off, quantization methods often present multiple distinct models trained at different precisions (*e.g.,* from 8-bit to 4-bit fixed-point bit-widths) that achieve varying degrees of performance/cost trade-off.

Unfortunately, conventional hardware is designed to only natively support a single precision, such as 8-bit fixed-point, and therefore can not efficiently implement quantized models across a wide range of precisions to support such a performance/cost trade-off. For instance, while an 8-bit MAC can be used to multiply two 4-bit numbers, the upper-half of the MAC will only be multiplying zeros bits.

To address this concern, we present a multi-resolution Multiplier-Accumulator (mMAC) design that inherently supports multiple resolutions. The mMAC operates on only the non-zero power-of-two terms in a value. For example, for the value $20 = 00010100_2$, mMAC only operates on the two terms, 2^4 and 2^2, corresponding to the two nonzero bits in the unsigned binary representation of the value. The approach also generalizes to signed-digit representations (SDRs) which can have both positive and negative terms. Consequently, the mMAC takes fewer cycles to multiply lower-resolution numbers (*i.e.,* numbers with few non-zero terms) and more cycles to multiply higher-resolution numbers with more non-zero terms. Note that, unlike conventional uniform quantization where resolution is tied to the bit-width of the representation, our mMAC design defines resolution as **the number of non-zero power-of-two terms in values, regardless of their positions in the encoding**.

To support field-configurable multi-resolution inference at a deployment point, we have developed a multi-resolution DNN training approach that jointly optimizes many sub-models across a wide range of resolutions (shown on the left of Figure 1). The result of this joint-optimization training is a meta multi-resolution model capable of supporting multiple resolutions at runtime, with two novel properties: **storage sharing** across the sub-models, as the same non-zero terms (which need not be adjacent) for lower-resolution sub-models also appear in higher-resolution sub-models, and **computation sharing** as all sub-models can share the same mMAC computation engine. In both training and inference, for the same set of DNN weights, we simply adjust the number of leading non-zero terms to implement different quantization resolutions (*i.e.,* sub-models).

The multi-resolution model is deployed with the proposed mMAC system, shown in the middle of Figure 1. A user (or other selection mechanism) can select which sub-model to use based on the current resource constraints in the performance/cost trade-off space (right of Figure 1). Configuring the system to use a low-budget sub-models is achieved by simply dropping more low-order power-of-two terms from the meta multi-resolution model. Since a low-resolution sub-model has fewer terms, the hardware system using mMACs will perform the inference computation at a lower computation cost. Therefore, mMAC can process these lower-resolution inference computations at an increased rate. The multi-resolution joint training procedure is critical to the success of this method, as otherwise, the multi-resolution hardware would incur high cost, and the performance of the lower-budget sub-models would be unacceptably poor. The main contributions of the paper are:

- A multi-resolution hardware system with mMAC for supporting field-configurable multi-resolution DNN inference. The mMAC computes the dot products by processing only

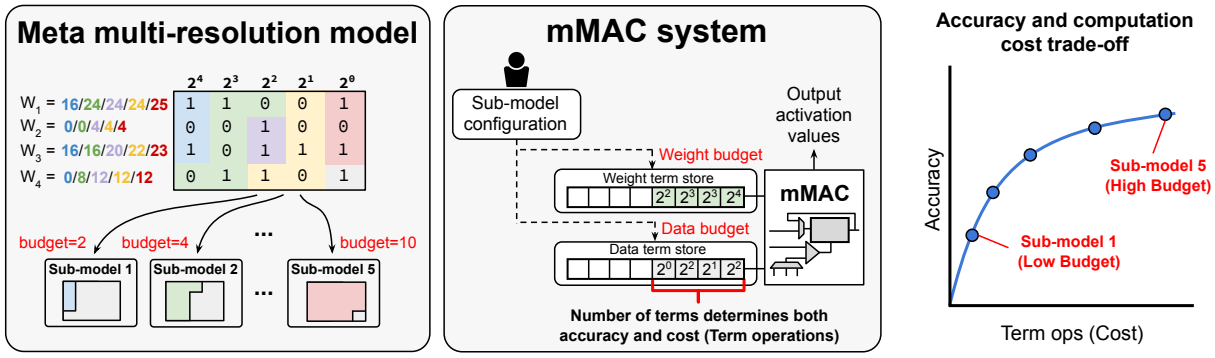


Figure 1: A multi-resolution model (left) contains multiple sub-models with varying power-of-two term budgets leading to different degrees of quantization. Here, a group of 4 weight values (25, 4, 23, 13) are depicted with 5 different term budgets. In a hardware deployment (middle) the multi-resolution model is implemented using an mMAC system designed around a multi-resolution Multiplier-Accumulator (mMAC) which all resolutions can share. The proposed multi-resolution approach enables an efficient cost/performance trade-off to suit the current runtime conditions (right) by selecting the appropriate sub-model with a corresponding resolution.

the non-zero terms in weight and data values.

- A *multi-resolution training paradigm* that supports efficient training of multiple sub-models that share power-of-two terms. The method uses a teacher-student approach to train two sub-models at each iteration.
- *Sub-model configuration* at inference to meet the current resource constraints which translates to simply adjusting the number of leading terms to use in learned weights. The multi-resolution model has a minimum memory footprint, as the sub-model instances share the same leading power-of-two terms across the model weights. In addition, these sub-models can efficiently share the same mMAC.

2. Background and Related Work

In Section 2.1, we discuss related work on neural networks that can dynamically trade-off computation for model performance (*e.g.,* accuracy). After that, we discuss the various types of quantization that have been used for DNNs in Section 2.2. Finally, in Section 2.3, we provide an overview of the Signed-digit Representation (SDR), which we use instead of the conventional Unsigned Binary Representation (UBR), and review specialized hardware accelerators which employ SDR for DNN inference.

2.1. DNN Supporting Performance/Cost Trade-off

In recent years, there has been a trend towards designing dynamic neural network to achieve an on-demand performance/cost trade-off. Many approaches achieve this trade-off by skipping parts of the inference computation based on the complexity of the input sample. In [41, 19, 33], the authors add early-exit branches to a DNN, so that easier input samples can exit at an earlier point in the network with high confidence. BlockDrop [44] and SkipNet [43] selectively drop the convolutional blocks in ResNet [16] architecture on a per-sample basis. In [31], the authors propose an approach to train a single model which can generate sub-models of different widths (*e.g.,* number of channels in each convolutional layer). During runtime, the number of activated channels in a convolutional layer can be adjusted dynamically based on the on-device resource budget. Once-For-All [4] allows for a significantly larger number of DNN architectural settings by exploring a greater design space (*e.g.,* depth, width, kernel size, and resolution) and using a teacher-student training paradigm. In contrast to these works, where the derived sub-models share **weight values**, our work proposes a complementary multi-resolution DNN approach by sharing **weight terms**. Our approach allows for greater sharing flexibility in weight representations and therefore a better performance/cost trade-off.

