TORCH2CHIP: AN END-TO-END CUSTOMIZABLE DEEP NEURAL NETWORK COMPRESSION AND DEPLOYMENT TOOLKIT FOR PROTOTYPE HARDWARE ACCELERATOR DESIGN

Jian Meng [1] **Yuan Liao** [1] **Anupreetham Anupreetham** [1] **Ahmed Hasssan** [1] **Shixing Yu** [1] **Han-sok Suh** [1] **Xiaofeng Hu** [1] **Jae-sun Seo** [1]

ABSTRACT

Deep neural network (DNN) compression (e.g., quantization, pruning) has been widely investigated in various deep learning tasks (e.g., vision and language). The development of model compression is continuously motivated by the evolution of various neural network accelerator designs with ASIC or FPGA. On the algorithm side, the ultimate goal of quantization or pruning is accelerating the expensive DNN computations on low-power hardware. However, such a "design-and-deploy" workflow faces under-explored challenges in the current hardware-algorithm co-design community due to some unavoidable flaws. First, although the state-of-the-art quantization algorithm can achieve ultra-low precision with negligible degradation of accuracy, the latest deep learning framework (e.g., PyTorch) can only support non-customizable 8-bit precision, data format, and parameter extraction workflow for CNN. Secondly, the ultimate goal of quantization is to enable the computation with low-precision data (e.g., 4-bit integer). However, the current SoTA algorithm treats the quantized integer as an intermediate result, while the final output of the quantizer is the "discretized" floating-point values, ignoring the practical needs and adding additional workload to hardware designers for integer parameter extraction and layer fusion. Finally, the compression toolkits designed by the industry are constrained to their in-house product or a handful of algorithms. The limited degree of freedom in the current toolkit and the under-explored customization hinder the prototype ASIC or FPGA-based accelerator design. To resolve these challenges, we propose Torch2Chip, an open-sourced, fully customizable, and high-performance toolkit that supports the user-designed compression algorithm followed by automatic model fusion and parameter extraction. Torch2Chip incorporates the hierarchical design workflow, and the user-customized compression algorithm will be directly packed into the deployment-ready format for either prototype chip verification with either CNN or vision transformer (ViT). Furthermore, Torch2Chip covers a wide range of training methods to achieve high performance, from basic supervised learning to state-of-the-art (SoTA) lightweight self-supervised learning (SSL). Code is available at `https://github.com/SeoLabCornell/torch2chip`.

1 INTRODUCTION

Deep neural network compression has been developed as a critical recipe and almost mandatory step for energy-efficient deep learning. Starting from the early exploration of magnitude-based pruning (Han et al., 2016) and low-precision quantization (Zhou et al., 2016), a variety of pruning/quantization algorithms have been presented along the rapid evolution in DNN architectures, across CNNs (Liu et al., 2021; Yang et al., 2020), transformers (Yu et al., 2022; Li & Gu, 2023), self-supervised learning (Meng et al., 2023) models, etc., generating compact models while preserving high accuracy.

To largely reduce hardware storage, computation and energy, hardware designers are motivated to adopt the state-of-the-art (SoTA) pruning and quantization schemes for custom DNN hardware accelerator designs, including digital ASIC (Lee et al., 2019; Moon et al., 2023; Conti et al., 2023; Desoli et al., 2023), FPGA (Yang et al., 2019b; Meng et al., 2021), or analog compute-in-memory platforms (Zhang et al., 2023; Huang et al., 2023). Besides the customized accelerator design, industry-standard compression toolkits (Gorbachev et al., 2019; Siddegowda et al., 2022) are developed to support the off-the-shelf commercial hardware platforms for automated compression and deployment.

Although significant improvements have been achieved in DNN compression algorithms and energy-efficient hardware accelerators, the hardware designer faces unavoidable

[1]Department of Electrical and Computer Engineering, Cornell University, USA. Correspondence to: Jae-sun Seo <js3528@cornell.edu>.

Proceedings of the 5th MLSys Conference, Santa Clara, CA, USA, 2024.

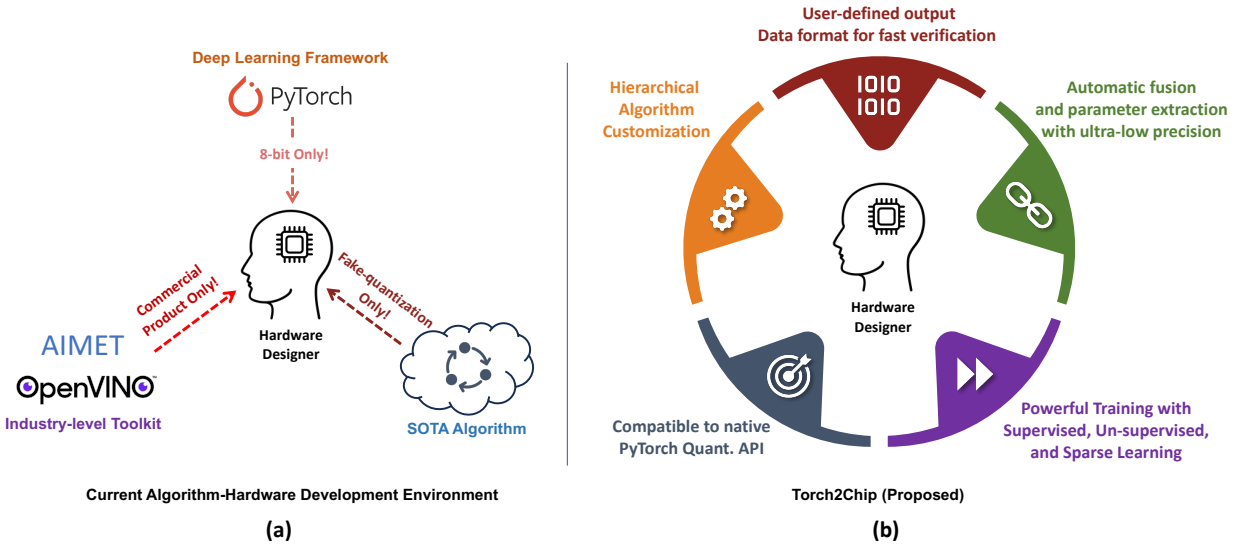

Figure 1. (a) Current algorithm-hardware development environment with disconnected workflow. (b) Proposed systematic toolkit `Torch2Chip` with high degree of customization versatility.

challenges in "compress-and-deploy" workflow due to the following reasons:

Gap between ML Framework and Hardware Designer. Motivated by the necessity of model compression, the popular deep learning (DL) framework (e.g., PyTorch (Paszke et al., 2019)) has released the dedicated compression API. However, the built-in quantization workflow is not the best candidate for hardware designers. In particular, the data precision is fixated as 8-bit with almost zero degree of customization. Although the compressed model is suitable for CPU or GPU-based acceleration, the pre-fixed, 8-bit only data format lags behind the ultra-low precision quantization algorithm and prototype hardware accelerators. The user-customized training method or post-training calibration scheme cannot be implemented directly inside the API for sub-8-bit precision. Meanwhile, prototype hardware accelerators use various precision and algorithm customizations (Lee et al., 2019; Moon et al., 2023). The non-customizable compression scheme of the DL framework is disconnected with the needs of hardware designers.

