BiBench: Benchmarking and Analyzing Network Binarization

Haotong Qin [*] [1] [2] Mingyuan Zhang [*] [3] Yifu Ding [1] Aoyu Li [3] Zhongang Cai [3]
Ziwei Liu [3] Fisher Yu [2] Xianglong Liu[✉] [1]

Abstract

Network binarization emerges as one of the most promising compression approaches offering extraordinary computation and memory savings by minimizing the bit-width. However, recent research has shown that applying existing binarization algorithms to diverse tasks, architectures, and hardware in realistic scenarios is still not straightforward. Common challenges of binarization, such as accuracy degradation and efficiency limitation, suggest that its attributes are not fully understood. To close this gap, we present **BiBench**, a rigorously designed benchmark with in-depth analysis for network binarization. We first carefully scrutinize the requirements of binarization in the actual production and define evaluation tracks and metrics for a comprehensive and fair investigation. Then, we evaluate and analyze a series of milestone binarization algorithms that function at the operator level and with extensive influence. Our benchmark reveals that 1) the binarized operator has a crucial impact on the performance and deployability of binarized networks; 2) the accuracy of binarization varies significantly across different learning tasks and neural architectures; 3) binarization has demonstrated promising efficiency potential on edge devices despite the limited hardware support. The results and analysis also lead to a promising paradigm for accurate and efficient binarization. We believe that BiBench will contribute to the broader adoption of binarization and serve as a foundation for future research. The code for our BiBench is released here.

1. Introduction

The rising of deep learning leads to the persistent contradiction between larger models and the limitations of deployment resources. Compression technologies have been widely studied to address this issue, including quantization (Gong et al., 2014; Wu et al., 2016; Vanhoucke et al., 2011; Gupta et al., 2015), pruning (Han et al., 2015; 2016; He et al., 2017), distillation (Hinton et al., 2015; Xu et al., 2018; Chen et al., 2018; Yim et al., 2017; Zagoruyko & Komodakis, 2017), lightweight architecture design (Howard et al., 2017; Sandler et al., 2018; Zhang et al., 2018b; Ma et al., 2018), and low-rank decomposition (Denton et al., 2014; Lebedev et al., 2015; Jaderberg et al., 2014; Lebedev & Lempitsky, 2016). These technologies are essential for the practical application of deep learning.

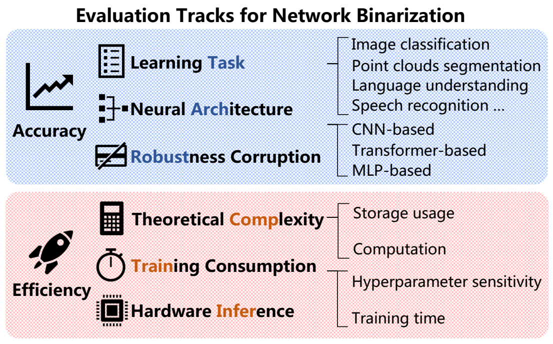
As a compression approach that reduces the bit-width to 1-bit, network binarization is regarded as the most aggressive quantization technology (Rusci et al., 2020; Choukroun et al., 2019; Qin et al., 2022a; Shang et al., 2022b; Zhang et al., 2022b; Bethge et al., 2020; 2019; Martinez et al., 2019; Helwegen et al., 2019). The binarized models take little storage and memory and accelerate the inference by efficient bitwise operations. Compared to other compression technologies like pruning and architecture design, network binarization has potent topological generics, as it only applies to parameters. As a result, it is widely studied in academic research as a standalone compression technique rather than just a 1-bit specialization of quantization (Gong et al., 2019; Gholami et al., 2021). Some state-of-the-art (SOTA) binarization algorithms have even achieved full-precision performance with binarized models on large-scale tasks (Deng et al., 2009; Liu et al., 2020).

