INTEGER-ONLY ZERO-SHOT QUANTIZATION FOR EFFICIENT SPEECH RECOGNITION

Sehoon Kim, Amir Gholami, Zhewei Yao, Nicholas Lee, Patrick Wang,
Aniruddha Nrusimha, Bohan Zhai, Tianren Gao, Michael W. Mahoney, Kurt Keutzer

University of California, Berkeley

ABSTRACT

End-to-end neural network models achieve improved performance on various automatic speech recognition (ASR) tasks. However, these models perform poorly on edge hardware due to large memory and computation requirements. While quantizing model weights and/or activations to low-precision can be a promising solution, previous research on quantizing ASR models is limited. In particular, the previous approaches use floating-point arithmetic during inference and thus they cannot fully exploit efficient integer processing units. Moreover, they require training and/or validation data during quantization, which may not be available due to security or privacy concerns. To address these limitations, we propose an integer-only, zero-shot quantization scheme for ASR models. In particular, we generate synthetic data whose runtime statistics resemble the real data, and we use it to calibrate models during quantization. We apply our method to quantize QuartzNet, Jasper, and Conformer and show negligible WER degradation as compared to the full-precision baseline models, even without using any data. Moreover, we achieve up to 2.35× speedup on a T4 GPU and 4× compression rate, with a modest WER degradation of <1% with INT8 quantization.

Index Terms— Automatic speech recognition, quantization, compression, integer-only, efficient inference

1. INTRODUCTION

End-to-end neural network models have achieved state-of-the-art results in various automatic speech recognition (ASR) tasks [1, 2, 3, 4, 5]. However, these accuracy improvements come with increasingly large model sizes. For instance, the highest performing versions of Jasper [1], Conformer [2], ContextNet [3], and Transformer-Transducer [4] contain 333M, 118.8M, 112.7M, and 139M parameters. This has made real-time deployment of these models challenging at the edge.

One promising solution to tackle this challenge is to quantize model weights and/or activations to low-precision. This provides multiple benefits. First, quantization reduces the memory footprint by storing weights and/or activations in low-precision. For instance, INT8 uses 4× less memory relative to FP32. Second, quantization accelerates execution and decreases power consumption by using specialized hardware for low-precision arithmetic [6, 7, 8]. Following its success on many computer vision [6, 7] and natural language processing tasks [8, 9, 10], there have been attempts to apply quantization to ASR models [11, 12, 13, 14]. However, prior quantization work lacks two important features: integer-only quantization and zero-shot quantization.

Integer-only quantization [6, 7, 8] is a quantization scheme where *all* operations (e.g., convolution and matrix multiplication) are performed using low-precision integer arithmetic. To the best of our knowledge, all the prior quantization methods for ASR models use *simulated quantization*, where all or part of operations are performed with floating point arithmetic. For instance, [13] only performs convolution with integer arithmetic and leaves ReLU and Batch Normalization (BatchNorm) as floating point operations. Compared to simulated quantization, integer-only quantization can decrease latency and power consumption by fully utilizing efficient integer processing units. More importantly, it also allows deployment on popular and highly optimized edge processors designed for embedded, mobile, or IoT devices that often do not support floating point arithmetic. ARM Cortex-M [15], GreenWaves GAP-9 [16], and Google's Edge TPU [17] are some examples of edge processors without dedicated floating point units.

Zero-shot quantization [18, 19, 20, 21, 22, 23] is a quantization scheme that does not require any training or validation data. This is important because datasets are not always available, especially for ASR use cases such as smart speakers or healthcare, where privacy and security are concerns. However, prior work is based on either quantization-aware training [11, 12, 13, 14] or post-training quantization [13, 14], both of which require access to training and/or validation data to finetune or calibrate the quantized models. As such, these methods cannot be applied if no data is available.

We propose to address the aforementioned limitations of the prior quantization work for ASR models. In particular:

- We develop an integer-only, zero-shot quantization scheme for ASR models, which allows efficient deployment on integer processing units and achieves negligible WER degradation even without any access to training and/or validation data (§ 2.2 and § 2.3).
- We apply our method to QuartzNet-15x5 [5], JasperDR-10x5 [1], and Conformer-Large [2] and evaluate their word-to-error-rate (WER) on the Librispeech benchmark [24]. With 8-bit quantization for weights and activations, we

achieve negligible WER degradation of up to 0.29%, 0.08%, and 0.87% for each model, respectively (§ 3.1).

