VS-QUANT: PER-VECTOR SCALED QUANTIZATION FOR ACCURATE LOW-PRECISION NEURAL NETWORK INFERENCE

Steve Dai [1] **Rangharajan Venkatesan** [1] **Haoxing Ren** [1] **Brian Zimmer** [1] **William J. Dally** [1] **Brucek Khailany** [1]

ABSTRACT

Quantization enables efficient acceleration of deep neural networks by reducing model memory footprint and exploiting low-cost integer math hardware units. Quantization maps floating-point weights and activations in a trained model to low-bitwidth integer values using scale factors. Excessive quantization, reducing precision too aggressively, results in accuracy degradation. When scale factors are shared at a coarse granularity across many dimensions of each tensor, effective precision of individual elements within the tensor are limited. To reduce quantization-related accuracy loss, we propose using a separate scale factor for each small vector of (≈16-64) elements within a single dimension of a tensor. To achieve an efficient hardware implementation, the per-vector scale factors can be implemented with low-bitwidth integers when calibrated using a two-level quantization scheme. We find that per-vector scaling consistently achieves better inference accuracy at low precision compared to conventional scaling techniques for popular neural networks without requiring retraining. We also modify a deep learning accelerator hardware design to study the area and energy overheads of per-vector scaling support. Our evaluation demonstrates that per-vector scaled quantization with 4-bit weights and activations achieves 69% energy saving and 36% area saving over an 8-bit baseline while maintaining over 75% accuracy for ResNet50 on ImageNet. 4-bit weights and 8-bit activations achieve near-full-precision accuracy for both BERT-base and BERT-large on SQuAD while reducing area by 28% compared to an 8-bit baseline.

1 INTRODUCTION

Deep neural networks (DNNs) continue to achieve groundbreaking accuracy on a range of tasks, including image classification, object detection, machine translation, and natural language processing (NLP) (LeCun et al., 2015). In parallel, hardware designers have been racing to achieve the best performance per watt for running DNN inference on devices ranging from the edge to the datacenter (Sze et al., 2020). While most DNN models are trained in single-precision floating-point, they can be deployed for inference in lower-precision formats such as half-precision floating-point, fixed-point, and low-bitwidth integer depending on the target device and application specifications. Quantizing DNN models to lower precisions allows us to accelerate compute-bound operations such as convolution on high-throughput low-cost math units, conserve memory bandwidth for memory-bound computations, and reduce storage requirements in on-chip buffers and caches. For example, NVIDIA's Ampere Graphics Processing Unit (GPU) architecture supports INT8 and INT4 data types for these purposes (NVIDIA Corporation, 2020).

One way to quantize a DNN model is through quantization-aware training (QAT). QAT either trains the model from scratch or finetunes the trained full-precision model, with quantization operations in the model. Alternatively, post-training quantization (PTQ) directly quantizes the values of the full-precision model before and during inference without any retraining (Wu et al., 2020). Often, PTQ is more desirable because it does not require access to the complete set of possibly confidential training data, eliminates lengthy fine-tuning, requires little hyperparameter tuning, and provides a turnkey solution for quantizing any DNN. However, PTQ usually results in more accuracy loss than QAT because of the lack of training with quantizers in the loop. For both QAT and PTQ, accuracy loss from quantization varies by precision, model, and quantization algorithm.

Quantization scales high-precision values of a particular range to lower-precision values of a different range. A high-precision number (x) is mapped to a lower-precision number (x_q) with $x_q = Q(x/s, N)$ where s is the scale factor and $Q(a, b)$ is the function that quantizes a to a b-bit integer. Scale factors play an important role in determining the quantization error, which affects the ultimate accuracy of the quantized model. To avoid overloading the quantized model with too many scale factors and nullifying the compute and memory benefits of quantization, scale factors

[1]NVIDIA. Correspondence to: Steve Dai <sdai@nvidia.com>.

Proceedings of the 4^{th} MLSys Conference, San Jose, CA, USA, 2021. Copyright 2021 by the author(s).

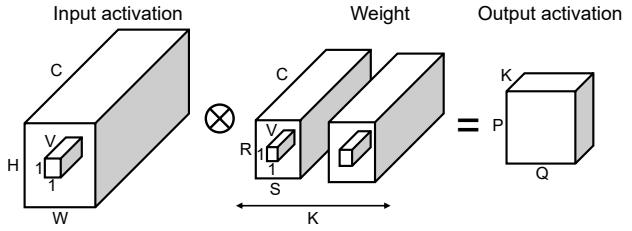

Figure 1. **Convolution** — Comparison between per-layer/per-output-channel scaling and per-vector scaling.

must be shared among multiple tensor elements. Typically, scale factors are shared at a coarse granularity by elements of an entire tensor or a large sub-tensor. For example, a typical quantized convolution layer as shown in Figure 1 employs a single scale factor for the entire input activation tensor ($C \times H \times W$) and each kernel ($C \times R \times S$) of the weight tensor. While coarse-grained scaling amortizes the cost of scaling across many elements, it likely requires mapping a broader range of values to the specified low-precision representation. The resulting increase in quantization error introduces significant accuracy loss for low-precision representations. The problem is exacerbated for DNNs whose input and/or weight values span a wide dynamic range.

We propose fine-grained per-vector scaled quantization (*VS-Quant*) to mitigate quantization-related accuracy loss. In contrast to coarse-grained per-layer/per-output-channel scaling, *VS-Quant* employs a scale factor for each vector of elements ($V \times 1 \times 1$) in the activation and/or weight tensor as shown in Figure 1. The range that must be represented within each vector is smaller than the range that must be represented across the entire layer, so many vectors can be represented at much higher precision and quantization error is only encountered in a small subset of vectors that contain wide ranges of values. Moreover, the unit of a vector matches the unit of vector multiply-and-accumulate (MAC) hardware ubiquitous in DNN accelerators (Sijstermans; Venkatesan et al., 2019; NVIDIA Corporation, 2020). This hardware-software synergy leads to an elegant extension of current accelerator architectures for implementing per-vector scaling with low overhead. The major contributions of our work are as follows:

- We propose *VS-Quant*, a novel per-vector scaled quantization technique to mitigate accuracy loss typical in existing quantized DNN models.
- We propose a two-level scaling scheme and algorithm that combine a set of fine-grained scale factors with each coarse-grained scale factor to enable efficient *VS-Quant* hardware implementations.
- We evaluate *VS-Quant* on popular DNN models and demonstrate significantly higher PTQ accuracy than conventionally scaled quantization on computer vision and NLP tasks.
- We extend the vector MAC unit of a DNN accelerator to support *VS-Quant* in hardware and analyze the area and power impact.
- We explore tradeoffs between accuracy and hardware efficiency across a range of hardware implementations and DNN models to identify Pareto-optimal designs for low-precision inference with PTQ.

The remainder of the paper is organized as follows: Section 2 reviews related work; Section 3 describes the fundamentals for quantization; Section 4 presents and evaluates our per-vector scaling technique and associated two-level scaling scheme; Section 5 describes the hardware implementation; Section 6 explores the accuracy and hardware efficiency tradeoff; Section 7 discusses quantization-aware retraining in the context of per-vector scaling, followed by conclusions in Section 8.