However, existing network binarization is still far from practical, and two worrisome trends appear in current research:

***Trend-1*: Accuracy comparison scope is limited.** In recent research, several image classification tasks (such as CIFAR-10 and ImageNet) have become standard options for comparing accuracy among different binarization algorithms. While this helps to clearly and fairly compare accuracy performance, it causes most binarization algorithms to be only engineered for image inputs (2D visual modality), and their insights and conclusions are rarely verified in other modalities and tasks. The use of monotonic tasks also hinders a comprehensive evaluation from an architectural perspective. Furthermore, data noise, such as corruption (Hendrycks & Dietterich, 2018), is a common problem on low-cost edge devices and is widely studied in compression, but this situation is hardly considered existing binarization algorithms.

[*]Equal contribution [1]Beihang University [2]ETH Zürich [3]Nanyang Technological University. Correspondence to: ✉ Xianglong Liu <xlliu@buaa.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).



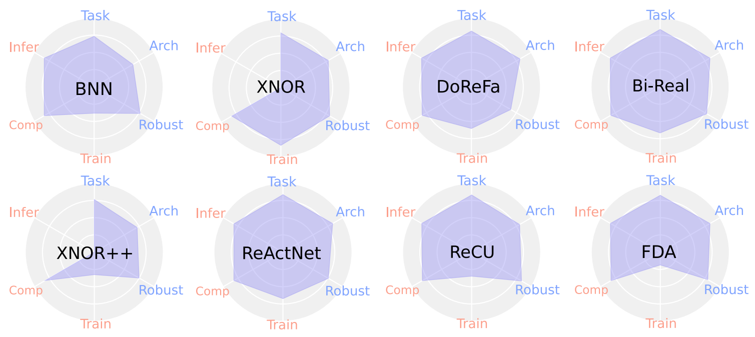


Figure 1: Evaluation tracks of BiBench. Our benchmark evaluates the performance of binarization algorithms on a range of comprehensive evaluation tracks, including: “Learning Task”, “Neural Architecture”, “Corruption Robustness”, “Training Consumption”, “Theoretical Complexity”, and “Hardware Inference”.

***Trend-2*: Efficiency analysis remains theoretical.** Network binarization is widely recognized for its significant storage and computation savings, with theoretical savings of up to 32× and 64× for convolutions, respectively (Rastegari et al., 2016; Bai et al., 2021). However, these efficiency claims lack experimental evidence due to the lack of hardware library support for deploying binarized models on real-world hardware. Additionally, the training efficiency of binarization algorithms is often ignored in current research, leading to negative phenomena during the training of binary networks, such as increased demand for computation resources and time consumption, sensitivity to hyperparameters, and the need for detailed optimization tuning.

This paper presents **BiBench**, a network **bin**arization **bench**mark designed to evaluate binarization algorithms comprehensively in terms of accuracy and efficiency (Table 1). Using BiBench, we select 8 representative binarization algorithms that are extensively influential and function at the operator level (details and selection criteria are in Appendix A.1) and benchmark algorithms on 9 deep learning datasets, 13 neural architectures, 2 deployment libraries, 14 hardware chips, and various hyperparameter settings. We invested approximately 4 GPU years of computation time in creating BiBench, intending to promote a comprehensive evaluation of network binarization from both accuracy and efficiency perspectives. We also provide in-depth analysis of the benchmark results, uncovering insights and offering suggestions for designing practical binarization algorithms.

We emphasize that our BiBench includes the following significant contributions: (1) *The **first** systematic benchmark enables a new view to quantitatively evaluate binarization algorithms at the operator level.* BiBench is the first effort to facilitate systematic and comprehensive comparisons between binarized algorithms. It provides a brand new perspective to decouple the binarized operators from the neural architectures for quantitative evaluations at the generic operator level. (2) *Revealing a practical binarized operator design paradigm.* BiBench reveals a practical paradigm of binarized operator designing. Based on the systemic and quantitative evaluation, superior techniques for more satisfactory binarization operators can emerge, which is essential for pushing binarization algorithms to be accurate and efficient. A more detailed discussion is in Appendix A.5.