2.2. Types of Quantization

Quantization [8, 9, 14, 46, 20, 37, 24] has been studied extensively for reducing the associated storage, I/O, and computation costs of DNNs. Several post-training quantization methods have been proposed to quantize floating-point weights after training, using 16-bit and 8-bit uniform quantization (UQ), without dramatically impacting classification accuracy [14, 12]. Recently, low-precision UQ approaches (*e.g.,* with less than 4 bits per value) have also been studied. To mitigate the accuracy degradation caused by low-precision weight and data, additional quantization-aware training [21] is required to fine-tune the model weights. Figure 2(a)-(b) show 5-bit UQ and 2-bit UQ applied on four values.

Logarithmic quantization (LQ) is a more aggressive form of quantization that works by rounding each value to the nearest power-of-two term as shown in Figure 2(c) [35, 45]. This allows for significantly more efficient inference, as fixed-point multiplication in UQ can be replaced with bit shift operations as each value has only a single power-of-two term. However, due to the aggressive form of quantization, LQ typically suffers from larger accuracy degradation than UQ, as the resolution decreases exponentially as values gets larger.

Power-of-two **term quantization** (TQ) relaxes LQ by al-

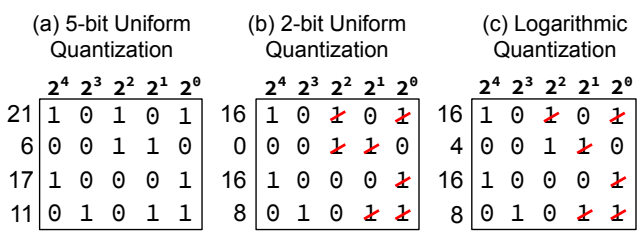


Figure 2: (a) 5-bit uniform quantization applied on four values. (b) 2-bit uniform quantization takes only the two leading terms (2^4 and 2^3) of the 5-bit uniform quantization in (a). (c) Logarithmic quantization uses only the largest term in each value.

lowing a term budget of one or more terms for values [27]. Unlike prior work that applied TQ in a post-training quantization fashion [27], in this work, we propose a multi-resolution training paradigm using TQ as our quantization function for weights and data. We discuss TQ in greater detail in Section 3, as it is the quantization scheme used throughout this paper.

2.3. DNN Hardware using Signed-digit Representations

Unsigned Binary Representations (UBRs) are a commonly used positional encoding system, where each position is either 0 or 1. By comparison, in Signed-digit Representations (SDRs), each position can have a coefficient of $\{-1, 0, 1\}$. Note that this means each digit in an SDR requires two bits: a sign bit and a $0/1$ bit. Allowing each position to have a negative coefficient leads to representations with fewer nonzero digits. For instance, 27 (11011 in UBR) can be represented as $100\bar{1}0\bar{1}$ in SDR.[1] SDRs have been extensively studied in the past [3, 2, 13], but have received less attention in relation to DNN hardware accelerators [6, 5, 36, 32, 15, 26].

Some recent works [1, 10, 39, 27] have observed that the distribution of DNN weight values, leading to most values having few nonzero terms, can be exploited to reduce the computational cost of DNN inference. Specifically, multiplication between a weight and data value can be decomposed into exponent additions between power-of-two term-pairs in the two values. Through this view, SDRs are an attractive representation as they reduces the number of terms in each value, leading to fewer term pairs. (In fact, it is known in the literature that SDR can achieve the minimum number of terms [22]). In this work, our proposed multi-resolution hardware architecture (Section 5) is designed to support weight and data values encoded in SDRs.

3. Term Quantization

In this section, we provide an overview of term quantization (TQ), which is used throughout the paper. First, in Section 3.1, we show how TQ can be applied to group of weights in a DNN and discuss the associated quantization error based on the distribution of weights. Then, in Section 3.2, we illustrate how TQ can be applied to individual data values. Finally, we

[1] Here, $\bar{1}$ represents a negative coefficient (*e.g.*, -2^0).

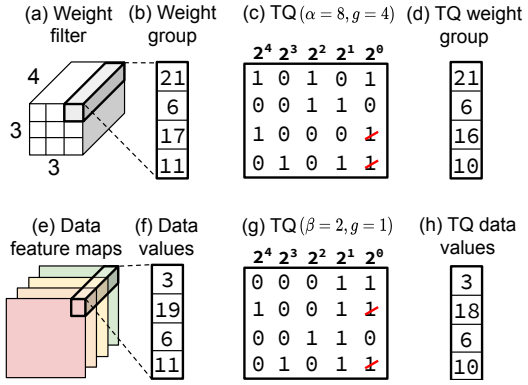


Figure 3: Term quantization (TQ) applied to a group of size $g = 4$ filter weights (top) and individual data values (bottom).

discuss the computational motivation for bounding the number of terms in weight and data values through TQ in Section 3.3.

3.1. Term Quantization on Weight Groups

Term quantization (TQ) is a new quantization technique, proposed in [27], which can be applied across a group of values (with group size g) by keeping the leading α terms across all values in the group. Figure 3(a)-(b) show how four weights in a convolutional weight filter can form a group to be processed by TQ. After applying TQ with a term budget of $\alpha = 8$ (Figure 3(c)), two of the 2^0 terms are dropped such that only 8 nonzero terms remain across the values in the group. Figure 3(d) shows the weight group values after applying TQ.

Compared with rigid quantization schemes, such as uniform quantization, where the terms can only be placed at certain digit location, TQ allows for a much greater flexibility in allocating terms across the group. By allocating more terms to larger values in the group and fewer terms to small values in the group, TQ achieves a much lower rounding error. This term allocation property is especially useful for DNN weights, as they are well approximated by a normal distribution. Figure 4(a) shows a histogram of the weights in the 13th convolution layer in ResNet-18 [16]. Applying a Maximum Likelihood Estimate (MLE) of a 1D normal distribution gives $N(0, 0.03)$. Other layers in ResNet-18 follow a similar distribution with σ between 0.01 and 0.04. Figure 4(b) shows the average quantization error due to TQ for samples drawn from a zero-mean normal distribution, with σ of 0.03, as a function of the TQ group size. The quantization error rapidly decreases as the group size goes from 1 to 4 and becomes flatter as the group size reaches 16. Based on this analysis, we use a group size of 16 for the rest of the paper, as it achieves most of the benefit from weight grouping.

3.2. Term Quantization for Data Values

TQ can also be applied to an individual value by keeping the leading β power-of-two terms in it. The bottom of Figure 3(e)-(h) show how TQ is applied to values in data feature maps of a Convolutional Neural Network (CNN). As shown in the figure,

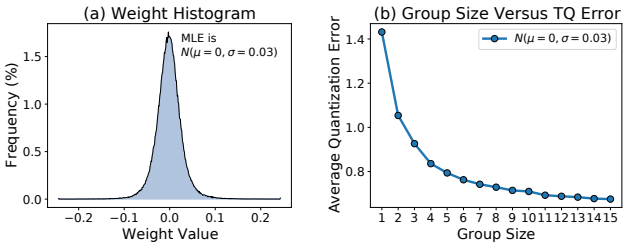


Figure 4: (a) The distribution of weights in the 13th convolutional layer in ResNet-18 [16]. (b) The quantization error using Term Quantization (TQ) with an average term budget of one term per value as the group size is varied from 1 to 15.