2 Related Work

Krishnamoorthi evaluates per-channel scaled quantization at various precisions for a set of convolutional neural networks (CNNs) (Krishnamoorthi, 2018). The paper finds that although PTQ can achieve good accuracy at 8 bits for these networks, QAT is required for getting good accuracy at lower precisions or for matching floating-point accuracy at 8-bits. McKinstry et al. shows that CNNs require only a small number of epochs of finetuning after carefully setting the learning rate schedule and fixing the quantization range (McKinstry et al., 2018). Instead of fixing the quantization range before QAT, PACT proposes to learn the range of weights and activations as part of training (Choi et al., 2018). Both papers achieve full-precision accuracy with only 4-bit precision. Other research has explored very low precision ternary (Zhu et al., 2016) and binary (Courbariaux et al., 2015; Hubara et al., 2016) weights and activations. These models required significant retraining to recover accuracy loss and do not reach full-precision accuracy for more difficult tasks. In addition to CNNs, recent work has proposed quantized transformer models for NLP (Zafrir et al., 2019; Shen et al., 2020) and for machine translation (Bhandare et al., 2019; Prato et al., 2019; Wu et al., 2016b). Wu et al. establishes a single 8-bit quantization workflow for maintaining less than 1% accuracy drop for many different types of networks (Wu et al., 2020).

Prior work has proposed schemes for uniform quantization (Courbariaux et al., 2014; Zhou et al., 2016) and non-uniform quantization (Han et al., 2015; Zhu et al., 2016). Uniform quantization uses integer or fixed-point format which can be accelerated with specialized math pipelines and is the focus of this paper. Non-uniform quantization leverages codebook look-ups to enable model compression and memory bandwidth reduction. To reduce quantization

error, vector quantization (Gray, 1984) based techniques take advantage of redundancy within a subspace of a weight or activation tensor. In particular, product quantization splits each subspace into vectors and optimizes a codebook for the vectors in each subspace (Wu et al., 2016a; Gong et al., 2014). Stock et al. extends this technique to preserve the reconstruction quality of actual network outputs instead of just weights (Stock et al., 2020). On a separate note, knowledge distillation can also improve the accuracy of quantized model (Mishra & Marr, 2017). By training the quantized model to mimic a high-precision model in a student-teacher setting, the paper obtains higher accuracy for a ternary ResNet-variant architecture.

Since the full set of training data may not be available at inference time, there is increasing interest in PTQ techniques that directly quantize full-precision values before and during inference (Krishnamoorthi, 2018; Lee et al., 2018; Nagel et al., 2019). More recently, Zhao et al. propose the outlier channel splitting technique to exactly represent outliers (Zhao et al., 2019). By duplicating channels that contain outliers and halving the values of those channels, this technique effectively shrinks the quantization range without modifying the network. Also focusing on the distribution of tensor values, Fang et al. propose a piecewise linear quantization scheme that optimally splits the quantization range into non-overlapping but differently-sized regions (Fang et al., 2020). With this, more precision can be assigned to the range where a majority of values lie. The paper also leverages a per-group quantization trick on each output channel filter to further reduce quantization error. To model long-tail effects in data distribution, Biscaled-DNN uses two scale factors for quantization (Jain et al., 2019). One scale factor is dedicated for increasing precision, and the other for increasing range. ZeroQ sidetracks the need for a training dataset by engineering a synthetic one that matches the statistics of the batch normalization operation of each layer of the network (Cai et al., 2020). This technique is considered another form of knowledge distillation.

Besides integer quantization, previous work proposes low-cost fixed-point and floating-point inspired data types for energy efficiency. For example, Moons et al. propose adaptive fixed-point quantization that trains a network for arbitrary fixed-point precision while minimizing energy (Moons et al., 2017). Flexpoint replaces 32-bit floating-point values with a block floating-point format that leverages shared exponents that can be dynamically adjusted to minimize overflow and maximize dynamic range (Köster et al., 2017). To avoid data loss from exponent sharing while improving energy efficiency, AdaptivFloat leverages a floating-point exponent bias based on absolute maximum value of the tensor to optimize the clipping of the tensor's dynamic range (Tambe et al., 2020). Rouhani et al. explore the accuracy-cost tradeoffs of different variants of a block floating-point format in production-level cloud-scale inference (Rouhani et al., 2020). Other work performs mixed-precision quantization on a per-layer basis to adapt to each layer's sensitivity to precision (Wu et al., 2018; Khoram & Li, 2018).

3 Quantization Fundamentals

Integer quantization maps high-precision floating-point weights and activations in a DNN to low-precision integer representations, typically with 8 or fewer bits. For simplicity, in this paper we refer to the floating-point weights and activations collectively as real values, and the quantized low-precision weights and activations collectively as integer values. We also focus on uniform integer quantization where the values are evenly distributed within the range of the integer format. While non-uniform quantization such as logarithmic quantization (Miyashita et al., 2016) is also possible, the techniques proposed in this paper are orthogonal and can be applied to either form of quantization.

There are several considerations when deciding how to quantize real values into integers. First, we must choose a range of real values to be represented so that any value out-of-range will be clipped. We may not necessarily want to choose the full range of presented real values, but rather clip outliers to improve the precision of quantized values within the range where most values reside. Second, we need to select the number of bits available for our integer values. With more integer bits, more discrete levels (integer values) are available to represent the same range of real values, resulting in smaller quantization error.

An N-bit signed two's complement integer quantization maps real values $x \in [x_{min}, x_{max}]$ to values $x_q \in [-2^{N-1}, 2^{N-1}-1]$. In general, a positive real scale factor s is used to scale the value from the real range to the integer range, and a zero point z represents the integer value corresponding to a real zero value. Since the zero point complicates integer computation pipelines, efficient DNN accelerators typically apply symmetric scale-only quantization assuming $z = 0$, $x_{min} = -x_{max}$, and $x_q \in [-2^{N-1}+1, 2^{N-1}-1]$ (Wu, 2019). If α denotes the absolute maximum real value that needs to be represented,

$$s = \frac{\alpha}{2^{N-1}-1} \tag{1}$$

$$x_q = clip\left(\left\lfloor \frac{x}{s} \right\rceil, -2^{N-1}+1, 2^{N-1}-1\right) \tag{2}$$

where $\lfloor \frac{x}{s} \rceil$ denotes rounding the scaled value to the nearest integer. If x is unsigned, $x_{min} = 0$ and x_q will be determined based on the integer range of $[0, 2^N - 1]$. To avoid issues with vanishing gradient, quantized integer values x_q are avoided during training. Instead, simulated quantization using discrete real values is applied to simulate the effect of integer quantization (Krishnamoorthi, 2018). Equation 3

Model	Task	Accuracy	Metric	Dataset
ResNet50 v1.5	Image classification	76.16	Top1	ImageNet 2012
BERT-base	Language model	86.88	F1	SQuAD v1.1
BERT-large	Language model	90.93	F1	SQuAD v1.1