Gap between SoTA Algorithm and Hardware Designer. On the algorithm side, most of the recent quantization algorithms merge the "quantize-dequantize" together to ensure the correct gradient propagation (Li et al., 2021a; Xiao et al., 2023; Wei et al., 2022). As a result, the quantized low-precision integer is not the final output of the quantizer, and it cannot be accessed directly with the pre-trained model. As a result, fusing the normalization layer (Jacob et al., 2018) and extracting the quantized parameters requires additional algorithm manipulation from the hardware designer. Recent efforts on vision model quantization focus on efficient post-training quantization (PTQ) with noise-robust biasing (Liu et al., 2023) and efficient dequantization (Han et al., 2023). However, the compressed model is saved in full floating-point precision in the PyTorch model file, and the compatibility of further hardware deployment is ignored by default. Therefore, the SoTA quantization algorithms and hardware designers are disconnected due to the absence of the systematic "customize-and-deploy" workflow.

Gap between Industry-level Toolkit and Prototyping. In addition to the native compression workflow of the DL frameworks, industry-standard hardware-algorithm co-design toolkits (Gorbachev et al., 2019; Siddegowda et al., 2022) have been recently proposed for compression with the customized algorithm. However, almost all of the industry-level toolkits are designed for the dedicated commercial product (e.g., CPU, Neural Compute Stick (Xu et al., 2017)) rather than customized chips. Specifically, the low-level RTL verification requires properly unrolled weight tensors with hexadecimal or binary data format in the memory module. With the fixed data precision (Gorbachev et al., 2019) and limited quantization options (Siddegowda et al., 2022), the industry-standard toolkits are sub-optimal for custom hardware that aim to adopt SoTA quantization algorithms.

From the perspectives of the hardware designers, the conflicts from the DL framework, SoTA algorithm, and current toolkits formulate the cumbersome designation workflow of chip prototyping, as shown in Figure 1(a). Motivated by that, the following question arises:

How to automate the deployment process from model com-

pression to the prototype chip deployment, where the automatic workflow supports the fully customized SoTA compression method and training scheme?

We attempt to answer the above question from the following perspectives that are embedded into the proposed `Torch2Chip` toolkit:

1) Hierarchical customized quantization build-up. Most open-sourced quantization algorithms (Li et al., 2021b; Liu et al., 2022a) perform "quantization-dequantization" all at once in a **single** computation path for both training or inference, and it is difficult to separate the integer-only computation from the custom quantizer. In this work, `Torch2Chip` separates the training and inference path with separate computation graphs, to perform the fake-quantized computation for training and quantized integer-only operation for inference. The proposed "Dual-Path" design is employed as the bottom-level logic for `Base quantizer` and `Base layers`, which avoids interference between training and inference while keeping the freedom of customization in training. On top of that, the user-defined algorithms are constructed in a hierarchical fashion by computing the quantization parameter properly for either quantization-aware training (QAT) or post-training quantization (PTQ). In other words, `Torch2Chip` **only** requires users to design the training path and update the quantization parameter properly, while the remaining conversion will be completed automatically and executed in the inference path.

2) Automatic fusion and parameter extraction for ultra-low precision. BatchNorm (BN) fusion (Jacob et al., 2018) has been employed as a mandatory step for the post-training process in various toolkits. Normalization parameters are assumed to be fused into weights before quantization. However, prior works have shown the instability and degraded performance caused by the BN fusion with ultra-low-precision quantization ($\leq$8-bit) (Park & Yoo, 2020). As a result, BN parameters have to be fused into channel-wise scaling and shifting. Unfortunately, the channel-wise fusion is **not** fully supported by PyTorch (Paszke et al., 2019), even though it has been employed in the low-precision accelerator design (Meng et al., 2021; Zhao et al., 2019). Motivated by that, `Torch2Chip` automates the fusion and parameter extraction in both weight-based BN fusion (for 8-bit) and channel-wise scaling (for sub-8-bit).

3) Unsupervised foundation model training for powerful model compression and deployment. In addition to the highly versatile post-compression conversion and extraction, `Torch2Chip` also incorporates a wide range of training methods. Recent works on self-supervised learning (Zbontar et al., 2021; Bardes et al., 2022; Meng et al., 2023) have demonstrated stronger transfer learning performance than supervised learning. The learned superior visual representation becomes a powerful foundation for downstream vision tasks, which widely exists in resource-constrained edge computing. Different from the industry-level toolkit (Gorbachev et al., 2019; Siddegowda et al., 2022) with supervised model trainer only, `Torch2Chip` provides the SoTA contrastive self-supervised learning method (Meng et al., 2023) that is designed for lightweight (e.g., MobileNet) encoders. In particular, the 8-bit, deployment-ready MobileNet-V1 can achieve 94.37% and 74.29% accuracy on CIFAR-10 and CIFAR-100 datasets, outperforming the conventional supervised training by a significant margin.

4) User-defined output data format for fast deployment Different from the conventional integer data format in PyTorch (`torch.qint8`) or Numpy (`np.int8`), hardware description language (HDL) such as Verilog and SystemVerilog only supports the raw Hexadecimal or Binary values. Ignoring the data format mismatch introduces additional conversion effort to the verification process of the prototyped chip. In this work, `Torch2Chip` exports the model with various formats, including raw integer tensors in PyTorch output file, hardware-readable Hexadecimal raw data, and the PyTorch native integer format (`torch.qint8`).

2 Related Work

2.1 Quantization

Quantization has been widely investigated together with the evolution of deep learning. Starting from the early ResNet (He et al., 2016) all the way to the recent large language models (Narayanan et al., 2021), the elevation of the model complexity keeps necessitating efficient compression algorithms. To that end, various quantization methods (Nagel et al., 2020; Li et al., 2021a; Wei et al., 2022; Xiao et al., 2023) have been proposed together with the dedicated prototype accelerator (Yin et al., 2020; He et al., 2020; Wang et al., 2023) among different hardware platforms. The core computation of quantization can be generalized as a) Data format scaling, b) Low precision computation, and c) Output rescaling. Mathematically, given the weights $\mathcal{W}^l$ and input activation $\mathcal{X}^l$ of a layer l, the quantization process can be expressed as:

$$\mathcal{W}_Q^l = \left\lfloor \frac{\mathcal{W}^l}{S_w^l} \right\rceil \in \texttt{INT}_{\mathbf{n}}, \qquad \mathcal{X}_Q^l = \left\lfloor \frac{\mathcal{X}^l}{S_x^l} \right\rceil \in \texttt{INT}_{\mathbf{n}} \quad (1)$$

Where S is the scaling factor determined by data precision $\mathbf{n}$ and the numerical range. Apparently, the integer-based data cannot be directly inserted back for training or optimization. In practice, the integer values will be scaled back to the floating-point domain as the "dequantized (DQ)" values

$\mathcal{W}^l_{\text{DQ}}$ and $\mathcal{X}^l_{\text{DQ}}$:

$$\mathcal{W}^l_{\text{DQ}} = \mathcal{W}^l_Q \times S^l_w \in \texttt{Float}, \quad \mathcal{X}^l_{\text{DQ}} = \mathcal{X}^l_Q \times S^l_x \in \texttt{Float} \tag{2}$$

In some cases, the integer-based zero point $\mathcal{Z}$ will also be part of the compression to shift the data range to formulate the signed or unsigned data, which is omitted in Eq. (2) for simplicity.

Practically, the compressed weights and activation will be deployed to hardware with the low precision values only, while most quantization algorithms merge the quantize-and-dequantize altogether to ensure smooth gradient propagation. With the common convolution operation, we have:

$$\text{Algorithm: } \mathcal{Y}^l = \mathcal{W}^l_{\text{DQ}} \circledast \mathcal{X}^l_{\text{DQ}} \tag{3}$$

$$\text{Hardware: } \mathcal{Y}^l = (S^l_w S^l_x) \underbrace{\left(\mathcal{W}^l_{\text{Q}} \circledast \mathcal{X}^l_{\text{Q}}\right)}_{\text{Hardware Compatible}} \tag{4}$$

As shown in Eq. (3) and Eq. (4), the model pre-trained by the quantization algorithm cannot be deployed directly to hardware since the dequantized output is kept in high floating-point precision format. Starting from the early exploration in `TensorFlow Lite` (Jacob et al., 2018) to the recent industry-level toolkit (Siddegowda et al., 2022; Gorbachev et al., 2019), prior works have attempted to automate the process from Eq. (3) to Eq. (4) with 8-bit precision, followed by parameter extraction or deployment to hardware.