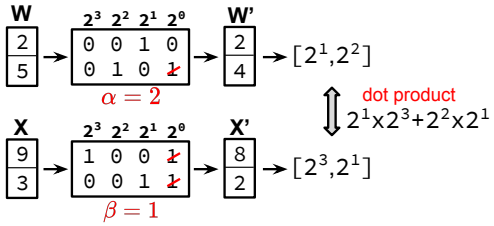


Figure 5: Term quantization of weight and data, followed by dot-production computation on quantized values.

for the value 19 (10011), TQ with a term budget $\beta = 2$ would quantize it to 18 (10010) by dropping the smallest 2^0 term (denoted by the red slash). Unlike UQ, which always truncates the low-order terms (*e.g.,* 2^0) by reducing the bit-width, TQ maintains a larger bit-width but reduces the number of 'active' terms in each value up to the term budget β. In Section 3.3, we discuss the computation implications of limiting the number of non-zero terms in data values.

3.3. Term-pair Multiplication Under Term Budget

Suppose we consider the dot-product computation between a group of weights and a group of data values. We use the follow terminologies and notations throughout the paper:

- *group size* (g)
- *term budget* (α) for a group of weights
- *data term budget* (β) for individual data items (*i.e.,* $g = 1$)
- *term-pair budget* (γ) is $\alpha \times \beta$

We note that the dot-product computation between term-quantized weights and data involves γ *term-pair multiplications*. Each term-pair multiplication amounts to an addition of the exponents in the two terms. An example is given in Figure 5. Assume TQ with a term budget $\alpha = 2$ and data term budget $\beta = 1$ is applied on a weight group $\mathbf{W} = [w_1, w_2] = [2, 5] = [2^1, 2^2 + 2^0]$ and data values $\mathbf{X} = [x_1, x_2] = [9, 3] = [2^3 + 2^0, 2^1 + 2^0]$, respectively. This produces the term-quantized weights $\mathbf{W'} = [2^1, 2^2]$ and the term-quantized data $\mathbf{X'} = [2^3, 2^1]$. The dot product between $\mathbf{W'}$ and $\mathbf{X'}$ is then computed by performing the following term-pair multiplications: $2^1 \times 2^3 + 2^2 \times 2^1 = 2^4 + 2^3 = 24$. This dot product requires 2 term multiplications, which equals γ. By limiting the number of leading terms in weights and data

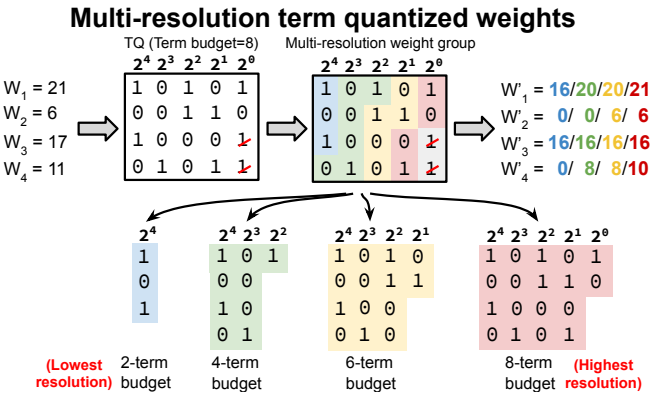


Figure 6: A multi-resolution weight group of size $g = 4$ with four different term budgets: 2 terms (blue), 4 terms (green), 6 terms (yellow), and 8 terms (red). Smaller term budgets share terms with all larger budget. Some smaller terms (*e.g.,* 2^0) are not used by any term budget and are effectively zero.

to a fixed group budget via TQ, we can bound the term-pair computations needed for each value-level multiplication in a convolutional layer, and further resulting in a adjustable implementation cost. We utilize this term-pair budget $\gamma = \alpha\beta$ in the design of our multi-resolution Multiplier-Accumulator (mMAC) discussed in Section 5.

4. Multi-resolution DNN

In this section, we illustrate how a single DNN can naturally support multiple resolutions (*i.e.,* term budgets) through the use of TQ. First, in Section 4.1, we show how the terms in a group of weights can be shared across multiple resolutions. Then, in Section 4.2, we present a teacher-student training algorithm to learn a meta multi-resolution DNN that supports multiple resolutions via term sharing.

4.1. Multi-resolution Weight Groups

In Section 3, we discussed how a group of weights can be quantized to meet a specific term budget α by dropping the low-order power-of-two terms in the group. Here, we extend this notion such that **a single group of weights can support multiple term budgets**. Figure 6 depicts a multi-resolution weight group for the same group of values in Figure 3, under four term budgets: 2, 4, 6, and 8 terms. When a lower-resolution term budget of $\alpha = 2$ is selected, the top two leading terms across the four values are kept (shown in the blue region), resulting in the set of term-quantized values of $[16, 0, 16, 0]$. In contrast, when a higher-resolution term budget (*e.g.,* $\alpha = 8$) is selected, the values in the group have less quantization error (*e.g.,* $[21, 6, 16, 10]$). This group-based multi-resolution approach can be applied to the weight across all layers of the DNN. Similarly, this multi-resolution paradigm can also apply to the data by dynamically selecting the data term budget β.

Under this multi-resolution paradigm, during inference runtime, the term budgets α, β can be selected to accommodate the current hardware resource constraint (*e.g.,* processing time

Algorithm 1: Meta Multi-resolution DNN Training

Input: $\mathbf{W}_l$ is the full-precision weights at layer l.
$\mathbf{X}_l^T$ is input data for the teacher sub-model at layer l.
$\mathbf{X}_l^S$ is input data for the student sub-model at layer l.
K and A are the sets of term budgets and data term budgets.
S is the set of term budget pairs, $S = \{(\alpha,\beta)|\alpha \in K, \beta \in A\}$.
b is the bit-width of the multi-resolution model.
I is the number of training iterations (*i.e.,* steps).
L is the total number of DNN layers.
g is the TQ weight group size (static across sub-models).

Output: The teacher model $\mathbf{W}_l^{tq,T}$.

for $i \leftarrow 1$ *to* I **do**
  for $l \leftarrow 1$ *to* L **do**
    Step 1 Apply b-bit uniform quantization on $\mathbf{W}^l$, $\mathbf{X}_l^S$ and $\mathbf{X}_l^T$ to produce the result quantized model $\mathbf{W}_l^q$ and quantized data $\mathbf{X}_l^{q,T}$, $\mathbf{X}_l^{q,S}$.
    Step 2 Let α_T and β_T denote the maximum possible term budgets for weight and data, respectively. Apply term quantization on SDR-encoded $\mathbf{W}_l^q$ with a budget α_T and a group size g, generating the result model $\mathbf{W}_l^{tq,T}$.
    Step 3 Apply term quantization on $\mathbf{X}_l^{q,T}$ with a budget β_T and a group size 1 to produce term-quantized data $\mathbf{X}_l^{tq,T}$.
    Step 4 Randomly select a pair of term budgets (α_S, β_S) from S, apply term quantization on SDR-encoded $\mathbf{W}_l^q$ under a budget α_S and a group size g to generate $\mathbf{W}_l^{tq,T}$.
    Step 5 Apply term quantization on $\mathbf{X}_l^{q,S}$ with a budget β_S to generate the term-quantized data $\mathbf{X}_l^{tq,S}$.
    Step 6 Perform the forward pass (*e.g.,* convolution) for $(\mathbf{W}_l^{tq,T}, \mathbf{X}_l^{tq,T})$ and $(\mathbf{W}_l^{tq,S}, \mathbf{X}_l^{tq,S})$ to produce $\mathbf{Y}_{out}^T$, $\mathbf{Y}_{out}^S$.
    Step 7 Perform additional operations (*e.g.,* non-linear activation, batch normalization) on $\mathbf{Y}_{out}^T$ and $\mathbf{Y}_{out}^S$ to produce $\mathbf{X}_{l+1}^T$ and $\mathbf{X}_{l+1}^S$.
  Step 8 Compute the loss L_T, L_S for teacher and student network.
  Step 9 Compute the gradient, update $\mathbf{W}_l$ and the corresponding clipping parameters for each layer $l \in L$.