Table 1. **Overview of DNN models in this study**

Model	Bitwidths	Max	Entropy	99.9%	99.99%	99.999%	99.9999%	MSE
ResNet50	Wt=3 Act=3U	0.18	6.61	**7.11**	4.60	1.54	0.45	0.23
	Wt=4 Act=4U	60.31	**70.76**	70.46	70.55	68.71	66.47	55.27
	Wt=6 Act=6U	75.43	75.79	75.24	**75.80**	75.73	75.66	75.34
	Wt=8 Act=8U	**76.16**	76.13	75.37	76.07	76.11	76.07	76.06
BERT-base	Wt=4 Act=4	1.48	**9.92**	5.23	6.13	2.41	1.48	7.77
	Wt=6 Act=6	9.33	15.63	61.32	**71.87**	69.89	14.22	69.79
	Wt=8 Act=8	76.88	73.17	69.78	84.10	84.65	79.67	**85.67**
BERT-large	Wt=4 Act=4	1.92	**6.96**	6.29	4.64	4.02	1.92	5.10
	Wt=6 Act=6	4.17	7.74	13.51	81.42	73.93	12.46	**84.17**
	Wt=8 Act=8	88.81	37.26	26.82	90.49	90.67	89.44	**90.81**

Table 2. **DNN accuracy with per-channel scaling and static calibration:** Weight and activation bitwidths are specified under `Bitwidths`. `U` indicates unsigned values. Values are otherwise signed. Max, Entropy, and MSE denote calibration using maximum absolute value, KL-divergence, and mean squared error, respectively. Percentages indicate the use of percentile calibration.

defines the simulated-quantized value x_q^s as a real value from rescaling the integer value by the original scale factor.

$$x_q^s = s \cdot x_q \tag{3}$$

Typically in a convolutional layer, a scale factor for weight or activation is determined for every layer of the network. Known as per-layer scaling, a single scale factor is used for each weight tensor (i.e., $K \times C \times R \times S$), and another scale factor is used for each activation tensor (i.e., $C \times H \times W$). To improve accuracy, multiple scale factors are determined for the weights of each layer. Known as per-channel scaling, a different scale factor is used for each output channel of a layer (i.e., $C \times R \times S$). We collectively refer to per-layer and per-channel scaling as coarse-grained scaling.

Scale factors must be chosen carefully to best approximate a real distribution with a limited number of discrete values. A calibration process is used to select the α used in Equation 1 for quantizing weights and activations. While α can be set to the maximum absolute value of the distribution (called max calibration), it is often beneficial to omit outlier values in favor of additional precision for inlier values. For example, percentile calibration sets α to a specific fraction of $|x_{max}|$. Entropy calibration, on the other hand, determines the α that minimizes the information loss between real and quantized distributions. For weights, scale factors are determined using static calibration prior to inference. For activations, scale factors can be determined using static calibration prior to inference or through dynamic calibration during inference. Note that static calibration for activations requires representative data samples that model the distribution of likely encountered inputs (Wu et al., 2018).

While per-channel scaling achieves better accuracy than per-layer scaling, coarse-grained scaling methods generally lead to significant accuracy degradation for a range of quantized models. With PTQ but without QAT, we observe accuracy degradation in popular image recognition and language models (listed in Table 1) after quantization, as indicated in Table 2. Even for models where coarse-grained scaling can be competitive, careful calibration of the scale factor with the right calibration technique is required for good accuracy. As shown in Table 2, the quality of calibration varies among different versions of the same network and across different networks. We first focus on enabling state-of-the-art inference accuracy with PTQ before discussing *VS-Quant* for QAT. Note that only convolution and linear layers have been quantized. Other layers remain in floating-point.

4 Per-vector Scaled Quantization

We propose *VS-Quant*, per-vector scaled quantization, to mitigate the accuracy loss from quantization. Rather than computing a single scale factor over multiple dimensions of a tensor, *VS-Quant* applies a scale factor for each vector of elements within a single dimension of a tensor. For a convolutional layer shown in Figure 1, per-vector scaling subdivides the input channel (C) dimension of the weight or activation tensor into $\lceil C/V \rceil$ vectors each with V elements. The number of vectors contained within a tensor depends on its shape and the designated vector size V.

In Table 3, we show that *VS-Quant* with static max calibration for weights and dynamic max calibration for activations has the potential to achieve significantly better accuracy with low bitwidths. Compared to the floating-point baseline, per-vector scaled quantization achieves negligible accuracy drop at 6 bits and less than 1% drop at 4 bits for ResNet50. In comparison, per-channel scaled quantization requires at

Model	Bitwidths	Per-vector	Best Per-channel
ResNet50	Wt=3 Act=3U	69.78	7.97
	Wt=4 Act=4U	75.28	70.76
	Wt=6 Act=6U	76.00	75.80
	Wt=8 Act=8U	76.15	76.16
BERT-base	Wt=3 Act=8	82.93	12.11
	Wt=4 Act=8	86.26	78.86
	Wt=6 Act=8	86.63	85.47
	Wt=8 Act=8	86.54	85.67
BERT-large	Wt=3 Act=8	89.54	11.59
	Wt=4 Act=8	90.66	86.75
	Wt=6 Act=8	90.76	90.58
	Wt=8 Act=8	90.88	90.81

Table 3. **PTQ accuracy of different DNN models with floating-point per-vector scale factors –** `Best Per-Channel` indicates the best calibrated per-channel scaled quantized accuracy among all calibration methods in Table 2.

least 6-bit weights for less than 1% drop. Both BERT-base and BERT-large achieve close to full-precision accuracy with 4-bit weights, compared to per-channel scaled quantization which has difficulty reach the same level even with 8 bits on BERT-base. Note that results are reported for PTQ where retraining is not required.

4.1 Vector Size

The quality of per-vector scaling depends on the vector size parameter. At one extreme with $V = 1$, each element would be individually quantized with its own scale factor and thus experience no loss in precision. At the other extreme with $V = C$, elements in each (R, S) in weight and (H, W) in activation would share the same scale factor. Table 4 compares the accuracy of a 6-bit quantized ResNet50 with per-vector scaling for different vector sizes. Accuracy decreases with increasing vector size because larger vectors have a higher probability of needing to represent a wider range of values. The goal is to carefully select V to minimize the required number of scale factors (maximize vector size) while maximizing the precision of the vector-scaled approximation and resulting network accuracy.

V=1	V=2	V=4	V=8	V=16	V=32	V=64
76.13	76.08	76.05	76.05	76.00	75.96	75.96

Table 4. **Accuracy of 6-bit ResNet50 on ImageNet with *VS-Quant* for different vector sizes**

4.2 Vector MAC

In addition to better precision, the vector granularity also maps naturally to the vector unit of compute in typical DNN accelerators. Because convolution and linear layers can be conveniently expressed as a collection of dot-products between an unrolled region of weights and an unrolled region of activations, vector-MAC units are the ubiquitous building blocks of many DNN processing architectures. Equation 4 shows the dot-product $y(j)$ between the jth vector region of weights $w(j)(i),\ i \in [0, V-1]$ and the jth vector region of activations $a(j)(i),\ i \in [0, V-1]$.

$$y(j) = \sum_{i=0}^{V-1} (w(j)(i) \cdot a(j)(i)) \tag{4}$$

With *VS-Quant*, we compute a scale factor $s_w(j)$ for the jth weight vector and a scale factor $s_a(j)$ for the jth activation vector to scale the quantized integer weights $w_q(j)(i),\ i \in [0, V-1]$ and integer activations $a_q(j)(i),\ i \in [0, V-1]$. Therefore, the dot-product in Equation 4 becomes the scaled dot-product in Equation 5.

$$y_q^s(j) = \left(\sum_{i=0}^{V-1} (w_q(i) a_q(i)) \right) s_w(j) s_a(j) \tag{5}$$

Note that the scale factors are factored out of each vector MAC, leading to a simple *VS-Quant* hardware implementation, as discussed in Section 5.