Meanwhile, the recent quantization algorithm mainly focuses on Eq. (3), where $\mathcal{W}^l_Q$ and $\mathcal{X}^l_Q$ can be compressed down to ultra-low precision (Xiao et al., 2023; Zhang et al., 2022b), power-of-two representation (Li et al., 2020), mixed precision with post-training quantization (PTQ) (Shang et al., 2023; Tu et al., 2023) or quantization-aware training (QAT) (Liu et al., 2022b; Yang et al., 2019a). Among all the recent algorithms, the dequantized $\mathcal{X}^l_{\text{DQ}}$ and $\mathcal{W}^l_{\text{DQ}}$ is essentially the "discretized" version of the original floating-point precision distribution. The hardware-readable $\mathcal{W}^l_{\text{Q}}$ and $\mathcal{X}^l_{\text{Q}}$ are treated as the intermediate results and **omitted** by the quantizer with in-place operation (Li et al., 2021a; Liu et al., 2022b):

```
# X_Q = x_quant; X_DQ = x_dequant
x_dequant = x_quant * scale
return x_dequant
```

With the model pre-trained by the quantization algorithms in PyTorch, **neither** $\mathcal{X}^l_{\text{Q}}$ or $\mathcal{W}^l_{\text{Q}}$ can be directly deployed for RTL-level verification. As a result, exploiting the benefits of the SoTA software-level quantization algorithm requires additional effort and conversion by the hardware designer.

2.2 Weight Sparsification

Exploring sparsity for storage reduction and computation skipping has been widely investigated with various models, tasks, and granularities. In general, sparse training works can be categorized based on 1) the starting point of sparsification and 2) the granularity of sparsity. Specifically, post-training sparsification (Dettmers & Zettlemoyer, 2019; Evci et al., 2020; Liu et al., 2021; Yuan et al., 2021) removes the weights and then fine-tunes the model to recover the accuracy loss. Meanwhile, before-training (Lee et al., 2018; Wang et al., 2019) sparsification creates a sparse model prior to the main training procedure. Regarding the granularity, element-wise pruning (Han et al., 2016) eliminates the individual weight value, while the recent works gradually expanded the spectrum, including hierarchical sparsity (Kadetotad et al., 2020; Wu et al., 2023), hardware-specific structured sparsity (Chu et al., 2020), filter-wise sparsity (Shen et al., 2022), and structured fine-grained sparsity (Zhou et al., 2021; Zhang et al., 2022a). The pruning candidate under different granularities can be generated based on magnitude score (Liu et al., 2021; Lee et al., 2018), gradient score (Liu et al., 2021; Yuan et al., 2021), or evaluating the impact between adjacent layers or channels (Park* et al., 2020).

Despite the variety of granularity and pruning methods, the ultimate objective of sparsification is reducing weight storage and skipping the computation corresponding to the sparse weights and accelerating the on-device computation of hardware. Motivated by that, various accelerator designs have been proposed on ASIC (Lee et al., 2019; Moon et al., 2023; Conti et al., 2023; Desoli et al., 2023) or FPGA (Meng et al., 2021; Cai et al., 2023) implementation with various architectures. Although the current SoTA pruning algorithm can achieve up to 98% element-wise sparsity (Yuan et al., 2021; Liu et al., 2021), most of the algorithms represent the sparsity by binary masks that are applied the **full-precision weights**. Given the persistent need to deploy low-precision models to prototype accelerators, isolating sparsification from quantization leads to limited practical benefits to hardware designers.

2.3 Self-supervised Pre-training

In addition to the success of supervised deep learning, recent research works have demonstrated the strong visual representation learned via self-supervised learning (SSL) (Zbontar et al., 2021; Grill et al., 2020; Bardes et al., 2022). In particular, the SSL-trained model outperforms the supervised learning counterpart in various downstream tasks. Starting from the early research with similarity-based sample alignment (Chen et al., 2020) to the recent image reconstruction with masked input (He et al., 2022), SSL is encouraging the model to learn the visual representation rather than

the supervised label-matching. As a result, the pre-trained encoder achieves superior performance in the small-scale downstream vision tasks (e.g., CIFAR, Flower), and outperforms the supervised learning counterpart.

Compared to supervised training from scratch, the strong and versatile vision model pre-trained by SSL provides an alternative candidate for model compression and hardware deployment. Due to the elevated baseline performance, the SSL-trained model exhibits a stronger "efficiency-accuracy" tradeoff in the small-scale vision tasks. However, compressing the SSL-trained foundation vision model has not been considered as a candidate for compression or deployment in the mainstream deployment toolkits yet.

3 TORCH2CHIP

3.1 Hierarchical Design for Customized Quantization

Different from the non-deployable computation path of most quantization algorithms, `Torch2Chip` employs the "Dual-Path" design, which separates the training and inference computation graph to support both floating-point-based "fake quantization" for training, and "low precision-only" computation for hardware verification and parameter extraction. Specifically, given the full precision weights `wfp` and the dequantized weights `wdq`, the training path of the `Base` quantizer is characterized as:

```
# training path
wq = wfp.div(scale).round()
wdq = (wq - wfp).detach() + wfp
self.scale.copy_(scale)
return wdq
```

While the inference path focuses on low-precision parameters only which will be saved as the final state of the model:

```
# inference path
wq = wfp.div(self.scale).round()
return wq
```

The "Dual-Path"-based computation scheme is designed as the **bottom-level logic** of the proposed `Torch2Chip` toolkit. On top of that, we define the `Base Module` and `Base Layer` as the starting point of the user-defined quantization method and computation, respectively. In particular, the customized quantization methods (weight or activation) are constructed based on `_QBase`, which holds the scaling factor and zero point as the registered parameters updated by the user-customized method in the training path. Subsequently, the customized quantizers are embedded into the second-level layer module to compress weights and activation. Following the same "Dual-Path" logic, the base layer modules also split the computation into inference and training paths separately to perform the same type of computation (e.g., convolution) with different data types (e.g., Integer or Float), as shown in Figure 2.

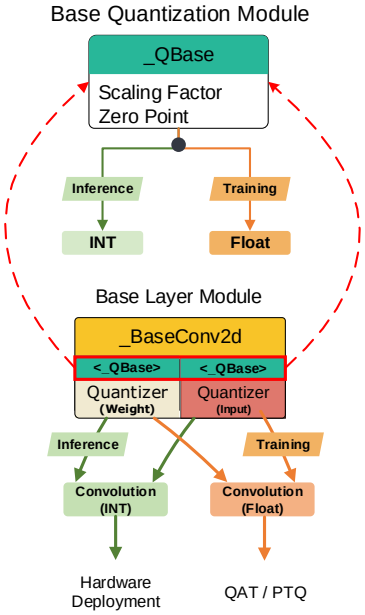

Figure 2. Hierarchical architecture of layer and module with the "Dual-Path"-based design.

The proposed hierarchical workflow shows pragmatic benefits for algorithm customization and deployment. First, the independent training and inference paths can be switched globally on the top-level training or evaluation workflow. The separate computation graph ensures the training process will not be interfered by the integer-only non-differentiable computation. Secondly, the proposed hierarchical customization provided a generic and versatile template for customization. Since all the quantization parameters are registered inside the bottom-level module `_QBase`, any kind of customized quantization can be converted correctly into low-precision only computation with the user-customized scaling factor and zero points.