or energy budget) at hand. For instance, when there is less processing demand, larger term budgets can be applied to the DNN weights, producing a DNN model with both a better performance (*e.g.,* classification accuracy) and higher computation cost, and vice versa. Additionally, since the terms in a weight group are shared across all resolutions, they only need to be stored a single time while still supporting all resolutions. Specifically, the terms for the lower-resolution weight group can be derived by applying TQ with a corresponding term budget. We call the resulting DNN model corresponding to a specific term budget pair (α,β) a *sub-model*.

4.2. Meta Multi-resolution Model Training

Training a meta multi-resolution DNN to support multiple term budgets (*i.e.,* sub-models) during inference requires special considerations. As we show later in Section 6.3, a simple post-training quantization approach [29] applied to a DNN trained without multi-resolution considerations does not achieve good performance for the low-resolution settings with smaller term budgets. Therefore, we propose to train the multi-resolution DNN such that the quantization error introduced by TQ during inference can be accounted for during training in a similar manner to other quantization-aware training approaches [21].

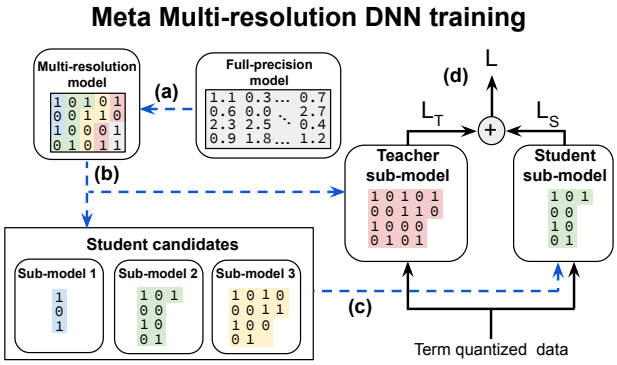


Figure 7: Meta multi-resolution DNN training procedure to support sub-models with different term budget at inference.

A straightforward training strategy is to jointly train all the possible sub-models by minimizing the sum of the losses derived from each sub-model. However, this formulation causes the training runtime and memory to grow super-linearly with the number of sub-models, making multi-resolution DNNs with more than a few settings (*e.g.,* 3 sub-models) impractical to train in this manner. To mitigate these training constraints, we propose a knowledge distillation mechanism which optimizes only two sub-models per iteration: a higher-resolution teacher sub-model and a lower-resolution student sub-model. To speed-up convergence of the multi-resolution model training, we use a pre-trained full-precision 32-bit floating-point model, which was trained on same training dataset.

Figure 7 provides an overview of this teacher-student training procedure. At each iteration, the full-precision model is first quantized to produce a multi-resolution model (step a). Then, the set of possible sub-models are generated by applying the corresponding term budgets on the multi-resolution model using TQ (step b). The sub-model with the largest term budget is always used as the teacher network. The student network is randomly selected from the remaining sub-models (step c). The training loss is computed using the knowledge distillation technique [17], which combines loss terms using both the real labels (L_T) and the soft labels generated by the teacher network (L_S) (step d). The resulting gradients will be applied to the the full-precision model.

As this training is run over many iterations (*e.g.,* 50000 steps), the student models will receive roughly the same number of updates via random selection. During each forward propagation, we apply SDR to reduce the weight and data terms in the multi-resolution models. Additionally, we adopt the techniques in [7] to learn the clipping parameters for both weight and data. See Algorithm 1 for a detailed summary of this training procedure.

5. System for Multi-resolution DNN Inference

In this section, we describe the design of the mMAC system (Figure 8) which achieves computation sharing between the sub-models. To allow simple system design, we choose a systolic array [25] for the implementation of the computation

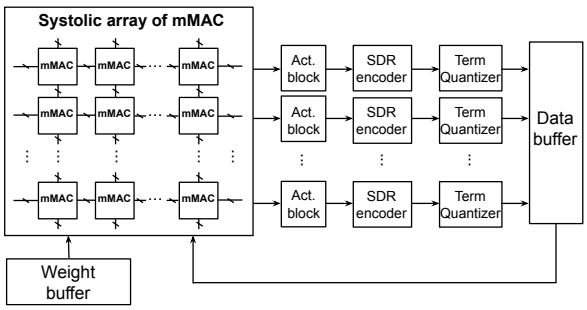


Figure 8: The mMAC system design.

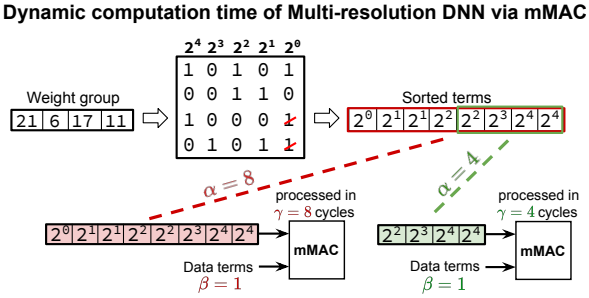


Figure 9: The mMAC can process multiple term budgets in a multi-resolution DNN with varying the processing time (*e.g.,* a 4-term budget in 4 cycles and a 8-term budget in 8 cycles).

engine, as it allows for a highly regular layout with low routing complexity. However, our multi-resolution paradigm also supports other computation engine designs.

5.1. Overview of Multi-resolution Model Deployment

Given a multi-resolution DNN model trained via Algorithm 1, the terms in each weight group are sorted and stored in memory. Before the execution, depending on the term budget α, the corresponding leading terms for each group are loaded from memory into each mMAC. During the execution, the data terms are first quantized under the data term budget β before entering the mMAC, which can process a fixed amount of term pairs per cycle. This leads to a processing latency which is directly proportional to the term-pair budget γ. An example is given in Figure 9, for a term budget of $\alpha = 8$ (shown in red), 8 weight terms are loaded from memory to mMAC, which leads to a processing time of $\gamma = 8$. In contrast, a lower term budget ($\alpha = 4$ shown in green) would require partial terms to be loaded, giving a lower processing time ($\gamma = 4$).