4.3 Calibration

While it is orthogonal to per-vector scaling, calibration is still needed to determine the range of real values to be represented, which is parameterized by α. As with conventional scaling techniques, weight scale factors $s_w(j)$ can be determined statically based on the trained model. Activation scale factors $s_a(j)$ can be computed statically with representative input samples or dynamically during inference. Likewise, calibration methods including maximum absolute value, percentile, and entropy can still be applied. However, because each vector only has a small number of elements, the distribution of a vector may lack enough samples to support more sophisticated calibration methods like percentile and entropy to determine a statistically useful α.

4.4 Two-Level Quantization

The results in Table 3 rely on floating-point scale factors per vector, which would lead to an inefficient hardware implementation. To improve area and energy efficiency, we introduce a two-level scaling scheme that further applies integer quantization on the per-vector scale factors. With this scheme, the per-vector scale factor s in Equation 3 is factored into the product of an integer per-vector scale factor s_q and a floating-point coarse-grained scale factor γ, as shown in Equation 6.

$$x_{q2}^s = s_q \cdot \gamma \cdot x_q \tag{6}$$

Here x_{q2}^s denotes the simulated-quantized value with two levels of scale factors. With an integer scale factor per-vector, we need to store only a low-bitwidth integer alongside each vector of tensor elements, and we can complete

all vector-wise computations with integer arithmetic. With the two-level scaling technique, we push the more expensive floating-point scale factors to the coarser granularity by introducing the less expensive integer scale factors at the finer granularity to achieve a balance between accuracy and hardware efficiency. Given N-bit integer weights or activations and M-bit integer per-vector scale factors, adding the M-bit scale factor alongside each V-element vector leads to a memory overhead of $M/(VN)$. To give a perspective with $N = M = 4$ and $V = 16$, the storage overhead is 6.25% which equates to an effective bitwidth of 4.25 bits. Compared to coarse-grained scaling, two-level per-vector scaling requires scaling the dot-product by the product of the integer scale factors, which represents an extra $(2N + log(V)) \times 2M$ multiplication for each vector dot-product.

Equations 7a-7j detail the algorithm for determining the scale factors when quantizing a real valued tensor x to an N-bit signed integer in the two-level quantization scheme. Index i indicates each vector; index j represents each element of a vector; and k is the index along the coarse-grained dimension with different coarse-grained scale factors. Assuming per-channel scale factors for the weight tensor of a convolutional layer, $k \in [0, K-1]$ while $i \in [0, \lceil C/V \rceil - 1]$ and $j \in [0, V-1]$.

The algorithm first computes floating-point scale factors at a per-vector granularity. Then it quantizes the per-vector scale factors by separating them into integer per-vector components and a floating-point per-channel component. We specify the datatype of each tensor in Equation 7 as fp for floating-point and int for integer.

$$x_{max}(k,i)_{fp} = \max_j |x(k,j,i)| \tag{7a}$$

$$s(k,i)_{fp} = \frac{x_{max}(k,i)}{(2^{N-1}-1)} \tag{7b}$$

$$x_q(k,j,i)_{int} = \left\lfloor \frac{x(k,j,i)}{s(k,i)} \right\rceil \tag{7c}$$

$$x^s_{q1}(k,j,i)_{fp} = x_q(k,j,i)s(k,i) \tag{7d}$$

$$s_{max}(k)_{fp} = \max_i s(k,i) \tag{7e}$$

$$\gamma(k)_{fp} = \frac{s_{max}(k)}{2^M - 1} \tag{7f}$$

$$s_q(k,i)_{int} = \left\lfloor \frac{s(k,i)}{\gamma(k)} \right\rceil \tag{7g}$$

$$s_{q2}(k,i)_{fp} = s_q(k,i)\gamma(k) \tag{7h}$$

$$x^s_{q2}(k,j,i)_{fp} = x_q(k,j,i)s_{q2}(k,i) \tag{7i}$$

$$= x_q(k,j,i)s_q(k,i)\gamma(k) \tag{7j}$$

To determine the per-vector scale factors, the algorithm computes the absolute maximum over the elements $j \in [0, V-1]$ of each vector (k,i) in Equation 7a and then determines the floating-point per-vector scale factor that would scale the absolute maximum to the maximum representable N-bit signed integer. This step is analogous to Equation 1 but at a per-vector granularity. Equation 7c performs the actual per-vector scaling and rounds the resulting tensor values to integers which will be used in our integer dot-product unit. Note that the scale factor here is per-vector for each (k,i) but broadcasted correspondingly to each element (k,j,i) of the tensor. At this point, we have everything we need if we were doing a single-level quantization with floating-point scale factors per-vector. The single-level simulated-quantized value is expressed in Equation 7d.

To further quantize the scale factor, we repeat the quantization process of taking the absolute maximum, computing the ratio of real valued maximum to integer maximum, and scaling and rounding to integer on the single-level scale factor as shown in Equations 7e to 7g. Equation 7h shows the two-level scale factor as a composition of integer per-vector scale factor and floating-point per-channel scale factor. The two-level simulated-quantized value is therefore represented as the product of the integer tensor values and the two levels of scale factors, as shown in Equation 7j.

Using two-level quantization for calibrating scale factors, DNN inference accuracy with PTQ across a range of weight, activation, and scale factor bitwidths is shown in Tables 5, 6, and 7. We compare the accuracy of *VS-Quant* with two-level scaling using low-bitwidth integer and fp16 scale factors to *VS-Quant* with fp32 scale factors and per-channel scaling (similar to Table 3). Compared to per-vector scaling, we consistently observe significantly lower accuracy loss with *VS-Quant* across all three DNNs, particularly at low weight and activation bitwidths. For example, *VS-Quant* with 3-bit weights and 8-bit activations achieves over 89% accuracy for BERT-large on SQuAD while the best per-channel calibrated quantization only achieves 11.59% accuracy.

The two-level quantization algorithm in Equation 7 is merely one of several ways to determine the two levels of scale factors. For example, instead of first computing the single-level per-vector scale factor and then breaking it down into the product of two levels of scale factors, we can do it one level at a time by first computing the per-channel scale factor and then back-calculating the per-vector scale factor. While this approach provides a larger space to explore the integer values and integer scale factors, it requires computing the absolute maximum over a larger tensor as opposed to just a vector, and is more expensive to implement in hardware.