`Torch2Chip` is also perfectly compatible with the recent adaptive quantization methods (Yang et al., 2019a; Nagel et al., 2020; Liu et al., 2023), while PyTorch (Paszke et al., 2019) and OpenViNO (Gorbachev et al., 2019) mainly incorporate the traditional nearest or stochastic rounding as the non-customizable option. For instance, AdaRound (Nagel et al., 2020) learns the rounding threshold via a differentiable non-linear function h, where the integer weights are computed by adding the learnable offset α:

$$\text{Training: } \mathcal{W}_Q^l = \left\lfloor \frac{\mathcal{W}^l}{S_w^l} \right\rfloor + h(\alpha) \tag{5}$$

$$\text{Inference: } \mathcal{W}_Q^l = \left\lfloor \frac{\mathcal{W}^l}{S_w^l} \right\rfloor + \{\mathbf{1}|\alpha \geq 0\} \tag{6}$$

To that end, the conventional scaling-based quantization in Eq. (1) is not sufficient, and AdaRound (Nagel et al., 2020) is not directly compatible with PyTorch Quantization.

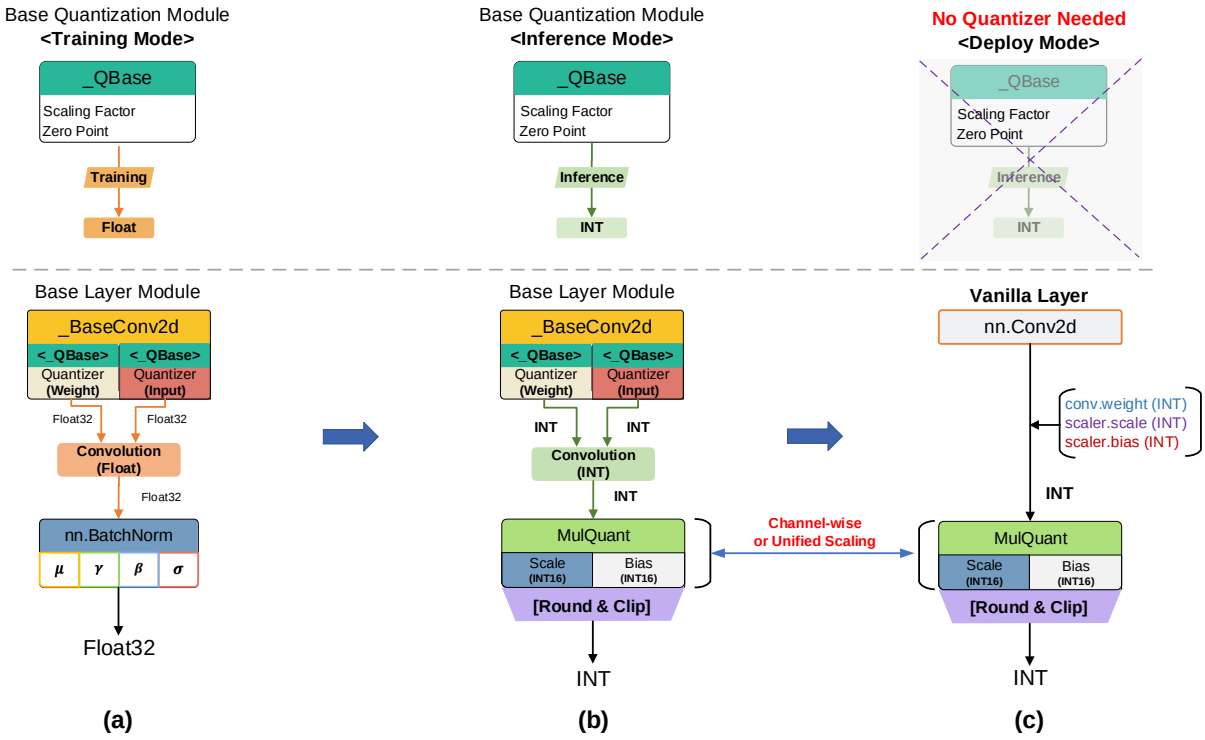

Figure 3. Training and fusion workflow of `Torch2Chip`: (a) training mode with fully customizable quantizer and layer, (b) inference mode with integer-only computation, and (c) deploy mode with integer-only parameters stored by the vanilla layer.

Instead, `Torch2Chip` only focuses on the quantized parameter rather than the rounding process. The low-precision weights $\mathcal{W}_Q^l$ will be registered as an additional parameter inside the base layer for inference only, while the differentiable soft offset ($h(\alpha)$) will be added during the training path of `Torch2Chip`. Although the training and inference are separately executed in `Torch2Chip`, users only need to implement the customized method in the training path, and the remaining steps will be executed **automatically** by `Torch2Chip`.

3.2 Fully Customizable Post-training Fusion

3.2.1 Fusing Quantizers and Normalization Layer

As one of the most important components for deep neural network training, normalization layers (e.g., BatchNorm, LayerNorm) stabilize the training by scaling and shifting the distribution of the prior layer output, while separating the channel-wise difference with the learnable weight and bias:

$$\bar{\mathcal{Y}}^l = \gamma \times \frac{\mathcal{Y}^l - \mu_s}{\sqrt{\sigma_s^2 + \epsilon}} + \beta \tag{7}$$

Where μ_s and σ_s^2 represent the mean and variance of $\mathcal{Y}^l$ along a certain dimension, and γ and β represent the learnable weight and bias of the normalization layer. The orientation of μ_s and σ_s^2 characterizes Eq. (7) into different types of normalization methods (Ioffe & Szegedy, 2015; Ulyanov et al., 2016; Ba et al., 2016; Wu & He, 2018).

The common approach to compressing the normalization is fusing the parameter into weights prior to the convolution (or matrix multiplication) and quantizing the resultant tensor altogether. Given the full precision weights W^l, such "Pre-Fusing" scheme can be represented as:

$$\mathcal{W}_{\text{Fuse}} = \frac{\gamma \mathcal{W}}{\sqrt{\sigma^2 + \epsilon}} \tag{8}$$

$$\mathcal{W}_Q = \left\lfloor \mathcal{W}_{\text{Fuse}} / S_w^l \right\rceil \tag{9}$$

The normalization bias β will be rearranged as the new bias term inside the layer (convolution or linear):

$$\bar{\mathcal{Y}}^l = \mathcal{Y}^l + \beta^* \tag{10}$$

$$\text{where} \quad \beta^* = \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{11}$$

Although the "Pre-Fusing" (Jacob et al., 2018; Paszke et al., 2019) scheme has been adopted as the mainstream fusing strategy in the ML framework and toolkits, the normalization parameters are sensitive to low precision representation. In particular, the success of pre-fusion is built upon the 8-bit quantization, which has relatively high precision compared to the recent ultra-low precision algorithms (<8-bit). With the low-precision quantization, the "Pre-Fusion" scheme exhibits a high degree of instability and large degraded accuracy, as reported in both algorithm (Park & Yoo, 2020) and hardware design (Meng et al., 2021).