5.2. Multi-resolution MAC (mMAC) Design

The multi-resolution MAC (mMAC) performs term-pair multiplication via exponent addition. The hardware design of a mMAC for a group size $g = 4$ and term budgets $\alpha = 8, \beta = 2$ is shown in Figure 10. The exponents for term pairs are stored in weight and data exponent queues, with the sign of each term stored in a separate queue with one bit per term. For example, the term -2^3 would save the exponent 3 in the exponent queue and a minus ($-$) in the sign queue. We use the same

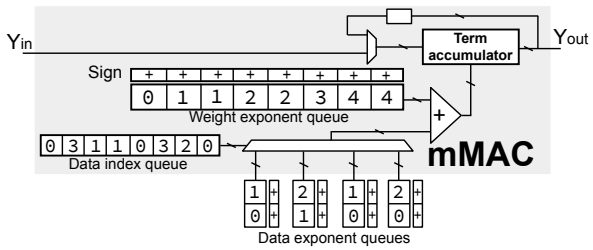


Figure 10: Multi-resolution MAC design.

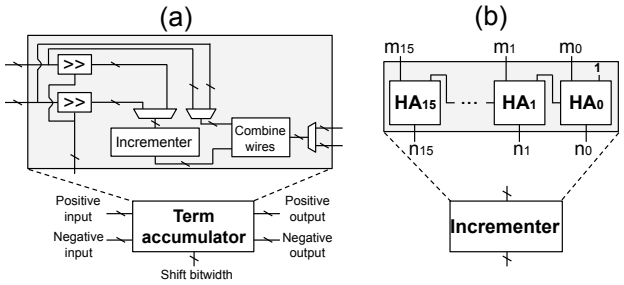


Figure 11: The term accumulator and incrementer.

weight terms as depicted in Figure 9 for illustration simplicity. Every cycle, a weight exponent is selected in order from the weight exponent queue, and a data exponent is selected by using indexes stored in the data index queue. Then, the adder computes the sum of the exponents, set the sign, and deliver the intermediate result to the term accumulator for accumulation operation within one cycle. Therefore, processing a group with 8 term pairs takes 8 cycles in total. The weight exponent queue and data index queue are implemented using linear feedback shift registers, so that the previous output is fed back into its input for the further usage.

Figure 12 depicts the operation of mMAC under a term budget $\alpha = 4$ across multiple cycles ($T = 0 \ldots 5$). Only the four leading terms and the corresponding data indexes are loaded in the weight exponent queue and data index queue. At each cycle, a pair of weight and data exponents are processed by the adder, with the resulting signed exponent sent to the term accumulator. The term accumulator converts the signed exponent to a value and then adds it with the input accumulation. The term accumulator output will loop back to be used again in the next cycle while there are remaining term pairs to be processed before being passed to the neighboring systolic cell.

A naive implementation of the term accumulator would transform the signed exponent into the corresponding binary value, and sum the intermediate result with the input accumulation using a parallel adder. Since the input accumulation can be large (*e.g.,* a 32-bit integer), a parallel adder would be expensive to implement. Instead, we propose an efficient hardware implementation for the term accumulator by leveraging the one-hot property of the binary expression for a power-of-two term, as shown in Figure 11(a).

For instance, we add 4 $(0100)_2$ to the accumulator with value 9 $(1001)_2$ by right shifting both by 2 bits, adding the resulting two numbers with the incrementer, and shifting back

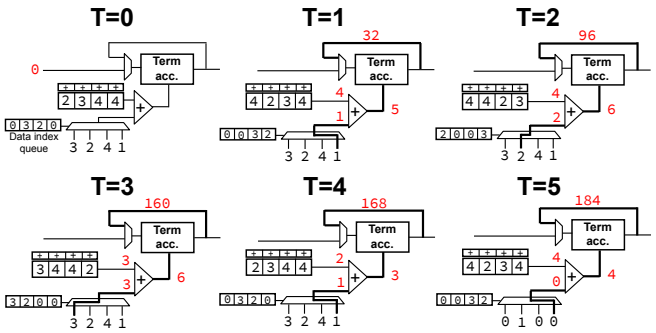


Figure 12: An example of mMAC operation.

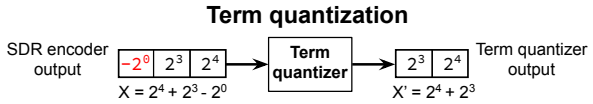


Figure 13: Term quantization with $\beta = 2$.

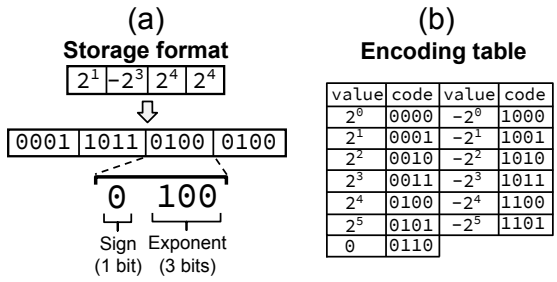
the shifted out 2 bits from the accumulator. Note, we can use an incrementer because 4 has only one nonzero bit in its representation because it is a term-pair multiplication product.

To perform the increment operation, instead of using a parallel adder, which consists of a chain of full adders, an incrementer can be implemented with an equivalent number of half adders (Figure 11(b)). This leads to a significant reduction in hardware resource consumption.

One problem of the above design is that, under SDR, a term can be negative, meaning that the design must support both increment and decrement operations. To mitigate this issue, we adopt two input accumulations to accumulate the positive terms and negative terms separately. The two input accumulations use the incrementer in a time-sharing fashion. A single parallel adder is used to perform subtraction between the two accumulations at the end of each row of systolic array to produce the final results.

5.3. Activation Block, SDR Encoder, and Term Quantizer

The activation block takes the outputs from the systolic array and applies the corresponding nonlinear activation function (*e.g.,* ReLU). To implement the ReLU function, the activation block first detects the sign of the input by checking its most significant bit (MSB). The activation block outputs a zero if the input is negative; otherwise it simply outputs the original input. The SDR encoder takes the outputs from the activation block and produces signed representations with the minimum number of terms, as described in Section 2.3. The SDR encoder can be implemented with a simple finite state machine (FSM). The term quantizer selects the top β terms for each data value. Figure 13 demonstrates this process for an input $x = 23 = 2^4 + 2^3 - 2^0$. One term of x is delivered to the term quantizer every cycle, which counts the total number of the sent terms and sets terms to 0 once the budget β has been reached (*e.g.,* 2 in this example). The outputs from the term quantizer will be saved in the data buffer.



value	code	value	code
2^0	0000	-2^0	1000
2^1	0001	-2^1	1001
2^2	0010	-2^2	1010
2^3	0011	-2^3	1011
2^4	0100	-2^4	1100
2^5	0101	-2^5	1101
0	0110		

Figure 14: (a) Terms are converted into a packed format for efficient storage. (b) The encoding table for storage.

5.4. Memory Subsystem

We have developed a compact format for the storage of weight and data terms in the memory. Figure 14(a) shows an example for the encoding of a group of 4 terms 2^4, 2^4, -2^3 and 2^1, where each term is encoded with 4 bits. The first three bits represent the exponent of the term, and the forth bit indicates the sign. Figure 14(b) shows encoding table for 5-bit TQ.