5 Hardware Implementation

To evaluate the hardware efficiency of *VS-Quant*, we extended a previous optimized DNN accelerator (Venkatesan et al., 2019) by adding per-vector scaling support. Fig-

Bitwidths	S=3/4	S=3/6	S=4/4	S=4/6	S=6/4	S=6/6	S=fp32	POC
Wt=4 Act=3U	72.64	73.51	73.53	74.33	73.82	74.69	74.71	67.20
Wt=4 Act=4U	73.39	74.20	74.36	75.04	74.58	75.35	75.28	70.76
Wt=4 Act=6U	73.68	74.45	74.64	75.25	74.89	75.40	75.40	72.20
Wt=4 Act=8U	73.65	74.42	74.66	75.21	74.83	75.42	75.42	72.30
Wt=6 Act=3U	73.57	74.36	74.22	74.99	74.35	75.32	75.23	71.52
Wt=6 Act=4U	74.26	75.12	74.95	75.59	75.14	75.80	75.83	74.77
Wt=6 Act=6U	74.69	75.13	75.13	75.74	75.40	75.96	76.00	75.80
Wt=6 Act=8U	74.55	75.19	75.19	75.73	75.41	76.02	76.03	75.89
Wt=8 Act=3U	73.65	74.47	74.24	75.13	74.67	75.35	75.56	71.98
Wt=8 Act=4U	74.48	75.16	75.08	75.71	75.21	75.96	75.91	75.11
Wt=8 Act=6U	74.77	75.32	75.26	75.86	75.46	76.01	76.17	76.01
Wt=8 Act=8U	74.61	75.33	75.15	75.85	75.47	76.10	76.15	76.16

Table 5. **ResNet50 on ImageNet with integer scale factors**

Bitwidths	S=4/8	S=4/10	S=6/8	S=6/10	S=fp16	S=fp32	POC
Wt=3 Act=8	81.57	81.77	82.64	82.80	82.90	82.93	12.11
Wt=4 Act=8	85.65	85.88	86.00	86.33	86.27	86.26	78.86
Wt=6 Act=8	85.87	86.22	86.29	86.65	86.59	86.63	85.47
Wt=8 Act=8	85.94	86.38	86.37	86.56	86.57	86.54	85.67

Table 6. **BERT-base on SQuAD with integer scale factors**

Bitwidths	S=4/8	S=4/10	S=6/8	S=6/10	S=fp16	S=fp32	POC
Wt=3 Act=6	84.99	85.56	85.78	86.18	86.50	86.36	8.24
Wt=3 Act=8	88.72	88.96	89.38	89.58	89.60	89.54	11.59
Wt=4 Act=6	87.36	87.98	88.19	88.36	88.46	88.43	50.45
Wt=4 Act=8	90.28	90.51	90.56	90.68	90.64	90.66	86.75
Wt=6 Act=6	87.78	88.22	88.32	88.65	88.81	88.67	84.17
Wt=6 Act=8	90.45	90.54	90.67	90.81	90.83	90.76	90.58
Wt=8 Act=6	87.98	88.38	88.41	88.84	88.73	88.68	86.44
Wt=8 Act=8	90.59	90.55	90.57	90.82	90.85	90.88	90.81

Table 7. **BERT-large on SQuAD with integer scale factors**

Tables 5-7. **Accuracy of different networks with integer scale factors –** Accuracy numbers are color-coded from highest (dark blue) to lowest acceptable (dark red). `S=Sw/Sa` indicates `Sw`-bit unsigned per-vector weight scale factors and `Sa`-bit unsigned per-vector activation scale factors. `S=fp16` and `S=fp32` indicate single-level fp16 and fp32 per-vector scale factors. `POC` column lists best accuracy achieved for per-channel scaled quantization.

ure 2(a) shows the micro-architecture of a processing element (PE) in the accelerator, which is responsible for the dot-product computation listed in Equations 4 and 5. The PE consists of a set of *VS-Quant* vector MAC units, a weight buffer, an input activation buffer, an accumulation collector, a *VS-Quant* post-processing unit, and control logic.

Each *VS-Quant* vector MAC unit, shown in Figure 2(b), performs a V-element dot-product between the corresponding weight and activation data. In parallel, the product of the per-vector weight scale factor s_w and activation scale factor s_a is computed and rounded to the desired precision. The two outputs are then multiplied to compute a scaled partial sum output. Each entry of the weight buffer stores a weight vector along with corresponding per-vector scale factor. Similarly, the input activation buffer stores an activation vector and a per-vector scale factor in each row. The accumulation collector stores partial sum values from all the vector MAC computations and temporally accumulates them across multiple cycles in an integer format. For N-bit weights and activations along with M-bit weight and activation scale factors, we have $N \times N \rightarrow 2N$-bit products that are accumulated over the vector size V, resulting in $2N + log_2V$ wide dot-product outputs. The dot-product results are multiplied with the product of the M-bit weight and activation scale factors to produce $2N + log_2V + 2M$ wide partial sums. For improved energy efficiency, the vector MAC unit can optionally round the product of the scale factors to fewer than $2M$ bits before multiplying with the dot-product result. Finally, the accumulation collectors are designed with appropriate widths to avoid overflow. Taken together, the PE achieves efficient data reuse across all three data types: (i) each input activation vector is shared spatially across multiple vector MAC units; (ii) weight vectors are reused temporally across multiple cycles using a weight collector; (iii) partial sums are reused spatially inside the vector MAC unit and temporally in the accumulation collector.

For post-processing, the output of the accumulation collector is fed to a post-processing unit (PPU). To implement dynamic calibration for the scale factors of the activations, we perform the required calibration operations in the *VS-Quant* PPU and convert the higher-precision output activations back to N-bit vector elements with per-vector scale factors for the next layer. Figure 2(c) shows the block diagram of the *VS-Quant* PPU that performs the calibrate-and-quantize operations. As a post-processing step following the completion of a layer of computation, we leverage a vector max unit to implement Equation 7a to compute the absolute maximum of each vector of elements. Then a reciprocal unit and shifter implement Equation 7b to compute the ratios of absolute maximums of the vector to the maximum representable value of an N-bit integer value. The computed ratios are the scale factors used to quantize the output activations and convert them to *VS-Quant* format for computation of the next layer.

To quantify the area and energy impact of supporting *VS-Quant* in hardware, we also consider a baseline PE architecture for comparison without the scale factor related multipliers in the vector MAC unit and without the scale factor overheads in the weight and activation buffers. In this case, each vector MAC unit simply performs a V-wide dot-product and produces a partial sum of width $2N + log_2V$ for N-bit weights and activations. Per-channel scaling is performed in the baseline design PPU.

We evaluate the impact on energy per operation of *VS-Quant* compared to the baseline design using the MAGNet DNN generator and exploration infrastructure (Venkatesan et al., 2019). MAGNet's published 8-bit configuration achieved 2.1 tera-operations/sec/mm^2 (TOPS/mm^2) and 69