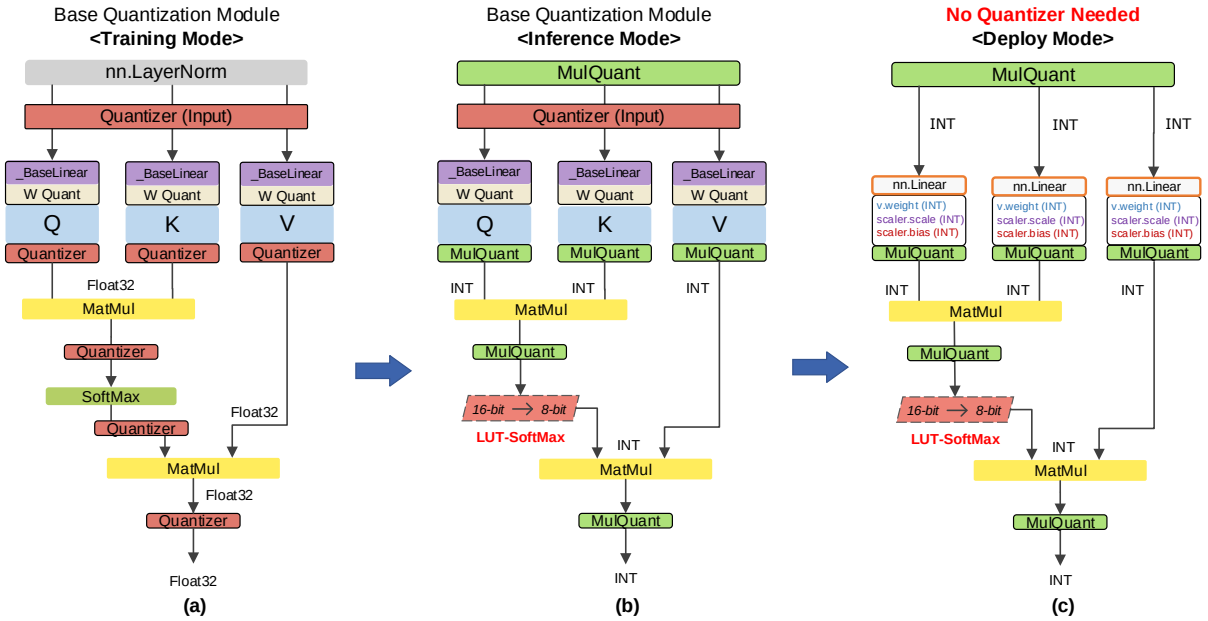

Figure 4. Training and fusion workflow of `Torch2Chip` with multi-head attention under 8-bit precision: (a) training mode with fully customizable quantizer and layer, (b) inference mode with integer-only computation and Look-up-Table (LUT)-based Softmax function, and (c) deploy mode with integer-only parameters stored by the vanilla fully-connected layer.

Alternatively, the normalization can be reformulated as the scaling and bias with respect to the channels of $\mathcal{Y}^l$:

$$\bar{\mathcal{Y}}^l = \gamma^* \times \mathcal{Y}^l + \beta^* \tag{12}$$

$$\text{where} \quad \gamma^* = \frac{\gamma}{\sqrt{\sigma_s^2 + \epsilon}} \tag{13}$$

Combining the fusion schemes in Eq. (10) and Eq. (12) with the quantization scaling in Eq. (4), we have:

$$\text{8-bit:} \quad \mathcal{Y}_Q^l = \left\lfloor \frac{S_w^l S_x^l}{S_x^{l+1}} \left(\mathcal{W}_\text{Q}^l \mathcal{X}_\text{Q}^l \right) + \frac{\beta^*}{S_x^{l+1}} \right\rceil \tag{14}$$

$$\text{Sub 8-bit:} \quad \mathcal{Y}_Q^l = \left\lfloor \gamma^* \times \frac{S_w^l S_x^l}{S_x^{l+1}} \left(\mathcal{W}_\text{Q}^l \mathcal{X}_\text{Q}^l \right) + \frac{\beta^*}{S_x^{l+1}} \right\rceil \tag{15}$$

As shown in Eq. (14) and Eq. (15), both scenarios can be formulated as integer-only convolution followed by scaling and then adding bias. The major separation is the channel-wise scaling or unified scaling. Different from the mainstream PyTorch and industry-standard toolkits that only support the unified scaling, Pre-Fusing, and 8-bit precision, `Torch2Chip` incorporates **both** scaling scheme and supports the model fusion with respect to conventional 8-bit precision and the sub-8-bit quantization.

Unlike Pytorch (Paszke et al., 2019) that keeps the unified scaling factor as high floating-point precision tensors, `Torch2Chip` automatically fuses the normalization module with the quantizer as the scaling and shifting into the high precision integer stored inside the `MulQuant` module with the user-defined integer and fractional precision. The "Dual-Path" computation graph and the automatic fusion logic ensure the user-defined quantization is deployable with integer-only parameters, as shown in Figure 3 (a) (b).

3.2.2 *Integer-only Vision Transformer*

In addition to the automated compression and deployment on CNN, `Torch2Chip` is also compatible with the compression on Vision Transformer (ViT). Following the proposed "Dual-Path" computation graph and hierarchical customization, the multi-head attention and transformer block are compressed and fused automatically for integer-only inference and parameter extraction, as shown in Figure 4(a) and Figure 4(b).

Different from the recent I-ViT (Li & Gu, 2023), which estimates the SoftMax function with shift-and-accumulation, `Torch2Chip` is equipped with Look-up-Table (LUT)-based non-linear function approximation (e.g., SoftMax, GeLU), which can be customized by users. The LUT-based approximation can enable an efficient hardware approximation. The non-linear function approximation and the accuracy impact have been largely ignored in the quantization of the mainstream DL frameworks (Paszke et al., 2019; Jacob et al., 2018) with the naïve full precision computation.

For the vision transformer with LayerNorm, μ and σ are computed on the fly along the channel dimension. The serialized summation and averaging increase the latency

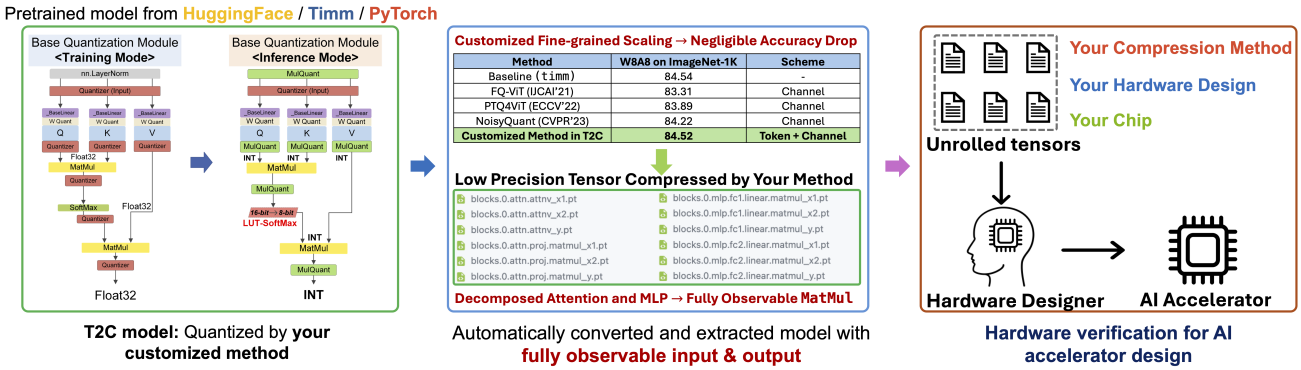

Figure 5. Automated and versatile parameter extraction with various output formats for RTL verification, integer-only model file, and native Pytorch quantization layer

compared to the pre-computed running statistics in BatchNorm. In `Torch2Chip`, LayerNorm is customizable with both instant statistics and pre-computed running statistics for different energy and latency requirements.

3.3 Compressing Powerful Foundation Vision Model

Besides the versatile post-compression fusion and parameter extraction, `Torch2Chip` incorporates both supervised and self-supervised trainers for quantization (QAT and PTQ) and powerful visual representation learning, respectively.