Under this storage scheme, each data value will be represented by 4β bits. Storing the terms and the data indexes for each weight group would require 4α bits and $\alpha log_2(g)$ bits, leading to an average of $\frac{4\alpha+\alpha\log_2(g)}{g}$ bits per weight value. For a group size of $g = 16$ and budgets of $\alpha = 16, \beta = 2$, a weight and data value both need 8 bits to be stored. Our memory subsystem consists of a weight and data buffer. The weight buffer holds the terms of the highest resolution in each weight group as well as the corresponding data indexes for selecting the data input. The data buffer holds the data terms for both the input and output data of the current layer with the format depicted in Figure 14(a).

6. Multi-resolution Performance Evaluation

In this section, we evaluate the performance (*e.g.,* classification accuracy, perplexity, or mean Average Precision) of DNNs trained under the multi-resolution paradigm proposed in Section 4 on a diverse range of applications including multiple CNNs (ResNet-18 [16], ResNet-50 [16] and MobileNet-V2 [38]) on ImageNet [11], an LSTM [18] on Wikitext-2 [34], and YOLO-v5 [23] on COCO [30]. We use pre-trained full-precision models as the initial models for the proposed multi-resolution paradigm discussed in Section 4.2. These models come from the PyTorch torchvision for ResNet-18, ResNet-50, and MobileNet-v2. We train a full-precision LSTM ourselves using the PyTorch language model example. For YOLO-v5, we use the pre-trained small model provided by the official repository (https://github.com/ultralytics/yolov5).

We use these pre-trained models to initialize the training procedure described in Algorithm 1. For all settings, we use a weight group size of $g = 16$. To perform the training efficiently, we have implemented a custom CUDA kernal to perform TQ and SDR encoding during the forward pass of training. Full details of the training hyper parameters can be found in the

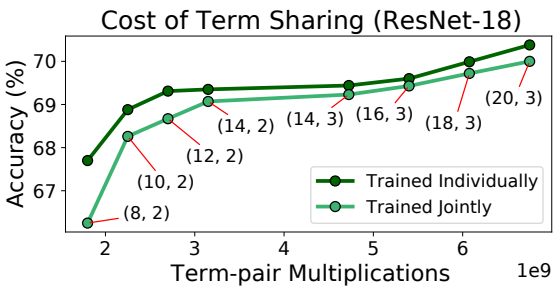


Figure 15: The multi-resolution model (light green) for ResNet-18 trained with 8 sub-model settings has slightly lower accuracy than settings trained individually (dark green). The (α, β) values for each sub-model are indicated by red lines and text.

appendix. This section answers the following questions on multi-resolution performance:

- (Section 6.1) How much performance is lost by enforcing term sharing instead of training each sub-model separately?
- (Section 6.2) How does the distribution of weight values change across sub-models as a function of the the weight term budget?
- (Section 6.3) How much performance is gained via the multi-resolution training approach (Algorithm 1) as opposed to a post-training term quantization approach as in [27]?
- (Section 6.4) How does UQ (with varying bit-widths) compare to TQ (with varying term budgets) under a bit or term sharing regime in terms of performance?

6.1. Impact of Term Sharing on Performance

In order to use the multi-resolution paradigm discussed in Section 4, it is required that the weight values must be shared across all sub-models. Therefore, it is important to investigate the impact in performance of enforcing this weight sharing across the different sub-models. Figure 15 shows the number of term-pair multiplications and classification accuracy for ResNet-18 models trained on ImageNet. The dark green points represent 8 models trained individually using different TQ settings, such as $(\alpha = 10, \beta = 2)$. By comparison, the light green points show the corresponding performance for the proposed multi-resolution model with 8 sub-models using the same TQ settings. Generally, we see that the multi-resolution model is 0.25% to 1.25% worse than each point trained individually, with the largest gap being for the most aggressive setting $(\alpha = 8, \beta = 2)$.

6.2. Multi-resolution Weight Distributions

As discussed in the previous section, the terms in a multi-resolution DNN are shared across all sub-models, meaning that the weight values change depending on the number of allocated terms for the sub-model. Figure 16 shows a histogram of the frequency of weight values for three different sub-models from a multi-resolution DNN and a 5-bit UQ

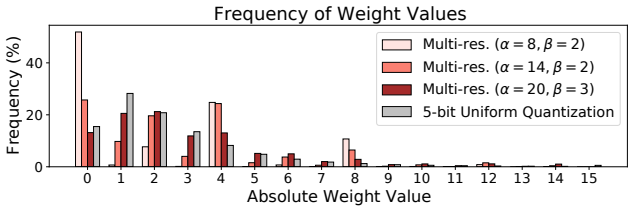


Figure 16: A histogram of the absolute weight values for 3 sub-models of ResNet-18 trained under the multi-resolution paradigm and an individual model trained under 5-bit UQ.

setting. Interestingly, for the low-resolution sub-model setting of $(\alpha = 8, \beta = 2)$, the weights are concentrated mostly at values that can be represented with a single power-of-two (*e.g.,* 2, 4, and 8) and almost 50% of values are 0. For the more high-resolution sub-model setting of $(\alpha = 20, \beta = 3)$, the distribution of values closely follows the 5-bit UQ model.

In this way, the proposed multi-resolution DNN can be viewed as interpolating between logarithmic quantization for the low-resolution sub-model and 5-bit uniform quantization for the high-resolution sub-model. By training many sub-models between these two extreme settings, the multi-resolution model is able to gradually trade-off computation (in the number of term operations per sample) for performance (*e.g.,* classification accuracy) as depicted earlier in Figure 15.

6.3. Comparison to Post-training Quantization

Instead of training a multi-resolution DNN using Algorithm 1, we could perform post-training quantization [29] on a pre-trained floating-point model. Generally, post-training quantization leads to poor performance for uniform quantization with less than 8 bits per weight. However, term quantization, which was originally posed as a post-training quantization approach in [27], leads to improved performance compared to UQ even when only a few terms are used per value.

Figure 17 provides a comparison between the proposed multi-resolution training with TQ approach and post-training TQ as in [27] for ResNet-18 and ResNet-50 on ImageNet. For both ResNet-18 and ResNet-50, we see that multi-resolution training outperforms post-training quantization for all settings. Additionally, we see that more aggressive settings lead to a larger degradation in accuracy, demonstrating that multi-resolution training is important to achieve a reasonable trade-off space between performance and number of operations.

6.4. Comparison to Term Sharing Uniform Quantization

In this section, we compare the number of term-pair multiplications required to process one sample at different sub-model settings across multiple domains (image classification, object detection, and language modeling) using either UQ or TQ under term sharing via a multi-resolution model. The training procedure for the UQ models is the same as in Algorithm 1, except with UQ substituted for TQ. For the UQ models, sub-models are obtained by varying the weight and

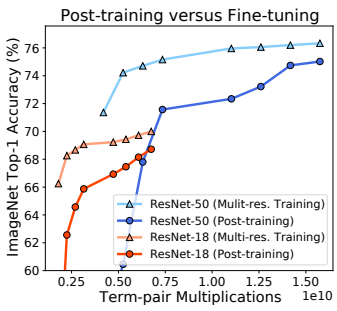


Figure 17: Comparing post-training quantization to multi-resolution training for ResNet-18 and ResNet-50 on ImageNet. Multi-resolution training significantly improves accuracy.