- We deploy INT8-only QuartzNet on a T4 GPU, and show up to $2.35\times$ speedup compared to its FP32 counterpart (§ 3.2).

2. METHODOLOGY

2.1. Basic Quantization Method

In this work, we use *uniform symmetric quantization*. This method uniformly maps a real number x to an integer value q:

$$q = \mathrm{Q}(x, b, S) = \mathrm{Int}\left(\frac{\mathrm{clip}(x, -\alpha, \alpha)}{S}\right), \tag{1}$$

where $q \in [-2^{b-1}, 2^{b-1} - 1]$, b is the quantization bit-width, Q is the quantization operator, Int is the round operation, clip is the truncation function with the clipping parameter α, and S is the scaling factor defined as $\alpha/(2^{b-1} - 1)$. The reverse mapping from the quantized values q to the real values (aka dequantization) is $\tilde{x} = \mathrm{DQ}(q, S) = Sq \approx x$, where DQ denotes the dequantization operator.

Determining the clipping range $[-\alpha, \alpha]$ that best represents the range of the input x is a primary challenge of quantization. One popular choice in practice is to use the minimum and maximum values of the input x, i.e., $\alpha = \max(|x_{max}|, |x_{min}|)$. However, this approach is susceptible to outliers. An unnecessarily large clipping range due to a few outliers increases the rounding error of quantized values within the range. One way to alleviate this is to use percentile values, e.g., 99% smallest/largest, instead of the min/max values.

In *dynamic quantization*, clipping ranges are computed during inference. However, calculating input statistics (e.g., min, max, and percentile) can be costly in real-time, and requires floating point operations that would prevent us from doing integer-only quantization. Therefore, in this work, we only consider *static quantization* where we pre-compute the clipping ranges and fix them during inference as in [6, 7, 8]. It is straightforward to pre-compute the ranges for weights as they have fixed values during inference. However, activations vary across different inputs, and therefore their ranges also vary. One popular method to address this is *calibration*, which runs a series of training data to compute the typical range of activations. Later in § 2.3, we extend this idea and show how to calibrate *without* training data. We refer the interested readers to [25] for more details in quantization methods.

2.2. Integer-only Quantization

Integer-only quantization [6, 7, 8] not only represents the model weights and activations with low-precision integer values, but it also carries out the entire inference with integer arithmetic. Broadly speaking, the core of integer-only quantization is the linear property of the operations. For instance, let q_W and q_x denote the quantized values for weight $\tilde{W}$ and activation $\tilde{x}$, respectively. That is, $\tilde{W} = S_W q_W$ and $\tilde{x} = S_x q_x$ where S_W and S_x are the scaling factors for $\tilde{W}$ and $\tilde{x}$ (§ 2.1). Then, $\mathrm{Conv}(\tilde{W}, \tilde{x}) = \mathrm{Conv}(S_W q_W, S_x q_x)$ is equivalent to $S_W S_x \mathrm{Conv}(q_W, q_x)$ due to the linear property. Therefore, we can apply *integer* convolution directly to the quantized values q_W and q_x without having to dequantize them into $\tilde{W}$ and $\tilde{x}$ and apply floating point convolution [6, 25]. This property holds true for convolution, matrix multiplication, and ReLU activation, which are the basic building blocks for neural architectures. In addition, BatchNorms are folded into the preceding convolution layers [6], and non-linear activations (e.g., Sigmoid, Softmax and Swish in Conformer) are approximated with 2nd-order polynomials following the procedure in [8] to avoid their floating point execution.

2.3. Zero-shot Quantization

Quantization generally requires training data (1) to pre-compute the clipping ranges for calibration (Equation 1) and/or (2) to finetune the quantized model to recover accuracy degradation from quantization. However, in many cases, pre-trained models are distributed without the data they were trained on due to proprietary, privacy, and security concerns. As such, multiple zero-shot quantization schemes have been proposed, mostly in the field of computer vision, to allow quantization in such cases. A widely adopted method is to generate synthetic image data that can be used in place of the real training data for calibration and/or finetuning [18, 19, 20, 21, 22, 23].