3.3.1 Self-supervised Pre-training

In addition to the supervised learning compression, recent works have demonstrated the strong transfer learning performance of self-supervised learning (SSL) (Bardes et al., 2022; Zbontar et al., 2021; Grill et al., 2020). With minimal fine-tuning, the SSL-trained backbone model outperforms the supervised learning (fine-tuning) counterpart in various small-scale computer vision tasks. `Torch2Chip` incorporates the SSL in the pre-training phase and employs the powerful "foundation model" for subsequent compression and parameter extraction.

In this work, we adopt the recent SoTA lightweight contrastive learning algorithm (Meng et al., 2023) as the pre-training method. The model is trained by applying the cross-distillation (XD) on top of the correlation-based contrastive learning loss (Zbontar et al., 2021). Given the augmented input images $(A, \tilde{A})$, the cross-distillation is formulated as:

$$\mathcal{L}^{A}_{\text{XD}} = \sum_{i}(1 - C^{A\tilde{A}}_{ii}) + \lambda \sum_{i} \sum_{i \neq j} (C^{A\tilde{A}}_{ij})^2 \qquad (16)$$

Where C_{ij} represents the correlation between the different dimensions of the latent output $(z, \tilde{z})$ encoded from $(A, \tilde{A})$ with the neural network model. Together with the Barlow Loss (Zbontar et al., 2021), the DNN model is trained to learn a robust visual representation from the input, which further leads to strong transfer learning performance for various small-scale vision tasks. Unlike other model compression toolkits (Gorbachev et al., 2019; Siddegowda et al., 2022) that employ supervised training only, `Torch2Chip` incorporates self-supervised learning as an alternative and high-performance pre-training option to users. The pre-trained model can be further fine-tuned and compressed in the downstream tasks to facilitate the performance. All the training and fine-tuning schemes are customizable in the `TRAINER` selection that is introduced in Section 3.4.

3.4 Automated and Customized Parameter Extraction

On top of all the automatic layer-wise fusion, the final phase of `Torch2Chip` is converting the user-customized low-precision model into the integer-only representation with the native vanilla layers of PyTorch (Paszke et al., 2019).

Since all the operations are converted to integer-based operations after the automated fusion, `Torch2Chip` removes the user-customized quantizer and assigns the low-precision integer weights to the vanilla layers of Pytorch. The scaling and normalization are fused into the `MulQuant`, as shown in Figure 3(c) and Figure 4(c). By doing so, the vanilla models are customized and compressed into the low-precision model while keeping the same architecture as the original model. Instead of saving the full-precision weight parameters and executing the quantization on the fly, our "*vanilla-custom-vanilla*" scheme leads to multiple benefits for the subsequent hardware deployment:

- Keeping the vanilla model architecture as the output of `Torch2Chip` ensures the compressed parameters are saved in the same file format as the original model. As a result, `Torch2Chip` guarantees the "real compression" with generic modules of PyTorch.
- The integer-only model file is compatible with various data formats, including both decimal and hexadecimal representation. From the perspective of prototyping, the customized compression methods are deployable to hardware in a simple and automated manner.
- Different from other open-sourced quantization algo-

Table 1. ImageNet-1K accuracy comparison between `Torch2Chip` and the recent DNN deployment toolkits.

Toolkit	Method	Training Method	Model	W/A (bit)	Scale and Bias (INT, Frac)	Accuracy (%)	Customizable
AIMET (Qualcomm) (Siddegowda et al., 2022)	AdaRound (Nagel et al., 2020)	PTQ	ResNet-50	8/8	Float	75.45 (-0.55)	✗
OpenVINO (Intel) (Gorbachev et al., 2019)	MinMax Quant.	PTQ	ResNet-50	8/8	Float	75.98 (0.02)	✗
Torch2Chip (Ours)	QDROP (Wei et al., 2022)	PTQ	ResNet-50	**4/4**	**INT (12, 4)**	74.40 (-1.60)	✓
Torch2Chip (Ours)	QDROP (Wei et al., 2022)	PTQ	ResNet-50	8/8	**INT (12, 4)**	**75.96 (-0.04)**	✓

Table 2. CIFAR-10 accuracy comparison with the integer-only DNN model compressed by `Torch2Chip` with customized quantization.

Method	Model	Training Method	W/A (bit)	# of Param. (M)	Scale and Bias (INT, Frac)	Accuracy (%)	Model Size (MB)
SAWB + PACT (Choi et al., 2019)	ResNet-20	QAT	**2/2**	0.27	INT (13, 3)	90.22 (-1.17)	0.07
SAWB + PACT (Choi et al., 2019)	ResNet-20	QAT	**4/4**	0.27	INT (13, 3)	91.24 (-0.73)	0.14
RCF (Li et al., 2020)	ResNet-18	QAT	**4/4**	11.17	INT (12, 4)	94.56 (-0.21)	5.59
RCF (Li et al., 2020)	ResNet-18	QAT	8/8	11.17	INT (12, 4)	94.77 (-0.01)	11.17
RCF (Li et al., 2020)	ViT-7	QAT	8/8	6.3	INT (13, 3)	89.63 (-0.02)	6.30
PROFIT (Park & Yoo, 2020)	MobileNet-V1	QAT	**4/4**	4.2	INT (12, 4)	89.42 (-0.35)	2.10
PROFIT (Park & Yoo, 2020)	MobileNet-V1	QAT	8/8	4.2	INT (12, 4)	89.73 (-0.01)	4.20
AdaRound (Nagel et al., 2020)	MobileNet-V1	PTQ	8/8	4.2	INT (12, 4)	89.57 (-0.17)	2.10
PyTorch Quant. (Paszke et al., 2019)	MobileNet-V1	PTQ	8/8	4.2	Float32	89.34 (-0.40)	4.20

rithms, the vanilla layer and the `MulQuant` layers can be packed into the native quantization layer of PyTorch (Paszke et al., 2019), which will be saved as the default `torch.qint` format.

At the top level, `Torch2Chip` automates the entire compression workflow with a simple and elegant execution process consisting of **five lines of code only**:

```
model = ...
# Define the trainer
trainer = TRAINER[user_select](args)
# Start training / fine-tuning
trainer.fit()
# Initiate T2C
nn2c = T2C(model, fuser=NetFuser)
# Save model
qnn = nn2c.nn2chip(save_model=True)
```

`Torch2Chip` covers a wide spectrum of training schemes for different compression algorithms (`TRAINER`), including supervised QAT, PTQ, sparse training, and self-supervised foundation vision model training. The detailed settings will be elaborated in the official document of `Torch2Chip`.

Overall, `Torch2Chip` successfully merges the gap between the customized SoTA algorithm, DL framework, and RTL-based chip prototyping while minimizing the additional programming effort of hardware designers. Figure 5 shows the high versatility of `Torch2Chip` with various export formats.

4 Experimental Results

This section reports the comprehensive performance evaluation of the proposed `Torch2Chip`. We evaluate the proposed toolkit with ResNet (He et al., 2016), MobileNet (Howard et al., 2017), and Vision Transformer (ViT) (Dosovitskiy et al., 2020) against various compression methods. We employ multiple recent quantization and sparsification algorithms based on the proposed workflow to demonstrate the high degree of customization.

4.1 Integer-only DNN on ImageNet-1K Dataset

`Torch2Chip` is also compatible with the large-scale ImageNet dataset. In addition to the re-implementation of the SoTA quantization algorithms based on `Torch2Chip`, we evaluate the versatility of `Torch2Chip` by comparing it with the native PyTorch quantization API (Paszke et al., 2019) and the recent industry-standard toolkits (Siddegowda et al., 2022; Gorbachev et al., 2019). Table 1 summarizes the PTQ performance on ResNet-18 compressed by `Torch2Chip` on the ImageNet-1K dataset with 8-bit and 4-bit model precision quantized by QDrop (Wei et al., 2022) under the workflow of `Torch2Chip`. The fully customizable and observable compression workflow enables the

Table 3. Sparse and low precision ResNet-50 compressed by `Torch2Chip` with customized methods on ImageNet-1K dataset.