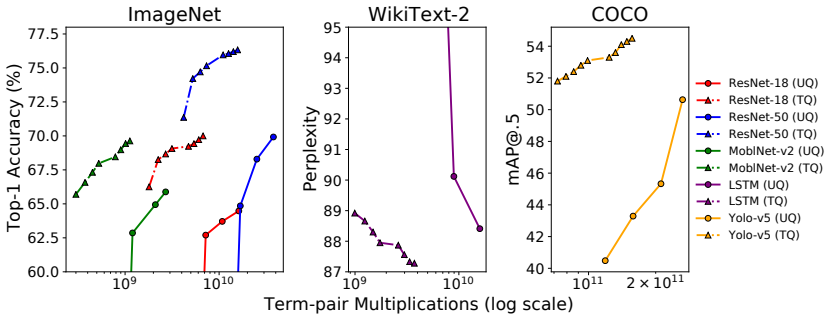


Figure 18: Comparing multi-resolution models trained under uniform quantization (UQ) and term quantization (TQ) for CNNs on ImageNet (left), an LSTM on Wikitext-2 (middle), and YOLO-v5 on COCO (right). The UQ sub-models vary the weight and data bit-width (*e.g.,* from 2-bit to 5-bit for ImageNet), while the TQ sub-models vary in α (number of terms per weight group) and β (number of terms per data value).

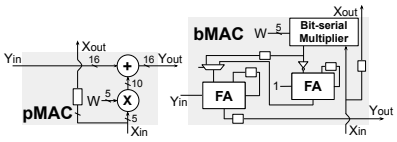


Figure 19: Bit-parallel MAC (left) and bit-serial MAC (right).

	LUT	FF
pMAC	57	44
bMAC	12	14
mMAC	21	25

Table 1: Resource consumption of FPGA

γ	16	20	24	28	42	48	54	60
bMAC	0.15×	0.17×	0.22×	0.26×	0.37×	0.44×	0.50×	0.56×
pMAC	0.17×	0.22×	0.27×	0.31×	0.47×	0.53×	0.61×	0.66×
mMAC	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×

Table 2: Comparison on energy efficiency for MAC designs.

data bit-width (*e.g.,* from 5-bit to 2-bit values for the CNNs trained on ImageNet). See the appendix for a complete description of parameters settings used in the evaluation.

6.4.1. ImageNet Figure 18(left) compares the performance of our multi-resolution approach under TQ and UQ across ResNet-18, ResNet-50, and MobileNet-v2 on ImageNet. For all three models, we observe that the multi-resolution approach using TQ greatly reduced the number of term multiplications compared to UQ while also achieving significantly better performance of roughly 5%. Enforcing term sharing across the UQ settings leads to a significant degradation in model performance, as all of the sub-models must share a common scale factor for quantization. Deriving a common scale factor is difficult for the 5-bit and 2-bit settings. Additionally, the trade-off between performance and operations is more graceful under TQ compared to UQ due to the more fine-grained nature of TQ as each point varies by two additional nonzero terms instead of a reduced bit-width.

6.4.2. LSTM Figure 18(middle) compares the performance of the two approaches on a 2-layer LSTM with 650 hidden units (*i.e.,* neurons), a word embedding of length 650, and a dropout rate of 0.5 trained on WikiText-2, following the PyTorch word language model example. We see that our multi-resolution approach with TQ outperforms UQ by a wide margin, with even the most aggressive sub-model setting still achieving a reasonable perplexity.

6.4.3. COCO Finally, Figure 18(right) provides a comparison on YOLO-v5 (small) trained on COCO. We find that object detection requires significantly more precision compared to image classification to achieve good performance. Due to this, the UQ settings span from 8-bit to 5-bit representations for weights and data. By comparison, the sub-models in our multi-resolution approach span from ($\alpha = 22, \beta = 4$) to ($\alpha = 38, \beta = 5$). Since TQ only specifies the number of nonzero terms and not the bit-width, we are able to achieve better performance by using a large bit-width (8-bit) for all settings while varying the term budget in each sub-model.

7. Hardware Evaluation

In this section, we evaluate the performance of the mMAC system described in Section 5. We have synthesized our mMAC system using Xilinx VC707 FPGA evaluation board. We first illustrate the advantage of mMAC design by comparing it against conventional bit-serial and bit-parallel MAC (Section 7.1). Then, we evaluate the hardware performance under different resolutions in Section 7.2. Finally, we compare our system against the other FPGA accelerators in Section 7.3.

7.1. Comparing mMAC to Conventional MACs

We evaluate the efficiency of our mMAC design by comparing it against bit-serial and bit-parallel implementations of a conventional MAC. We evaluate all three designs on the following computation: $y_{out} = \sum_{i=1}^{g} x_i w_i + y_{in}$, where y_{in}, y_{out}, x_i and w_i are 16-bit, 16-bit, 5-bit and 5-bit, respectively, and g is the number of accumulating operations (*i.e.,* group size in TQ).

The left part of Figure 19 shows the design of a bit-parallel MAC (pMAC), which performs multiplication between x_i and w_i and sums the result with y_{in} in one cycle and generates y_{out} in g cycles. The bit-serial MAC (bMAC), based on [26], is shown in Figure 19(right). It consists of a bit-serial multiplier and additional logic elements to negate the multiplier output and perform accumulation. It requires 16 cycles to process one

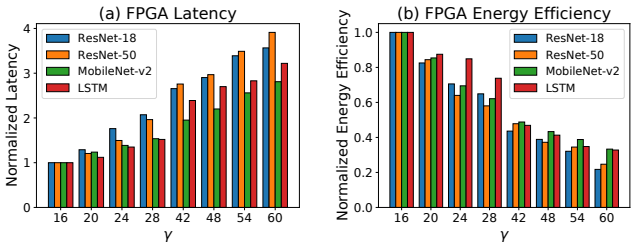


Figure 20: The mMAC system supports a range of designs with varying energy efficiency and latency under different γ settings across multiple networks (normalized to $\gamma = 16$).

pair of values, for a total of $16 \times g$ cycles to generate y_{out}. In contrast, mMAC takes γ cycles to perform this accumulation operation, where γ is the term-pair budget for the weight and data. We set the group size $g = 16$ for evaluation.

Table 1 depicts the FPGA resource consumption of the three MAC designs in terms of LookUp Tables (LUTs) and Flip-flops (FFs). Compared with pMAC, mMAC consumes $2.8\times$ less LUT and $1.8\times$ less FFs, this is due to the fact that mMAC performs additions on the exponents as opposed to multiplication for computing the term products. Although the bMAC achieves a even lower hardware source consumption than mMAC, it requires a much larger processing latency.

Table 2 shows the evaluation on energy efficiency for all the three MAC designs on FPGA, where all the results are normalized by the performance of mMAC. We evaluate mMAC under different term-pair budget γ used in Figure 15. We observe that the performance of mMAC improves as term-pair budget reduces, with all settings outperforming both bMAC and pMAC. A smaller term-pair budget allows for a smaller amount of term-pair operations required to generate the result. This reduces the processing time and further improves the energy efficiency of mMAC. In particular, compared with pMAC and bMAC, mMAC achieves a $3.1\times$ and $5.6\times$ higher energy efficiency on average for all the term-pair budget settings.