2. Background

2.1. Network Binarization

Binarization compresses weights $\boldsymbol{w} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}} \times k \times k}$ and activations $\boldsymbol{a} \in \mathbb{R}^{c_{\text{in}} \times w \times h}$ to 1-bit in computationally dense convolution, where c_{in}, k, c_{out}, w, and h denote the input channel, kernel size, output channel, input width, and input height. The computation can be expressed as

$$\boldsymbol{o} = \alpha \operatorname{popcount}\left(\operatorname{xnor}\left(\operatorname{sign}(\boldsymbol{a}), \operatorname{sign}(\boldsymbol{w})\right)\right), \qquad (1)$$

where $\boldsymbol{o}$ denotes the outputs and $\alpha \in \mathbb{R}^{c_{\text{out}}}$ denotes the optional scaling factor calculated as $\alpha = \frac{\|w\|}{n}$ (Courbariaux et al., 2016b; Rastegari et al., 2016), xnor and popcount are bitwise instructions defined as (Arm, 2020; AMD, 2022). The popcount counts the number of bits with the ”one” value in the input vector and writes the result to the targeted register. While binarized parameters of networks offer significant compression and acceleration benefits, their limited representation can lead to decreased accuracy. Various algorithms have been proposed to address this issue to improve the accuracy of binarized networks (Yuan & Agaian, 2021).

The majority of binarization algorithms aim to improve binarized operators (as Eq. (1) shows), which play a crucial role in the optimization and hardware efficiency of binarized models (Alizadeh et al., 2019; Geiger & Team, 2020). These operator improvements are also flexible across different neural architectures and learning tasks, demonstrating the generalizability of bit-width compression (Wang et al., 2020b; Qin et al., 2022a; Zhao et al., 2022a). Our BiBench considers 8 extensively influential binarization algorithms that

Table 1: Comparison between BiBench and existing binarization works along evaluation tracks.

Algorithm	Technique			Accurate Binarization			Efficient Binarization		
	s	τ	g	#Task	#Arch	Robust	Train	Comp	Infer
BNN (Courbariaux et al., 2016b)	×	×	√	3	3	*	√	√	√
XNOR (Rastegari et al., 2016)	√	×	×	2	3	*	√	√	√
DoReFa (Zhou et al., 2016)	√	×	×	2	2	*	×	√	×
Bi-Real (Liu et al., 2018b)	×	×	√	1	2	×	×	√	×
XNOR++ (Bulat et al., 2019)	√	×	×	1	2	×	×	×	×
ReActNet (Liu et al., 2020)	×	√	×	1	2	×	×	√	×
ReCU (Xu et al., 2021b)	×	√	√	2	4	×	×	×	×
FDA (Xu et al., 2021a)	×	×	√	1	6	×	×	×	×
*Our Benchmark (**BiBench**)*	√	√	√	**9**	**13**	√	√	√	√

[1] "√" and "×" indicates the track is considered in the original binarization algorithm, while "*" indicates only being studied in other related studies. "s", "τ", or "g" indicates "scaling factor", "parameter redistribution", or "gradient approximation" techniques proposed in this work, respectively. And we also present a more detailed list of these techniques (Table 6) for binarization algorithms in Appendix A.1.

focus on improving operators and can be broadly classified into three categories: scaling factors, parameter redistribution, and gradient approximation (Courbariaux et al., 2016b; Rastegari et al., 2016; Zhou et al., 2016; Liu et al., 2018b; Bulat et al., 2019; Liu et al., 2020; Xu et al., 2021b;a). Note that for selected binarization algorithms, the techniques requiring specified local structures or training pipelines are excluded for fairness, *i.e.*, the bi-real shortcut of Bi-Real (Liu et al., 2018a) and duplicate activation of ReActNet (Liu et al., 2020) in CNN neural architectures. See Appendix A.1 for more information on the algorithms in our BiBench.