Similar to [19, 21], we generate synthetic data with a similar data distribution to the training data. In particular, we generate the data to match the runtime statistics (i.e., hidden activation distributions) of the training data using the running means and variances stored in BatchNorm layers. Note that the running means and variances capture the runtime statistics of the training data. Let x denote the synthetic data, and μ_i and σ_i the mean and variance of the activation from the i-th BatchNorm with x as an input. Additionally, let $\hat{\mu}_i$ and $\hat{\sigma}_i$ denote the running mean and variance of the BatchNorm. Then, we aim to minimize the loss defined by the Kullback-Leibler (KL) divergence between $N(\hat{\mu}_i, \hat{\sigma}_i)$ and $N(\mu_i, \sigma_i)$, assuming that the hidden activations follow the Gaussian distribution:

$$\begin{aligned} x^* &= \arg\min_x \sum_i KL(N(\hat{\mu}_i, \hat{\sigma}_i)||N(\mu_i, \sigma_i)) \\ &= \arg\min_x \sum_i \log\frac{\sigma_i}{\hat{\sigma}_i} - \frac{1}{2}\left(1 - \frac{\hat{\sigma}_i^2 + (\hat{\mu}_i - \mu_i)^2}{\sigma_i^2}\right). \end{aligned} \tag{2}$$

In computer vision tasks, input x is often regarded as a learnable parameter and is trained with standard backpropagation to reduce the defined loss [19, 20]. However, it is not straightforward to apply this directly to speech data, as we are dealing with time varying audio signals rather than static images. As such, we instead propose to generate synthetic mel spectograms. Mel spectrogram is a widely used preprocessed representation of audio signals [1, 2, 3, 5, 26]. Also, its data representation is more suitable for synthetic data generation as

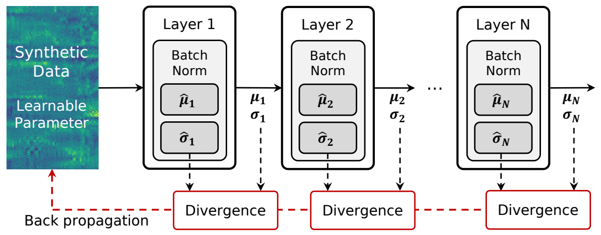

Fig. 1: End-to-end process of synthetic data (i.e., mel spectrogram) generation for zero-shot quantization. Input data is trained with standard backpropagation in a direction that matches the hidden activation statistics of the real data.

it can be regarded as a 1-dimensional image. Specifically, our zero-shot quantization scheme consists of two steps.

- **Synthetic data generation:** A batch of 1-dimensional arrays is randomly initialized and set as a learnable parameter. It is then fed into the full-precision (i.e., non-quantized) model to compute the loss in Equation 2. This minimizes the KL divergence between two distributions, one from the real training data and the other from the synthetic data, for each BatchNorm activation. Using backpropagation, the input is iteratively trained using gradient descent to minimize the loss. Figure 1 illustrates the end-to-end process.
- **Calibration:** The synthetic data is fed into the target model in place of the real mel spectrograms to determine the clipping ranges for all activations. Then the model is quantized according to Equation 1.

The benefit of this approach is that we can use this synthetic data to perform quantization, and avoid the need to get direct access to the training data which may not be available.

3. RESULTS

In this section, we first demonstrate in § 3.1 that the quantized models can match the accuracy of full-precision baselines. We then evaluate the latency speedup of the quantized models on real hardware in § 3.2. As baseline models, we select QuartzNet15x5 [5] and JasperDR-10x5 [1] as CNN-based architectures, and Conformer-Large [2] as a more complex architecture with multi-head attentions and non-linear activations such as Sigmoid, Softmax, and Swish. Such non-linear activations can be computed with integer arithmetic by approximating them with 2nd-order polynomials as described in [8]. We note that our quantization method is not a model-specific solution, and can be applied to many other ASR models.

3.1. Accuracy Evaluation

For accuracy evaluation, we use the NeMo [27] implementation of the pre-trained QuartzNet-15x5, JasperDR-10x5, and Conformer-Large for the full-precision baseline models. The synthetic data is initialized with the uniform distribution from $[-0.3, 0.3]$, and is trained using the Adam optimizer [28] with batch size 8, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and learning rate of {0.03, 0.04, 0.05, 0.06} for {200, 250} iterations. We generate 20 such batches and use them for calibration. Figure 3 contains some examples of the generated synthetic data. For calibration, we use the min/max values for clipping. We quantize the baseline models with W8A8 (i.e., 8-bit weights and activations) and W6A8, and do not finetune the models after quantization. All the reported numbers are averaged over 4 different runs (i.e., synthetic data generation and calibration).