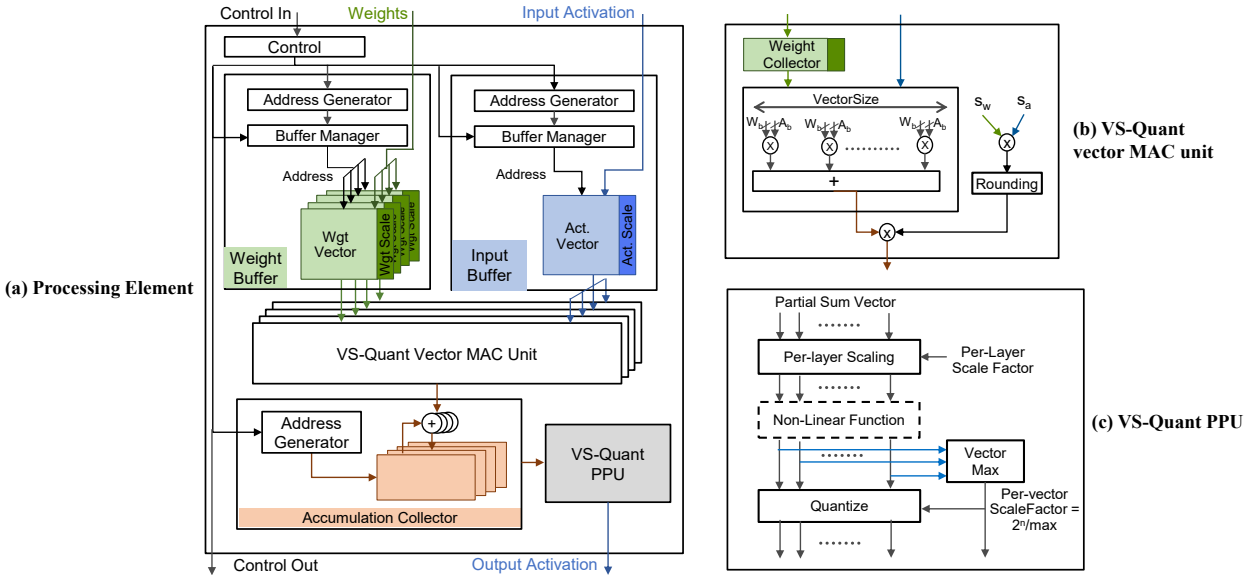

Figure 2. **Hardware diagram** — DNN accelerator with per-vector scaling support.

fJ/operation (14.5 TOPS/Watt) in a 16nm FinFET technology node. We normalize all subsequent energy and area numbers in this paper to a similar baseline design with 8-bit weights and activations. The design tools shown in Table 9 of Appendix A are used to implement the hardware and measure area and power in a sub-16nm process technology.

Figure 3 shows the average energy per operation across a range of hardware configurations. In this and all subsequent plots, we use `W/A/ws/as` to denote each configuration, where `W` stands for weight bitwidth, `A` for activation bitwidth, `ws` for weight scale bitwidth, and `as` for activation scale bitwidth. – indicates use of per-channel scaling. Energy is normalized to that of the 8/8/-/- configuration. The black bars for the per-channel scaled configurations (4/4/-/- and 8/8/-/-) show that quantization can achieve up to 2x energy savings over an 8-bit baseline. When the *VS-Quant* hardware is introduced and the scale factor product ($s_w \times s_a$) in Figure 2(b) is kept at full-bitwidth precision (i.e., no rounding), the yellow bars for the 4/4/4/6 and 4/4/8/8 configurations show energy overheads over corresponding 4-bit per-channel scaled configurations due to additional multipliers for scaling and wider accumulation widths. When the scale factor product is rounded to an intermediate size of 8 bits or 12 bits, the energy overheads of adding *VS-Quant* support to the hardware can be substantially reduced, as demonstrated by the blue and orange bars. As a result, per-vector scaled configurations with rounding can achieve lower energy consumption for a given accuracy target.

6 Design Space Exploration

To better understand the accuracy, energy, and area tradeoffs enabled by *VS-Quant*, we combine the energy and area

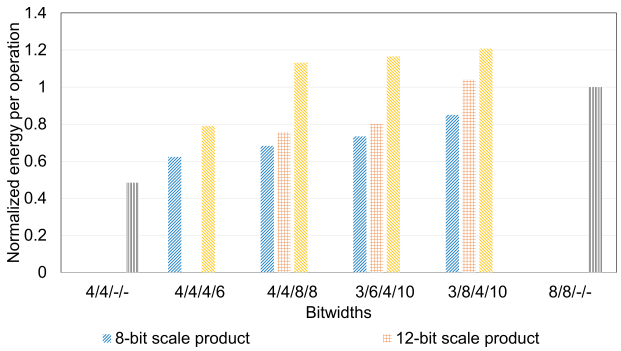

Figure 3. **Effect of scale product bitwidth on energy –** For this and subsequent figures, `W/A/ws/as` indicates weight, activation, weight scale, and activation scale bitwidths. Dashes indicate per-channel/per-layer scaling for weights/activations.

results from our DNN inference accelerator with accuracy results from real networks using a Pytorch-based PTQ library (Wu et al., 2020). Table 9 in Appendix A details the design tools used and parameters explored in our evaluation.

Figures 4, 5, and 6 present the design spaces of ResNet50, BERT-base, and BERT-large, respectively, for various bitwidth configurations of our DNN accelerator hardware. Results are shown as a tradeoff among energy efficiency (x-axis), area efficiency as performance per unit area (y-axis), and inference accuracy (color/shape). Since all configurations run with the same throughput (operations per cycle), performance is identical and only the VLSI energy and area costs vary. Each point in the plot reports metrics for a synthesized hardware instance selected from the set of precision parameter options in Table 9 of Appendix A, normalized to our baseline design (8/8/-/- configuration). Energy results are averaged over layers of the networks, weighted by the number of operations in each layer. For each network,

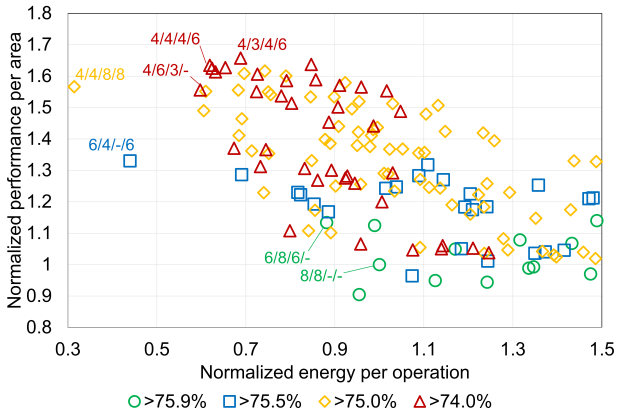

Figure 4. **ResNet50 design space**

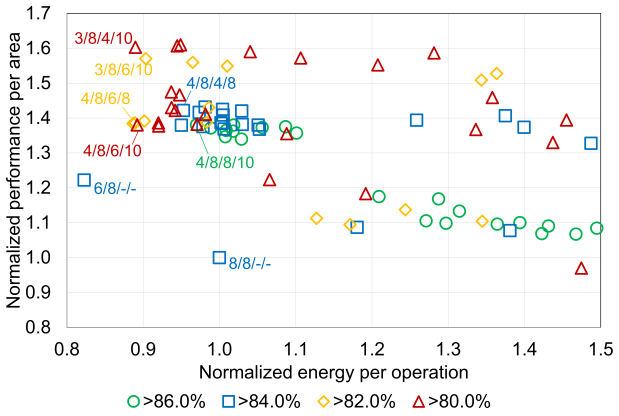

Figure 5. **BERT-base design space**

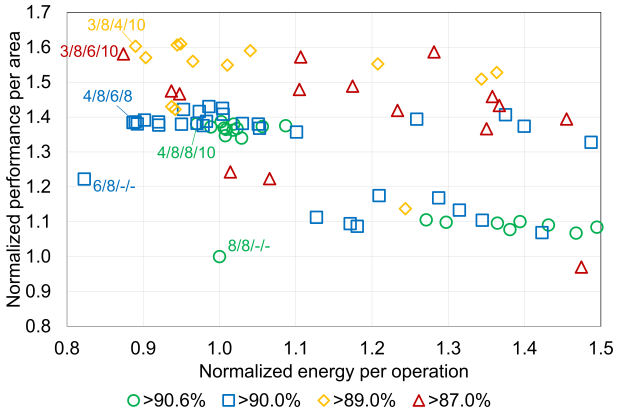

Figure 6. **BERT-large design space**

we decide the acceptable amount of accuracy loss against the full-precision baseline and only visualize those design points that are within an acceptable accuracy range.