2.2. Challenges for Binarization

Since around 2015, network binarization has garnered significant attention in various fields of research, including but not limited to vision and language understanding. However, several challenges still arise during the production and deployment of binarized networks in practice. The goal of binarization production is to train accurate binarized networks that are resource-efficient. Some recent studies have demonstrated that the performance of binarization algorithms on image classification tasks may not always generalize to other learning tasks and neural architectures (Qin et al., 2020a; Wang et al., 2020b; Qin et al., 2021; Liu et al., 2022b). In order to achieve higher accuracy, some binarization algorithms may require several times more training resources compared to full-precision networks. Ideally, binarized networks should be hardware-friendly and robust when deployed on edge devices. However, most mainstream inference libraries do not currently support the deployment of binarized networks on hardware (NVIDIA, 2022; HUAWEI, 2022; Qualcomm, 2022), which limits the performance of existing binarization algorithms in practice. In addition, the data collected by low-cost devices in natural edge scenarios is often of low quality and even be corrupted, which can negatively impact the robustness of binarized models (Lin et al., 2018; Ye et al., 2019; Cygert & Czyżewski, 2021). However, most existing binarization algorithms do not consider corruption robustness when designed.

3. BiBench: Tracks and Metrics

In this section, we present BiBench, a benchmark for accurate and efficient network binarization. Our evaluation consists of 6 tracks and corresponding metrics, as shown in Figure 1, which address the practical challenges of producing and deploying binarized networks. Higher scores on these metrics indicate better performance.

3.1. Towards Accurate Binarization

In our BiBench, the evaluation tracks for accurate binarization are "Learning Task", "Neural Architecture" (for production), and "Corruption Robustness" (for deployment).

① **Learning Task**. We comprehensively evaluate network binarization algorithms using 9 learning tasks across 4 different data modalities. For the widely-evaluated 2D visual modality tasks, we include image classification on CIFAR-10 (Krizhevsky et al., 2014) and ImageNet (Krizhevsky et al., 2012) datasets, as well as object detection on PASCAL VOC (Hoiem et al., 2009) and COCO (Lin et al., 2014) datasets. In the 3D visual modality tasks, we evaluate the algorithm on ModelNet40 classification (Wu et al., 2015) and ShapeNet segmentation (Chang et al., 2015) datasets of 3D point clouds. For the textual modality tasks, we use the natural language understanding in the GLUE benchmark (Wang et al., 2018a). For the speech modality tasks, we evaluate the algorithms on the Speech Commands KWS dataset (Warden, 2018). See Appendix. A.2 for more details on the tasks and datasets.

To evaluate the performance of a binarization algorithm on this track, we use the accuracy of full-precision models as a baseline and calculate the mean relative accuracy for all architectures on each task. The Overall Metric (OM) for this track is then calculated as the quadratic mean of the relative accuracies across all tasks (Curtis & Marshall, 2000). The equation for this evaluation metric is as follows:

$$\mathrm{OM}_{\text{task}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}^2\left(\frac{\boldsymbol{A}^{bi}_{\text{task}_i}}{\boldsymbol{A}_{\text{task}_i}}\right)}, \tag{2}$$

where $\boldsymbol{A}^{bi}_{\text{task}_i}$ and $\boldsymbol{A}_{\text{task}_i}$ denote the accuracy of the binarized and full-precision models on the i-th task, respectively, N is the number of tasks, and $\mathbb{E}(\cdot)$ is the mean operation.

Note that the quadratic mean form is used uniformly in BiBench to unify all overall metrics of tracks, which helps prevent certain poor performers from disproportionately influencing the metric and allows for a more accurate measure of overall performance on each track.