Method	W/A (bit)	Quantization	Model	W/A (bit)	Sparsity (%)	Accuracy (%)
GraNet (Liu et al., 2021)	8/8	PTQ	ResNet-50	8/8	80%	75.15 (-0.85%)
GraNet (Liu et al., 2021)	4/4	PTQ	ResNet-50	4/4	80%	73.38 (-2.62%)
N:M = 2:4 (Zhou et al., 2021)	8/8	PTQ	ResNet-50	8/8	50%	75.44 (-0.75%)
N:M = 2:4 (Zhou et al., 2021)	4/4	PTQ	ResNet-50	4/4	50%	74.16 (-1.84%)

Table 4. Transfer fine-tuning of MobileNet (Howard et al., 2017) pretrained by the proposed method.

Method	Encoder	W/A (bit)	CIFAR-10	CIFAR-100	Aircraft	Flowers	Food-101
Torch2Chip + Supervised Learning + PTQ	Mob-V1 (1×)	8/8	89.74	65.98	60.09	72.23	56.41
Torch2Chip + XD + PTQ (Meng et al., 2023)	Mob-V1 (1×)	8/8	**94.37**	**74.29**	**68.44**	**86.42**	**70.21**

freedom of employing high-performance algorithms with sub-8-bit precision while keeping the integer-only computation. The starting point of the PTQ is the full-precision ResNet-50 model pre-trained by PyTorch. The fused scaling factor and bias are quantized to INT16 fixed-point representation with 12 fractional bits and 4 integer bits.

Compared to other recent deployment toolkits (Siddegowda et al., 2022; Gorbachev et al., 2019), the objective of `Torc2Chip` is "customize a compressed model" rather than "getting a compressed model". The full stack of customization and the automated model conversion enables the hardware designers to implement the algorithm freely and deploy the model directly.

4.2 Integer-only DNN on CIFAR-10 Dataset

We also evaluate the compressed DNN model performance on the CIFAR-10 dataset and summarize the results in Table 2. For the QAT-based algorithms, the CNN models are trained by 200 epochs from scratch with SGD optimizer. The learning rate schedule and the required regularization are reimplemented based on the parameter settings reported in the original paper. Table 2 reports the optimal scaling precision with the best accuracy.

Besides the fully customizable compression scheme for ultra-low precision, `Torch2Chip` performs better than PyTorch quantization with pre-defined 8-bit precision. With the MobileNet-V1 model, `Torch2Chip` outperforms the PyTorch native quantization scheme without introducing any floating point operations, as presented in Section 4.

4.3 Sparse Training with User-defined Sparsity

Besides the quantization, `Torch2Chip` also allows users to exploit sparsity with the user-customized sparsification method. Starting from the basic pruner with element-wise sparsity (Han et al., 2016), `Torch2Chip` provides the customized example pruner with N: M sparsity (Zhou et al., 2021). Table 3 summarizes the performance of the sparse ResNet-50 pruned by Torch2Chip with different granularity. Starting from scratch, the dense model is pruned with gradually increased sparsity, and the post-training quantization is applied to compress the full-precision sparse model into integer-only representation. Instead of simply representing the sparsity as an additional mask, the sparsified and converted models save the pruned parameter as the raw zero values in the resultant integer model, which can be saved and exported directly to the user-defined file format.

4.4 Compressed Transfer Learning based on Self-supervised Pre-training

Different from (Siddegowda et al., 2022; Gorbachev et al., 2019) with supervised pre-training only, `Torch2Chip` incorporates the powerful SSL-trainer with cross-distillation (XD) (Meng et al., 2023) to achieve the superior performance on the downstream small-scale vision tasks. Before quantizing the model, the full-precision MobileNet-V1 is pre-trained by the self-supervised `Trainer` on ImageNet-1K. The pre-trained foundation model is fine-tuned and compressed with the supervised `Trainer` on various downstream tasks.

Table 4 shows a transfer learning example of SSL-trained MobileNet-V1 compressed by `Torch2Chip` with 8-bit precision. Compared to supervised learning from scratch, the SSL-trained MobileNet-V1 achieves **94.37%** and **78.22%** CIFAR-10 and CIFAR-100 accuracy, which largely outperforms the supervised learning baseline with 4.63% and 12.2% accuracy improvements. The entire workflow of training, fine-tuning, and compression is fully customizable by users with different datasets and precision. The resultant high-performance models are automatically converted into integer-only computation, which can be exported to various formats as shown in Figure 5.

5 CONCLUSION

In this paper, we presented `Torch2Chip`, a fully customizable, fully observable development toolkit designed for prototype DNN hardware accelerator design. Unlike other works with limited customization or non-extractable parameters, our work enables end-to-end customization while automating post-compression fusion and parameter extraction with different output formats. `Torch2Chip` bridges the research gaps between algorithm, hardware designer, and DL frameworks and further enriches the training method with powerful self-supervised learning. The automated deployment process accelerates the design process of prototyping with ASIC or FPGA platforms. The open-sourced toolkit will be updated continuously to support full-stack customization with different deep-learning model deployment scenarios.

6 ACKNOWLEDGEMENT

This work was supported in part by Samsung Electronics and the Center for the Co-Design of Cognitive Systems (CoCoSys) in JUMP 2.0, a Semiconductor Research Corporation (SRC) Program sponsored by the Defense Advanced Research Projects Agency (DARPA).

REFERENCES

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bardes, A., Ponce, J., and Lecun, Y. Vicregl: Self-supervised learning of local visual features. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Cai, L., Wang, J., Yu, L., Yan, B., Tao, Y., and Yang, Y. Accelerating neural-ode inference on fpgas with two-stage structured pruning and history-based stepsize search. In *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 177–183, 2023.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A Simple Framework for Contrastive Learning of Visual Representations. In *International Conference on Machine Learning (ICML)*, 2020.

Choi, J., Venkataramani, S., Srinivasan, V. V., Gopalakrishnan, K., Wang, Z., and Chuang, P. Accurate and efficient 2-bit quantized neural networks. *Proceedings of Machine Learning and Systems*, 1:348–359, 2019.

Chu, C., Wang, Y., Zhao, Y., Ma, X., Ye, S., Hong, Y., Liang, X., Han, Y., and Jiang, L. Pim-prune: Fine-grain dcnn pruning for crossbar-based process-in-memory architecture. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Conti, F., Rossi, D., Paulin, G., Garofalo, A., Di Mauro, A., Rutishauer, G., Ottavi, G. m., Eggimann, M., Okuhara, H., Huard, V., Montfort, O., Jure, L., Exibard, N., Gouedo, P., Louvat, M., Botte, E., and Benini, L. A 12.4TOPS/W @ 136GOPS AI-IoT system-on-chip with 16 RISC-V, 2-to-8b precision-scalable DNN acceleration and 30 In *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 21–23, 2023. doi: 10.1109/ISSCC42615.2023.10067643.

Desoli, G., Chawla, N., Boesch, T., Avodhyawasi, M., Rawat, H., Chawla, H., Abhijith, V., Zambotti, P., Sharma, A., Cappetta, C., Rossi, M., De Vita, A., and Girardi, F. 16.7 a 40-310tops/w sram-based all-digital up to 4b in-memory computing multi-tiled nn accelerator in fd-soi 18nm for deep-learning edge applications. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 260–262, 2023. doi: 10.1109/ISSCC42615.2023.10067422.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning (ICML)*, 2020.

Gorbachev, Y., Fedorov, M., Slavutin, I., Tugarev, A., Fatekhov, M., and Tarkan, Y. Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

Han, W. K., Lee, B., Park, S. H., and Jin, K. H. Abcd: Arbitrary bitwise coefficient for de-quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5876–5885, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16000–16009, 2022.