As indicated in the legends in Figures 4, 5, and 6, we plot only ResNet50, BERT-base, and BERT-large design points that have an accuracy above 74.0%, 80.0%, and 87.0%, respectively. We then subdivide the acceptable range into finer accuracy ranges (four colors/shapes) to help visualize the achieved accuracy on top of the area-energy space. For design points of the same color/shape (within the same accuracy range), the upper left of the plot is optimal with the lowest energy per operation and highest performance per area. We label the prominent Pareto-optimal points as well as any per-channel scaled points. Overall, *VS-Quant* provides a much more expansive space of design tradeoffs than baseline 4-bit, 6-bit, and 8-bit datapaths, which we discuss in detail for each network below. For some *VS-Quant* points with per-vector scaling on both weights and activations, we use scale product rounding (typically 4-8 bits) to improve energy efficiency. We include the measured loss from scale product rounding in our accuracy data.

For ResNet50 results (Figure 4), the baseline 8/8/-/- already has minimal accuracy loss compared to the floating-point reference, so limited accuracy gains are available from *VS-Quant*. However, the green/circle 6/8/6/- *VS-Quant* point (6-bit weights, 8-bit activations, and 6-bit per-vector scale factors for weights) provides 12% lower energy as well as 12% smaller area at similar accuracy. When moving to 4-bit and 6-bit representations, *VS-Quant* provides even more energy and area reductions in the slightly lower accuracy range. For example, in the >75.5% accuracy range (blue/square points), the 6/4/-/6 design point achieves 57% less energy and 25% smaller area than the baseline design. In the >75.0% accuracy range (yellow/rhombus points), the 4/4/8/8 design point achieves 69% less energy and 36% smaller area than the baseline design. When moving to even lower accuracy in the >74.0% range (red/triangle points), even smaller area can be found at 4/4/4/6 and 4/3/4/6. In prior work, limiting accuracy loss to 1-2% with 4-to-6-bit integer representations had only been possible with QAT.

Figures 5 and 6 highlight the best energy efficiency and area achieved for BERT-base and BERT-large similarly at different accuracy targets. For both BERT models, *VS-Quant* is observed to be the most competitive across multiple accuracy targets, requiring very few bits for representing weights. In particular, a 4/8/8/10 configuration (4-bit weights, 8-bit activations, 8-bit per-vector weight scale factors, and 10-bit per-vector activation scale factors) for either model can achieve an accuracy target close to that of the full-precision baseline while reducing area by 28% and energy by 3%. For BERT-base, this kind of accuracy is not attainable even with our baseline design (8-bit per-channel scaled quantization) according to Table 2. If we relax our accuracy requirement to at least 82.0% for BERT-base and 87.0% for BERT-large, we can further decrease area and energy by dropping weight precision to only 3 bits. Based on the design points, the only BERT configuration where it makes sense to implement per-channel scaled quantization is the 6/8/-/- configuration targeting around 1% accuracy loss, although this configuration trades off significant area to attain the lowest energy in that accuracy range. On the other hand, configurations such as 4/8/6/8 for BERT-base and BERT-large are able to save energy without compromising on area. In comparison

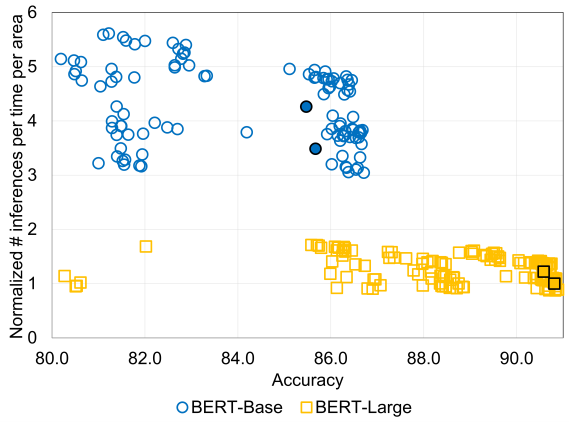

Figure 7. **Accuracy and area tradeoff for BERT models of different sizes –** Per-channel design points are outlined and filled.

to these lower-energy points, optimal *VS-Quant* hardware configurations such as 4/8/8/10 achieve great accuracy on both BERT-base and BERT-large.

We further study how the size of a network affects its accuracy, energy, and area tradeoff by comparing the design points of BERT-base against those of BERT-large. As shown in Figure 7, for example, BERT-large is the only choice if the accuracy target is beyond the best that BERT-base is able to achieve. Below that threshold, we should always select BERT-base because it is consistently more area-efficient than BERT-large. This suggests that one should configure the size of the model based on the desired accuracy target to realize the best hardware efficiency.

7 Quantization-aware Retraining

While we are able to leverage per-vector scaled PTQ to maintain reasonable accuracy down to 3 bits in some cases, accuracy loss is inevitable at low precision without QAT when compared to a full-precision baseline. The loss can be substantial if an inferior combination of weight and activation precisions is used. For example, BERT generally requires 8-bit precision for activations to get reasonable accuracy even with *VS-Quant*. Furthermore, many practical inference deployment scenarios may not have QAT as an option due to lack of access to full training datasets or limits on compute time and tuning effort. However, there are cases in which we can finetune a pretrained model with quantization in-the-loop for only a limited number of iterations to adapt the weights and activations to the quantized model (McKinstry et al., 2018).

VS-Quant is not limited to PTQ and can also be applied to QAT to achieve even higher accuracy for a given set of bitwidths. We apply per-vector scaled QAT using a conventional QAT framework that leverages a straight-through estimator (STE) in the backward pass to propagate the gradient through any quantizer. While the framework trains the weights that get fed into the quantizers in the model, the quantization scale factors are not parameters and are not explicitly trained. Table 8 evaluates the best accuracy achieved with QAT-based finetuning for both per-vector scaled quantization and per-channel scaled quantization. The number of retraining epochs taken to recover the specified accuracy is shown in parentheses. Based on the presented cases in Table 8, per-vector scaled QAT gives significantly better accuracy than per-channel scaled QAT and requires much less effort to recover accuracy loss from quantization.

Model	Bitwidths	Accuracy with QAT	
		PVAW	POC
ResNet50	Wt=3 Act=3U	75.53 (20)	72.02 (20)
BERT-base	Wt=4 Act=4	86.21 (5)	73.29 (20)
	Wt=4 Act=8	87.60 (1)	86.81 (1)
BERT-large	Wt=3 Act=4	89.16 (2)	21.63 (2)
	Wt=3 Act=8	90.43 (1)	88.84 (1)

Table 8. **QAT study –** Compares best accuracy achieved after QAT-based finetuning, with number of finetuned epochs in parentheses.

8 Conclusions

In this paper, we introduced *VS-Quant*, a novel per-vector scaled quantization technique that employs per-vector scale factors to mitigate accuracy loss typical in existing quantized DNN models. To support efficient per-vector scaling in hardware, we implemented a two-level scaling scheme and associated algorithm that combine a set of fine-grained scale factors with each coarse-grained scale factor. We evaluated *VS-Quant* on a set of popular DNN models and tasks and demonstrated that it achieves significant improvement in post-training quantization accuracy when compared to conventional per-channel scaled quantization techniques.