② **Neural Architecture**. We evaluate various neural architectures, including mainstream CNN-based, transformer-based, and MLP-based architectures, to assess the generalizability of binarization algorithms from the perspective of neural architecture. Specifically, we use standard ResNet-18/20/34 (He et al., 2016) and VGG (Simonyan & Zisserman, 2015) to evaluate CNN architectures, and apply the Faster-RCNN (Ren et al., 2015) and SSD300 (Liu et al., 2016) frameworks as detectors. To evaluate transformer-based architectures, we binarize BERT-Tiny4/Tiny6/Base (Kenton & Toutanova, 2019) with the bi-attention mechanism (Qin et al., 2021) for convergence. We also evaluate MLP-based architectures, including PointNet$_{\text{vanilla}}$ and PointNet (Qi et al., 2017) with EMA aggregator (Qin et al., 2020a), FSMN (Zhang et al., 2015), and Deep-FSMN (Zhang et al., 2018a), due to their linear unit composition. Detailed descriptions of these architectures can be found in Appendix A.3.

Similar to the overall metric for learning task track, we build the overall metric for neural architecture track:

$$\mathrm{OM}_{\text{arch}} = \sqrt{\frac{1}{3}\left(\mathbb{E}^2\left(\frac{\boldsymbol{A}^{bi}_{\text{CNN}}}{\boldsymbol{A}_{\text{CNN}}}\right) + \mathbb{E}^2\left(\frac{\boldsymbol{A}^{bi}_{\text{Transformer}}}{\boldsymbol{A}_{\text{Transformer}}}\right) + \mathbb{E}^2\left(\frac{\boldsymbol{A}^{bi}_{\text{MLP}}}{\boldsymbol{A}_{\text{MLP}}}\right)\right)}. \tag{3}$$

③ **Corruption Robustness**. The corruption robustness of binarization on deployment is important to handle scenarios such as perceptual device damage, a common issue with low-cost equipment in real-world implementations. To assess the robustness of binarized models to corruption of 2D visual data, we evaluate algorithms on the CIFAR10-C (Hendrycks & Dietterich, 2018) benchmark.

Therefore, we evaluate the performance of binarization algorithms on corrupted data compared to normal data using the corruption generalization gap (Zhang et al., 2022a):

$$\boldsymbol{G}_{\text{task}_i} = \boldsymbol{A}^{\text{norm}}_{\text{task}_i} - \boldsymbol{A}^{\text{corr}}_{\text{task}_i}, \tag{4}$$

where $\boldsymbol{A}^{\text{corr}}_{\text{task}i}$ and $\boldsymbol{A}^{\text{norm}}\text{task}_i$ denote the accuracy results under all architectures on the i-th corruption task and corresponding normal task, respectively. The overall metric on this track is then calculated by

$$\mathrm{OM}_{\text{robust}} = \sqrt{\frac{1}{C}\sum_{i=1}^{C}\mathbb{E}^2\left(\frac{\boldsymbol{G}_{\text{task}_i}}{\boldsymbol{G}^{bi}_{\text{task}_i}}\right)}. \tag{5}$$

3.2. Towards Efficient Binarization

We evaluate the efficiency of network binarization in terms of "Training Consumption" for production, "Theoretical Complexity" and "Hardware Inference" for deployment.

④ **Training Consumption**. We consider the occupied training resource and the hyperparameter sensitivity of binarization algorithms, which impact the consumption of a single training and the overall tuning process. To evaluate the ease of tuning binarization algorithms to optimal performance, we train their binarized networks with various hyperparameter settings, including different learning rates, learning rate schedulers, optimizers, and even random seeds. We align the epochs for binarized and full-precision networks and compare their consumption and time.

The metric used to evaluate the training consumption track is based on both training time and sensitivity to hyperparameters. For one binarization algorithm, we have

$$\mathrm{OM}_{\text{train}} = \sqrt{\frac{1}{2}\left(\mathbb{E}^2\left(\frac{\boldsymbol{T}_{\text{train}}}{\boldsymbol{T}^{bi}_{\text{train}}}\right) + \mathbb{E}^2\left(\frac{\mathrm{std}(\boldsymbol{A}_{\text{hyper}})}{\mathrm{std}(\boldsymbol{A}^{bi}_{\text{hyper}})}\right)\right)}, \tag{6}$$

where the set $\boldsymbol{T}_{\text{train}}$ represents the time spent on a single training instance, $\boldsymbol{A}_{\text{hyper}}$ is the set of results obtained using different hyperparameter configurations, and $\mathrm{std}(\cdot)$ calculates standard deviation values.