He, W., Yin, S., Kim, Y., Sun, X., Kim, J.-J., Yu, S., and Seo, J.-S. 2-bit-per-cell rram-based in-memory computing for area-/energy-efficient deep learning. *IEEE Solid-State Circuits Letters*, 3:194–197, 2020.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Huang, W.-H., Wen, T.-H., Hung, J.-M., Khwa, W.-S., Lo, Y.-C., Jhang, C.-J., Hsu, H.-H., Chin, Y.-H., Chen, Y.-C., Lo, C.-C., Liu, R.-S., Tang, K.-T., Hsieh, C.-C., Chih, Y.-D., Chang, T.-Y., and Chang, M.-F. A nonvolatile AI-edge processor with 4MB SLC-MLC hybrid-mode ReRAM compute-in-memory macro and 51.4-251TOPS/W. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 15–17, 2023. doi: 10.1109/ISSCC42615.2023.10067610.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Kadetotad, D., Meng, J., Berisha, V., Chakrabarti, C., and Seo, J.-s. Compressing lstm networks with hierarchical coarse-grain sparsity. *Interspeech 2020*, 2020.

Lee, J., Kim, C., Kang, S., Shin, D., Kim, S., and Yoo, H.-J. UNPU: an energy-efficient deep neural network accelerator with fully variable weight bit precision. *IEEE Journal of Solid-State Circuits*, 54(1):173–185, 2019. doi: 10.1109/JSSC.2018.2865489.

Lee, N., Ajanthan, T., and Torr, P. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR)*, 2018.

Li, Y., Dong, X., and Wang, W. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2020.

Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., Yu, F., Wang, W., and Gu, S. {BRECQ}: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations (ICLR)*, 2021a.

Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., Yu, F., Wang, W., and Gu, S. Brecq: Pushing the limit of post-training quantization by block reconstruction. https://github.com/yhhhli/BRECQ, 2021b.

Li, Z. and Gu, Q. I-vit: Integer-only quantization for efficient vision transformer inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17065–17075, 2023.

Liu, S., Chen, T., Chen, X., Atashgahi, Z., Yin, L., Kou, H., Shen, L., Pechenizkiy, M., Wang, Z., and Mocanu, D. C. Sparse training via boosting pruning plasticity

with neuroregeneration. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Liu, Y., Yang, H., Dong, Z., Keutzer, K., Du, L., and Zhang, S. Noisyquant: Noisy bias-enhanced post-training activation quantization for vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20321–20330, 2023.

Liu, Z., Cheng, K.-T., Huang, D., Xing, E., and Shen, Z. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. https://github.com/liuzechun/Nonuniform-to-Uniform-Quantization, 2022a.

Liu, Z., Cheng, K.-T., Huang, D., Xing, E. P., and Shen, Z. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022b.

Meng, J., Venkataramanaiah, S. K., Zhou, C., Hansen, P., Whatmough, P., and Seo, J.-s. Fixyfpga: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021.

Meng, J., Yang, L., Kyungmin, L., Jinwoo, S., Deliang, F., and Jae-sun, S. Slimmed asymmetrical contrastive learning and cross distillation for lightweight model training. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Moon, S., Mun, H.-G., Son, H., and Sim, J.-Y. A 127.8tops/w arbitrarily quantized 1-to-8b scalable-precision accelerator for general-purpose deep learning with reduction of storage, logic and latency waste. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 21–23, 2023. doi: 10.1109/ISSCC42615.2023.10067615.

Nagel, M., Amjad, R. A., Van Baalen, M., Louizos, C., and Blankevoort, T. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, 2020.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.

Park, E. and Yoo, S. Profit: A novel training method for sub-4-bit mobilenet models. In *European Conference on Computer Vision (ECCV)*, 2020.

Park*, S., Lee*, J., Mo, S., and Shin, J. Lookahead: A far-sighted alternative of magnitude-based pruning. In *International Conference on Learning Representations (ICLR)*, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

Shang, Y., Yuan, Z., Xie, B., Wu, B., and Yan, Y. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (ICCV)*, 2023.

Shen, M., Yin, H., Molchanov, P., Mao, L., Liu, J., and Alvarez, J. M. Structural pruning via latency-saliency knapsack. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Siddegowda, S., Fournarakis, M., Nagel, M., Blankevoort, T., Patel, C., and Khobare, A. Neural network quantization with ai model efficiency toolkit (aimet). *arXiv preprint arXiv:2201.08442*, 2022.

Tu, Z., Hu, J., Chen, H., and Wang, Y. Toward accurate post-training quantization for image super resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations (ICLR)*, 2019.

Wang, Z., Nalla, P. S., Krishnan, G., Joshi, R. V., Cady, N. C., Fan, D., Seo, J.-s., and Cao, Y. Digital-assisted analog in-memory computing with rram devices. In *2023 International VLSI Symposium on Technology, Systems and Applications (VLSI-TSA/VLSI-DAT)*, 2023.

Wei, X., Gong, R., Li, Y., Liu, X., and Yu, F. Qdrop: Randomly dropping quantization for extremely low-bit post-training quantization. In *International Conference on Learning Representations (ICLR)*, 2022.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Wu, Y. N., Tsai, P.-A., Muralidharan, S., Parashar, A., Sze, V., and Emer, J. S. Highlight: Efficient and flexible dnn acceleration with hierarchical structured sparsity. *EEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*, 2023.

Xu, X., Amaro, J., Caulfield, S., Forembski, A., Falcao, G., and Moloney, D. Convolutional neural network on neural compute stick for voxelized point-clouds classification. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1–7. IEEE, 2017.

Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., and Hua, X.-s. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019a.

Yang, L., He, Z., and Fan, D. Harmonious coexistence of structured weight pruning and ternarization for deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Yang, Y., Huang, Q., Wu, B., Zhang, T., Ma, L., Gambardella, G., Blott, M., Lavagno, L., Vissers, K., Wawrzynek, J., and Keutzer, K. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded FPGAs. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 23–32, 2019b.

Yin, S., Jiang, Z., Seo, J.-S., and Seok, M. Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits*, 55(6):1733–1743, 2020.

Yu, S., Chen, T., Shen, J., Yuan, H., Tan, J., Yang, S., Liu, J., and Wang, Z. Unified visual transformer compression. In *International Conference on Learning Representations (ICLR)*, 2022.

Yuan, G., Ma, X., Niu, W., Li, Z., Kong, Z., Liu, N., Gong, Y., Zhan, Z., He, C., Jin, Q., et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. Barlow Twins: Self-supervised Learning via Redundancy Reduction. In *International Conference on Machine Learning (ICML)*, 2021.

Zhang, B., Yin, S., Kim, M., Saikia, J., Kwon, S., Myung, S., Kim, H., Kim, S. J., Seo, J.-S., and Seok, M. PIMCA: a programmable in-memory computing accelerator for energy-efficient dnn inference. *IEEE Journal of Solid-State Circuits*, 58(5):1436–1449, 2023. doi: 10.1109/JSSC.2022.3211290.

Zhang, Y., Lin, M., Lin, Z., Luo, Y., Li, K., Chao, F., Wu, Y., and Ji, R. Learning best combination for efficient n: M sparsity. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022a.

Zhang, Y., Zhang, Z., and Lew, L. Pokebnn: A binary pursuit of lightweight accuracy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022b.

Zhao, Y., Gao, X., Guo, X., Liu, J., Wang, E., Mullins, R., Cheung, P. Y., Constantinides, G., and Xu, C.-Z. Automatic generation of multi-precision multi-arithmetic cnn accelerators for fpgas. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019.

Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., and Li, H. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations (ICLR)*, 2021.

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.