Table 1: WER of the quantized QuartzNet15x5, JasperDR-10x5, and Conformer-Large with different bit-width settings, evaluated on LibriSpeech. W and A are the bit-width for weights and activations. Note that 32/32 indicates full-precision baseline. We also include the model size and BOPs (for 10-second input with sampling rate 16K) of each model.

Model	W/A	dev clean	dev other	test clean	test other	Size (MB)	BOPs (T)
QuartzNet	32/32	3.80	10.05	3.82	10.08	73.81	9.48
	8/8	3.92	10.28	4.04	10.37	18.45	0.61
	6/8	4.16	10.83	4.32	11.06	13.84	0.46
Jasper	32/32	3.47	10.40	3.68	10.49	1208	171
	8/8	3.47	10.49	3.68	10.57	302.0	10.7
	6/8	3.52	10.59	3.73	10.68	226.5	8.00
Conformer	32/32	2.47	6.04	2.78	6.19	494.8	44.0
	8/8	2.75	6.95	3.06	7.06	123.7	2.75
	6/8	3.63	8.21	4.03	8.48	92.78	2.06

Both the baseline and quantized models are evaluated on the widely used LibriSpeech [24] benchmark. Table 1 compares the WER of the baseline and quantized models with different bit-width settings on dev-clean/other and test-clean/other datasets, where hyperparameters are tuned on the dev sets. We also include the model size and BOPs (bit-operations) of each model for comparison. BOP is the total number of bit operations and is a hardware-agnostic proxy to model complexity [29]. For QuartzNet, the W8A8 setting results in a modest WER degradation of 0.23% and 0.29% on the test-clean/other datasets with $4\times$ reduction of the model size as compared to the full-precision baseline. The W6A8 setting further reduces the model size to $\sim 5\times$ of the baseline only within 0.50% and 0.97% WER degradation on the test sets. On Jasper, the quantized model exhibits negligible WER increase of up to 0.08% and 0.19% on the test sets with the W8A8 and W8A6 settings, respectively. Despite the complexity of the Conformer architecture, INT8-only version only exhibits 0.87% WER degradation. Note that all the reported values are *without* any finetuning after quantization.

3.2. Latency Evaluation

We evaluate the latency speedup of quantized models by direct deployment of the quantized QuartzNet on a Tesla T4 GPU with the Turing Tensor Cores that support accelerated INT8 execution. We use NVIDIA's TensorRT library [30] for model deployment. All the experiments were conducted on Google Cloud Platform virtual machine with a single Tesla T4 GPU, CUDA 11.1, cuDNN 8.0, and TensorRT 7.2.1. Although we

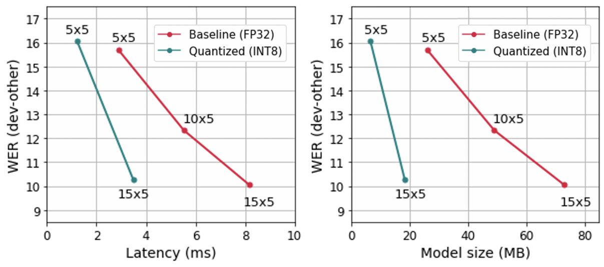

Fig. 2: Latency (Left) and model size (Right) of the quantized and baseline QuartzNets with different sizes. Latency is measured with 10-second audio input with sampling rate 16K.

select T4 GPU as our target device due to its extensive software support [30], we should highlight that our approach is not specific to GPUs, and one could also deploy the quantized models to other processors. For evaluation, we use QuartzNet-5x5, 10x5, and 15x5, which are the small, medium, and large variants of the QuartzNet family [5]. We measure the inference latency using a 10-second audio input with a sampling rate 16K, and test the W8A8 (i.e., INT8-only) setting for the quantized model.

Figure 2 compares the latency and model size of the baseline and quantized QuartzNet models with different size configurations. As shown in the plot, INT8-only QuartzNet15x5 achieves 2.35$\times$ speedup and 4$\times$ compression rate as compared to the FP32 baseline. Also, the quantized QuartzNet15x5 matches the WER of the largest model, while only needing the computation and memory requirements of the smallest model. This is an important observation for practitioners who wants to design an efficient model, as the alternative approach of using a full precision but shallower model can lead to a sub-optimal solution compared to quantization.