⑤ **Theoretical Complexity**. To evaluate complexity, we compute the compression and speedup ratios before and after binarization on architectures such as ResNet18.

The evaluation metric is based on model size (MB) and computational floating-point operations (FLOPs) savings during inference. Binarized parameters occupy 1/32 the storage of their 32-bit floating-point counterparts (Rastegari et al., 2016). Binarized operations, where a 1-bit weight is multiplied by a 1-bit activation, take approximately 1*1/64

FLOPs on a CPU with 64-bit instruction size (Zhou et al., 2016; Liu et al., 2018b; Li et al., 2019). The compression ratio $\boldsymbol{r}_c$ and speedup ratio $\boldsymbol{r}_s$ are

$$\begin{aligned} \boldsymbol{r}_c &= \frac{|\boldsymbol{M}|_{\ell 0}}{\frac{1}{32}\left(|\boldsymbol{M}|_{\ell 0} - |\hat{\boldsymbol{M}}|_{\ell 0}\right) + |\hat{\boldsymbol{M}}|_{\ell 0}}, \\ \boldsymbol{r}_s &= \frac{\text{FLOPs}_{\boldsymbol{M}}}{\frac{1}{64}\left(\text{FLOPs}_{\boldsymbol{M}} - \text{FLOPs}_{\hat{\boldsymbol{M}}}\right) + \text{FLOPs}_{\hat{\boldsymbol{M}}}}, \end{aligned} \tag{7}$$

where $\boldsymbol{M}$ and $\hat{\boldsymbol{M}}$ are the number of full-precision parameters that remains in the original and binarized models, respectively, and $\text{FLOPs}_{\boldsymbol{M}}$ and $\text{FLOPs}_{\hat{\boldsymbol{M}}}$ represent the computation related to these parameters. The overall metric for theoretical complexity is

$$\text{OM}_{\text{comp}} = \sqrt{\frac{1}{2}\left(\mathbb{E}^2(\boldsymbol{r}_c) + \mathbb{E}^2(\boldsymbol{r}_s)\right)}. \tag{8}$$

⑥ **Hardware Inference**. As binarization is not widely supported in hardware deployment, only two inference libraries, Larq's Compute Engine (Geiger & Team, 2020) and JD's daBNN (Zhang et al., 2019), can deploy and evaluate binarized models on ARM hardware in practice. We focus on ARM CPU inference on mainstream hardware for edge scenarios, such as HUAWEI Kirin, Qualcomm Snapdragon, Apple M1, MediaTek Dimensity, and Raspberry Pi (details in Appendix A.4).

For a given binarization algorithm, we use the savings in storage and inference time under different inference libraries and hardware as evaluation metrics:

$$\text{OM}_{\text{infer}} = \sqrt{\frac{1}{2}\left(\mathbb{E}^2\left(\frac{\boldsymbol{T}_{\text{infer}}}{\boldsymbol{T}^{bi}_{\text{infer}}}\right) + \mathbb{E}^2\left(\frac{\boldsymbol{S}_{\text{infer}}}{\boldsymbol{S}^{bi}_{\text{infer}}}\right)\right)}, \tag{9}$$

where $\boldsymbol{T}_{\text{infer}}$ is the inference time and $\boldsymbol{S}_{\text{infer}}$ is the storage used on different devices.

4. BiBench Implementation

This section presents the implementation details, training, and inference pipelines of BiBench.

Implementation details. BiBench is implemented using the PyTorch (Paszke et al., 2019) package. The definitions of the binarized operators are contained in individual, separate files, enabling the flexible replacement of the corresponding operator in the original model when evaluating different tasks and architectures. When deployed, well-trained binarized models for a particular binarization algorithm are exported to the Open Neural Network Exchange (ONNX) format (developers, 2021) and provided as input to the appropriate inference libraries (if applicable for the algorithm).