7.2. FPGA system evaluation

In this section, we evaluate our mMAC system performance under different resolutions. The mMAC system contains a 128×128 systolic array. We adopt the corresponding term-pair budgets used in Figure 18 for evaluation.

Figure 20 illustrates the trend in average processing latency (*i.e.,* number of cycles to finish one input sample) and energy efficiency (*i.e.,* number of input samples processed for one Joule of energy) of the mMAC system under different γ values (*i.e.,* term-pair budget) across multiple models. We notice that the processing latency decreases ($3.1\times$ on average) and the energy efficiency grows ($3.25\times$ on average), as γ reduces from 60 to 16. This is because a lower term-pair budget leads to a smaller mMAC processing time for a group of weight and data values. Additionally, a smaller term-pair budget also reduces on-chip traffic between the storage and computation engine, since only the terms in the low-resolution terms need to be transferred. Figure 20 demonstrates that our mMAC system can provide a range of designs with varying energy efficiency and latency by dynamically adjusting its computational cost based on the term-pair budget γ.

	[28]	[40]	[42]	[27]	Ours
FPGA Chip	VC709	Virtex-7	ZC706	VC707	VC707
Frequency (MHz)	150	100	200	170	150
FF	262k(30%)	348k(40%)	51k(12%)	316k(51%)	409k(66%)
LUT	273k(63%)	236k(55%)	86k(39%)	201k(65%)	275k(91%)
DSP	2144(59%)	3177(88%)	808(90%)	756(27%)	996(36%)
BRAM	1913(65%)	1436(49%)	303(56%)	606(59%)	524(51%)
Latency (ms)	2.56	11.7	5.84	7.21	3.98
Energy eff. (frames/J)	12.93	8.39	40.7	25.22	71.48

Table 3: Comparison of our FPGA implementation of ResNet-18 to other FPGA-based accelerators on ImageNet.

7.3. Comparison Against Other FPGA Designs

Lastly, we compare our mMAC system with the other FPGA DNN accelerator designs on ResNet-18. Specifically, we apply term budgets of $(\alpha, \beta) = (20, 3)$ and $g = 16$ for our mMAC system, which can achieve a top-1 prediction accuracy of 70.02% on ImageNet. The results are shown in Table 3. Our system achieves the highest energy efficiency. Although the processing latency of [28] is even lower, it has a much larger hardware resource cost and lower energy efficiency. On average, our system outperforms the other designs by $1.7\times$ and $3.28\times$ in terms of the processing latency and energy efficiency.

Our mMAC system achieves superior performance for several reasons. Most importantly, the multi-resolution DNN with TQ significantly reduces the number of term-pair operations, which further allows the computation engine to achieve a much tighter processing bound and therefore a lower processing latency. For instance, under a group size $g = 16$ and term-pair budget $\gamma = 60$, the mMAC system achieves a worst case processing time of only 60 cycles to compute the dot product for the 16 values in the group. This is in contrast to the conventional accelerator design, where the computation latency is always impeded by the slowest computation unit in the system. Second, the efficient design of mMAC converts the expensive multiplication operations between values into a series of additions between the term exponents, and the incrementer in the term accumulator further mitigates the implementation cost of the expensive parallel adder. Finally, the compact memory encoding scheme leads to a lighter traffic between the on-chip buffer and the computation engine.

8. Conclusion

We have shown that via term quantization, a single meta model can spawn sub-models of varying resolutions with low system overheads and performance loss. To this end, we train the meta model by jointly optimizing multiple sub-models of different resolutions. During inference, we implement multiple resolutions by simply adjusting the number of leading non-zero terms on the learned weights of the meta model. To minimize memory footprint of the metal model and stream-

line its training, we share terms across multiple sub-models. These approaches together lead to a multi-resolution MAC (mMAC) design that can efficiently implement multiple resolutions.Results of this paper demonstrate that field-configurable multi-resolution inference is viable.

References

[1] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 382–394. ACM, 2017.

[2] Algirdas Avizienis. Signed-digit numbe representations for fast parallel arithmetic. *IRE Transactions on electronic computers*, (3):389–400, 1961.

[3] Andrew D Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[4] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[5] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 367–379. IEEE Press, 2016.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[7] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks, 2018.

[8] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[10] Alberto Delmas, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, and Andreas Moshovos. Bit-tactical: Exploiting ineffectual computations in convolutional neural networks: Which, why, and how. *24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[12] Tim Dettmers. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561*, 2015.

[13] Barry L Drake, Richard P Bocker, Mark E Lasher, Richard H Patterson, and William J Miceli. Photonic computing using the modified signed-digit number representation. *Optical Engineering*, 25(1):250138, 1986.

[14] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[15] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

[20] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[21] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[22] Jonathan Jedwab and Chris J Mitchell. Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters*, 25(17):1171–1172, 1989.

[23] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, Adam Hogan, lorenzomammana, tkianai, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Hatovix, Jake Poznanski, Lijun Yu, changyu98, Prashant Rai, Russ Ferriday, Trevor Sullivan, Wang Xinyu, YuriRibeiro, Eduard Reñé Claramunt, hopesala, pritul dave, and yzchen. ultralytics/yolov5: v3.0, August 2020.

[24] Supriya Kapur, Asit Mishra, and Debbie Marr. Low precision rnns: Quantizing rnns without losing accuracy. *arXiv preprint arXiv:1710.07706*, 2017.

[25] H. T. Kung. Why systolic architectures? *IEEE Computer*, 15:37–46, 1982.

[26] H. T. Kung, Bradley McDanel, and Sai Qian Zhang. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. *24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[27] H. T. Kung, Bradley McDanel, and Sai Qian Zhang. Term revealing: Furthering quantization at run time on quantized dnns. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[28] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A high performance fpga-based accelerator for large-scale convolutional neural networks. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pages 1–9. IEEE, 2016.

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

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[31] Bradley McDanel, Surat Teerapittayanon, and HT Kung. Incomplete dot products for dynamic computation scaling in neural network inference. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 186–193. IEEE, 2017.

[32] Bradley McDanel, Sai Qian Zhang, H. T. Kung, and Xin Dong. Full-stack optimization for accelerating cnns using powers-of-two weights with fpga validation. *International Conference on Supercomputing*, 2019.

[33] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. *arXiv preprint arXiv:1703.06217*, 2017.

[34] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[35] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[36] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 27–40. ACM, 2017.

[37] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5456–5464, 2017.

[38] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[39] Sayeh Sharify, Alberto Delmas Lascorz, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Dylan Malone Stuart, Zissis Poulos, and Andreas Moshovos. Laconic deep learning inference acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 304–317. ACM, 2019.

[40] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing cnn accelerator efficiency through resource partitioning. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 535–547. IEEE, 2017.

[41] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[42] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga. *arXiv preprint arXiv:1808.04311*, 2018.

[43] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[44] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018.

[45] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[46] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.