4. DISCUSSION

4.1. Ablation studies

Here, we show the effectiveness of the zero-shot quantization scheme. To do so, we quantize QuartzNet15x5 using a set of random calibration data from the uniform distribution in $[-3, 3]$. We observe significant WER degradation when we calibrate the model with random data instead of the synthetic data of our zero-shot quantization scheme. With random data, the test-other WER degrades by 2.47% and 4.02% for W8A8 and W6A8, respectively, which are noticeably higher than the corresponding values with the synthetic data as shown in Table 2. The results confirm the need to generate synthetic data that well represents the actual data used in training.

4.2. Analysis of Synthetic Data Generation

In this section, we briefly analyze the synthetic data, which we start with a randomly initialized array (second row of Figure 3) and solve Equation 2 to generate synthetic mel spectrograms. We show two such examples in the last three rows of Figure 3. First, note that the synthetic data can capture the local patterns observed in real mel spectrograms. Second, we observe that if the synthetic data is fed into QuartzNet followed by a 3-gram language model, it can be decoded into meaningful words and sentences, instead of a sequence of random characters. Examples of the decoded sentences are captioned below the corresponding synthetic data in Figure 3. This is a further evidence that the synthetic data is close to the actual training data, and it explains why it enables good quantization results.

Table 2: WER of the quantized QuartzNet15x5 calibrated with random data and the synthetic data. W and A are the bit-width for weights and activations, respectively. Calibration with the synthetic data consistently outperforms calibration with random data for all bit-width settings.

Method	W/A	dev		test		Size	BOPs
		clean	other	clean	other	(MB)	(T)
Baseline	32/32	3.80	10.05	3.82	10.08	73.81	9.48
Random	8/8	5.39	12.20	5.56	12.55	18.45	0.61
Synthetic	8/8	3.92	10.28	4.04	10.37	18.45	0.61
Random	6/8	6.26	13.57	6.56	14.10	13.84	0.46
Synthetic	6/8	4.16	10.83	4.32	11.06	13.84	0.46

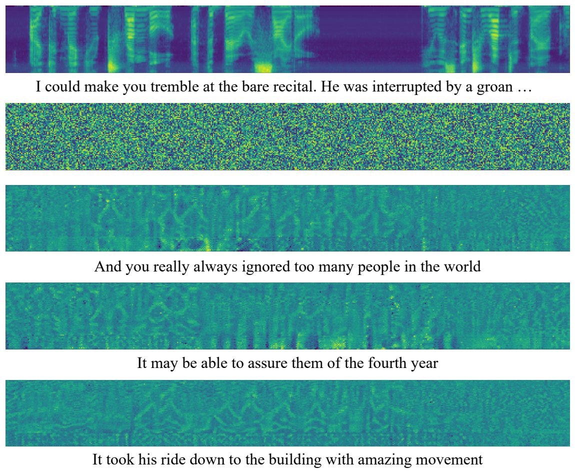

Fig. 3: Comparison of the real (1st), random (2nd), and synthetic (3rd-5th) mel spectrograms. Below each synthetic mel spectrogram is the text that it decodes to.

5. CONCLUSION

In this work, we propose a zero-shot quantization scheme for ASR models that only uses integer-only computation. Our proposed method entirely removes floating point operations from inference, thereby allowing efficient model deployment on fast and power-efficient integer processing units. In addition, it does not require any training and/or validation data, as it calibrates models with synthetic data that resembles the real data. Our quantization scheme exhibits a large compression rate of 4$\times$ with small WER degradation of 0.29%, 0.08%, and 0.87% on QuartzNet, Jasper, and Conformer for INT8-only quantization. Furthermore, we achieve 2.35$\times$ speedup by directly deploying INT8-only QuartzNet on a T4 GPU.

6. REFERENCES

[1] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M Cohen, Huyen Nguyen, and Ravi Teja Gadde, "Jasper: An end-to-end convolutional neural acoustic model," *arXiv preprint arXiv:1904.03288*, 2019.

[2] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang, "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[3] Wei Han, Zhengdong Zhang, Yu Zhang, Jiahui Yu, Chung-Cheng Chiu, James Qin, Anmol Gulati, Ruoming Pang, and Yonghui Wu, "ContextNet: Improving convolutional neural networks for automatic speech recognition with global context," *arXiv preprint arXiv:2005.03191*, 2020.