Training and inference pipelines. *Hyperparameters*: Binarized networks are trained for the same number of epochs as their full-precision counterparts. Inspired by the results in Section 5.2.1, we use the Adam optimizer for all binarized models for well converging. The default initial learning rate is $1e-3$ (or $0.1\times$ the default learning rate), and the learning rate scheduler is CosineAnnealingLR (Loshchilov & Hutter, 2017). *Architecture*: BiBench follows the original architectures of full-precision models, binarizing their convolution, linear, and multiplication units with the selected binarization algorithms. Hardtanh is uniformly used as the activation function to prevent all-one features. *Pretraining*: All binarization algorithms use finetuning. For each one, all binarized models are initialized using the same pre-trained model for the specific neural architecture and learning task to eliminate inconsistency at initialization.

5. BiBench Evaluation and Analysis

This section presents and analyzes the evaluation results in BiBench. The main accuracy results are in Table 2, and the efficiency results are in Table 3. Details are in Appendix B.

5.1. Accuracy Tracks

The accuracy results for network binarization are presented in Table 2. These results were obtained using the metrics defined in Section 3.1 for each accuracy-related track.

5.1.1. Learning Task: Performance Varies Greatly by Algorithms and Modalities

We present the evaluation results of binarization on various learning tasks. In addition to the overall metric OM_{task}, we also provide the relative accuracy of binarized networks compared to their full-precision counterparts.

The impact of binarized operators is crucial and significant. With fully unified training pipelines and architectures, a substantial variation in performance appears among binarization algorithms across every learning task. For example, the SOTA FDA algorithm exhibits a 21.3% improvement in accuracy on GLUE datasets compared to XNOR++, and the difference is even greater at 33.0% on ShapeNet between XNOR and ReCU. This suggests that binarized operators play a crucial role in the learning task track, and their importance is also confirmed in other tracks.

Binarization algorithms vary greatly under different data modalities. When comparing various learning tasks, it is notable that binarized networks suffer a significant drop in accuracy on the language understanding GLUE benchmark but can approach full-precision performance on the ModelNet40 point cloud classification task. This and similar phenomena suggest that the direct transfer of binarization insights across learning tasks is non-trivial.

For overall performance, both ReCU and ReActNet have

Table 2: Accuracy benchmark for network binarization. Blue: best in a row. Red: worst in a row.

Track	Metric	Binarization Algorithm							
		BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA
Learning Task (%)	CIFAR10	94.54	94.73	95.03	95.61	94.52	95.92	96.72	94.66
	ImageNet	75.81	77.24	76.61	78.38	75.01	78.64	77.98	78.15
	VOC07	76.97	74.61	76.35	80.07	74.41	81.38	81.65	79.02
	COCO17	77.94	75.37	78.31	81.62	79.41	83.82	85.66	82.35
	ModelNet40	54.19	93.86	93.74	93.23	85.20	92.41	95.07	94.38
	ShapeNet	48.96	73.62	70.79	68.13	41.16	68.51	40.65	71.16
	GLUE	49.75	59.63	66.60	69.42	49.33	67.64	50.66	70.61
	SpeechCom.	75.03	76.93	76.64	82.42	68.65	81.86	76.98	77.90
	OM$_{\text{task}}$	70.82	78.97	79.82	81.63	72.89	81.81	77.96	81.49
Neural Architecture (%)	CNNs	72.90	83.74	83.86	85.02	78.95	86.20	83.50	86.34
	Transformers	49.75	59.63	66.60	69.42	49.33	67.64	50.66	70.61
	MLPs	64.61	85.40	85.19	87.83	76.92	87.13	86.02	86.14
	OM$_{\text{arch}}$	63.15	77.16	79.01	81.16	69.72	80.82	75.14	81.36
Robustness Corruption (%)	CIFAR10-C	95.26	100.97	81.43	96.56	92.69	94.01	103.29	98.35
	OM$_{\text{corr}}$	95.26	100.97	81.43	96.56	92.69	94.01	103.29	98.35

high accuracy across various learning tasks. While ReCU performs best on most individual tasks, ReActNet ultimately stands out in the overall metric comparison. Both algorithms apply reparameterization in the forward propagation and gradient approximation in the backward propagation.