By extending the vector MAC unit of a DNN accelerator to dynamically support per-vector scaling at inference-time, we analyze the area and power implications of per-vector scaling on the hardware. Experiments demonstrate that *VS-Quant* with 4-bit weights and activations achieves 69% energy saving and 36% area saving while maintaining over 75% accuracy for ResNet50 on ImageNet. Furthermore, *VS-Quant* with 4-bit weights and 8-bit activations achieves near-full-precision accuracy for both BERT-base and BERT-large on SQuAD while reducing area by 28% compared to a non-*VS-Quant* 8-bit baseline. By exploring the design space, we find that per-vector scaling provides better accuracy, energy, and area tradeoffs for low-precision inference. For future work, we will continue to optimize the *VS-Quant* hardware and study scale factor and other intermediate rounding. We will extend QAT to explicitly learn per-vector scale factors.

Acknowledgements

We would like to thank Hao Wu and Patrick Judd for useful discussions and help with quantization-related software.

REFERENCES

Bhandare, A., Sripathi, V., Karkada, D., Menon, V., Choi, S., Datta, K., and Saletore, V. Efficient 8-bit Quantization of Transformer Neural Machine Language Translation Model. *arXiv preprint arXiv:1906.00532*, 2019.

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. ZeroQ: A Novel Zero Shot Quantization Framework. *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., and Gopalakrishnan, K. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *arXiv preprint arXiv:1805.06085*, 2018.

Courbariaux, M., Bengio, Y., and David, J.-P. Training Deep Neural Networks with Low Precision Multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

Courbariaux, M., Bengio, Y., and David, J.-P. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2015.

Fang, J., Shafiee, A., Abdel-Aziz, H., Thorsley, D., Georgiadis, G., and Hassoun, J. Post-Training Piecewise Linear Quantization for Deep Neural Networks. *European Conference on Computer Vision (ECCV)*, 2020.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Gray, R. Vector Quantization. *IEEE ASSP Magazine*, 1984.

Han, S., Mao, H., and Dally, W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*, 2015.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized Neural Networks. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2016.

Jain, S., Venkataramani, S., Srinivasan, V., Choi, J., Gopalakrishnan, K., and Chang, L. BiScaled-DNN: Quantizing Long-tailed Datastructures with Two Scale Factors for Deep Neural Networks. *Design Automation Conf. (DAC)*, 2019.

Khoram, S. and Li, J. Adaptive Quantization of Neural Networks. *Int'l Conf. on Learning Representations (ICLR)*, 2018.

Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A. K., Constable, W., Elibol, O., Gray, S., Hall, S., Hornof, L., et al. Flexpoint: An adaptive Numerical Format for Efficient Training of Deep Neural Networks. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2017.

Krishnamoorthi, R. Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

LeCun, Y., Bengio, Y., and Hinton, G. Deep Learning. *Nature*, 521(7553):436–444, 2015.

Lee, J. H., Ha, S., Choi, S., Lee, W.-J., and Lee, S. Quantization for Rapid Deployment of Deep Neural Networks. *arXiv preprint arXiv:1810.05488*, 2018.

McKinstry, J. L., Esser, S. K., Appuswamy, R., Bablani, D., Arthur, J. V., Yildiz, I. B., and Modha, D. S. Discovering Low-precision Networks Close to Full-precision Networks for Efficient Embedded Inference. *arXiv preprint arXiv:1809.04191*, 2018.

Mishra, A. and Marr, D. Apprentice: Using Knowledge Distillation Techniques to Improve Low-precision Network Accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

Miyashita, D., Lee, E. H., and Murmann, B. Convolutional Neural Networks using Logarithmic Data Representation. *arXiv preprint arXiv:1603.01025*, 2016.

Moons, B., Goetschalckx, K., Van Berckelaer, N., and Verhelst, M. Minimum Energy Quantized Neural Networks. *Asilomar Conference on Signals, Systems, and Computers*, 2017.

Nagel, M., Baalen, M. v., Blankevoort, T., and Welling, M. Data-free Quantization through Weight Equalization and Bias Correction. *Int'l Conf. on Computer Vision (ICCV)*, 2019.

NVIDIA Corporation. NVIDIA A100 Tensor Core GPU Architecture. *NVIDIA*, 2020.

Prato, G., Charlaix, E., and Rezagholizadeh, M. Fully Quantized Transformer for Improved Translation. *arXiv preprint arXiv:1910.10485*, 2019.

Rouhani, B., Lo, D., Zhao, R., Liu, M., Fowers, J., Ovtcharov, K., Vinogradsky, A., Massengill, S., Yang, L., Bittner, R., et al. Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. *AAAI Conf. on Artificial Intelligence (AAAI)*, 2020.

Sijstermans, F. The NVIDIA Deep Learning Accelerator.

Stock, P., Joulin, A., Gribonval, R., Graham, B., and Jégou, H. And the Bit Goes Down: Revisiting the Quantization of Neural Networks. *Int'l Conf. on Learning Representations (ICLR)*, 2020.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. Efficient Processing of Deep Neural Networks. *Synthesis Lectures on Computer Architecture*, 2020.

Tambe, T., Yang, E.-Y., Wan, Z., Deng, Y., Reddi, V. J., Rush, A., Brooks, D., and Wei, G.-Y. Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference. *Design Automation Conference (DAC)*, 2020.

Venkatesan, R., Shao, Y. S., Wang, M., Clemons, J., Dai, S., Fojtik, M., Keller, B., Klinefelter, A., Pinckney, N. R., Raina, P., et al. MAGNet: A Modular Accelerator Generator for Neural Networks. *Int'l Conf. on Computer Aided Design (ICCAD)*, 2019.

Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., and Keutzer, K. Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search. *arXiv preprint arXiv:1812.00090*, 2018.

Wu, H. Low Precision Inference on GPUs. *GPU Technology Conference (GTC)*, 2019.

Wu, H., Judd, P., Zhang, X., Isaev, M., and Micikevicius, P. Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. *arXiv preprint arXiv:2004.09602*, 2020.

Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. Quantized Convolutional Neural Networks for Mobile Devices. *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016a.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*, 2016b.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. Q8bert: Quantized 8bit BERT. *arXiv preprint arXiv:1910.06188*, 2019.

Zhao, R., Hu, Y., Dotzel, J., De Sa, C., and Zhang, Z. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. *Int'l Conf. on Machine Learning (ICML)*, 2019.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained Ternary Quantization. *arXiv preprint arXiv:1612.01064*, 2016.

A EXPERIMENTAL SETUP

Design tools	
HLS Compiler	Mentor Graphics Catapult HLS
Verilog simulator	Synopsys VCS
Logic synthesis	Synopsys Design Compiler Graphical
Place-and-route	Synopsys ICC2
Power analysis	Synopsys PT-PX
Design space	
Vector size	16
Weight/activation precision	3-bit, 4-bit, 6-bit, 8-bit
Weight/activation scale precision	3-bit, 4-bit, 6-bit, 8-bit, 10-bit
Scaling granularity	POC, PVAO, PVWO, PVAW

Table 9. **Experimental setup –** POC = per-channel, PVAO = per-vector on activations only, PVWO = per-vector on weights only, PVAW = per-vector on both weights and activations.