[4] Qian Zhang, Han Lu, Hasim Sak, Anshuman Tripathi, Erik McDermott, Stephen Koo, and Shankar Kumar, "Transformer Transducer: A streamable speech recognition model with transformer encoders and RNN-T loss," in *ICASSP*. IEEE, 2020, pp. 7829–7833.

[5] Samuel Kriman, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, and Yang Zhang, "QuartzNet: Deep automatic speech recognition with 1d time-channel separable convolutions," in *ICASSP*. IEEE, 2020, pp. 6124–6128.

[6] 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 *Proc. CVPR*, 2018, pp. 2704–2713.

[7] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W Mahoney, and Kurt Keutzer, "HAWQV3: Dyadic neural network quantization," *arXiv preprint arXiv:2011.10680*, 2020.

[8] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer, "I-BERT: Integer-only BERT quantization," *arXiv preprint arXiv:2101.01321*, 2021.

[9] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat, "Q8BERT: Quantized 8bit BERT," *arXiv preprint arXiv:1910.06188*, 2019.

[10] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer, "Q-BERT: Hessian based ultra low precision quantization of BERT," in *AAAI*, 2020, pp. 8815–8821.

[11] Yiwu Yao, Yuchao Li, Chengyu Wang, Tianhang Yu, Houjiang Chen, Xiaotang Jiang, Jun Yang, Jun Huang, Wei Lin, Hui Shu, and Chengfei Lv, "Int8 winograd acceleration for conv1d equipped asr models deployed on mobile devices," *arXiv preprint arXiv:2010.14841*, 2020.

[12] Hieu Duy Nguyen, Anastasios Alexandridis, and Athanasios Mouchtaris, "Quantization aware training with absolute-cosine regularization for automatic speech recognition," *Proc. Interspeech 2020*, pp. 3366–3370, 2020.

[13] Amrutha Prasad, Petr Motlicek, and Srikanth Madikeri, "Quantization of acoustic model parameters in automatic speech recognition framework," *arXiv preprint arXiv:2006.09054*, 2020.

[14] Alex Bie, Bharat Venkitesh, Joao Monteiro, Md Haidar, and Mehdi Rezagholizadeh, "A simplified fully quantized transformer for end-to-end speech recognition," *arXiv preprint arXiv:1911.03604*, 2019.

[15] ARM, "Cortex-M, https://developer.arm.com/ip-products/processors/cortex-m," .

[16] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini, "GAP-8: A RISC-V SoC for AI at the edge of the IoT," in *ASAP*. IEEE, 2018, pp. 1–4.

[17] "Google Edge TPU, https://cloud.google.com/edge-tpu," .

[18] Hanting Chen, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu, and Qi Tian, "Data-free learning of student networks," in *Proc. ICCV*, 2019, pp. 3514–3522.

[19] Matan Haroush, Itay Hubara, Elad Hoffer, and Daniel Soudry, "The knowledge within: Methods for data-free model compression," in *Proc. CVPR*, 2020, pp. 8494–8502.

[20] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer, "ZeroQ: A novel zero shot quantization framework," in *Proc. CVPR*, 2020, pp. 13169–13178.

[21] Yoojin Choi, Jihwan Choi, Mostafa El-Khamy, and Jungwon Lee, "Data-free network quantization with adversarial knowledge distillation," in *Proc. CVPR Workshops*, 2020, pp. 710–711.

[22] Shoukai Xu, Haokun Li, Bohan Zhuang, Jing Liu, Jiezhang Cao, Chuangrun Liang, and Mingkui Tan, "Generative low-bitwidth data free quantization," in *ECCV*. Springer, 2020, pp. 1–17.

[23] Xiangyu He, Qinghao Hu, Peisong Wang, and Jian Cheng, "Generative zero-shot network quantization," *arXiv preprint arXiv:2101.08430*, 2021.

[24] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *ICASSP*. IEEE, 2015, pp. 5206–5210.

[25] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[26] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le, "SpecAugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[27] "NeMo, https://github.com/nvidia/nemo," .

[28] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] Mart van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling, "Bayesian bits: Unifying quantization and pruning," *arXiv preprint arXiv:2005.07093*, 2020.

[30] NVIDIA, "TensorRT: https://developer.nvidia.com/tensorrt," .