5.1.2. Neural Architecture: Binarization on Transformers Is Challenging

Binarization exhibits a clear advantage on CNN- and MLP-based architectures compared to transformer-based ones. Advanced binarization algorithms can achieve 78%-86% of full-precision accuracy on CNNs, and binarized networks with MLP architectures can even approach full-precision performance (*e.g.*, Bi-Real 87.83%). In contrast, transformer-based architectures suffer significant performance degradation when binarized, and none of the algorithms achieve an overall accuracy higher than 70%.

The main reason for the worse performance of transformer-based architectures is the activation binarization in the attention mechanism. Compared to CNNs/MLPs that mainly use convolution or linear operations on the input data, the transformer-based architectures heavily rely on the attention mechanism, which involves multiplications operation between two binarized activations (between the query and key, attention possibilities, and value) and causes twice as much loss of information in one computation. Moreover, the binarization of activations is determined and cannot be adjusted during training, while the binarized weight can be learned through backward propagation during training. Since the attention mechanism requires more binarization of activation, the training of binarized transformers is more difficult and the inference is more unstable compared to CNNs/MLPs, while the binarized weights of the latter participate in each binarization computation and can be continuously optimized during training. We present more detailed discussions and empirical results in Appendix A.7.

These results indicate that transformer-based architectures, with their unique attention mechanisms, require specialized binarization designs rather than direct binarization. And the overall winner on the architecture track is the FDA algorithm, which performs best on both CNNs and transformers. The evaluation of these two tracks shows that binarization algorithms that use statistical channel-wise scaling factors and custom gradient approximation, such as FDA and ReActNet, have some degree of stability advantage.

5.1.3. Corruption Robustness: Binarization Exhibits Robust to Corruption

Binarized networks can approach full-precision level robustness for corruption. Interestingly, binarized networks demonstrate robustness comparable to full-precision counterparts when evaluated for corruption. Evaluation results on the CIFAR10-C dataset reveal that binarized networks perform similarly to full-precision networks on typical 2D image corruption tasks. In some cases, such as ReCU and XNOR-Net, binarized networks outperform their full-precision counterparts. If the same level of robustness to corruption is required, the binarized network version typically requires little additional design or supervision to achieve it. As such, binarized networks generally exhibit similar robustness to corruption as full-precision networks, which appears to be a general property of binarized networks rather than a

Table 3: Efficiency benchmark for network binarization. Blue: best in a row. Red: worst in a row.

Track	Metric	Binarization Algorithm							
		BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA
Training	Sensitivity	27.28	175.53	113.40	144.59	28.66	146.06	53.33	36.62
Consumption	Time Cost	82.19	71.43	76.92	45.80	68.18	45.45	58.25	20.62
(%)	**OM$_{\text{train}}$**	61.23	134.00	96.89	107.25	52.30	108.16	55.84	29.72
Theoretical	Speedup	12.60	12.26	12.37	12.37	12.26	12.26	12.37	12.37
Complexity	Compression	13.27	13.20	13.20	13.20	13.16	13.20	13.20	13.20
($\times$)	**OM$_{\text{comp}}$**	12.94	12.74	12.79	12.79	12.71	12.74	12.79	12.79
Hardware	Speedup	5.45	False	5.45	5.45	False	4.89	5.45	5.45
Deployment	Compression	15.62	False	15.62	15.62	False	15.52	15.62	15.62
($\times$)	**OM$_{\text{infer}}$**	11.70	False	11.70	11.70	False	11.51	11.70	11.70

specific characteristic of certain algorithms. And our results also suggest that the reason binarized models are robust to data corruption cannot be directly attributed to the smaller model scale, but is more likely to be a unique characteristic of some binarization algorithms (Appendix A.8).

We also analyze that the robustness of BNN to data corruption originates from the high discretization of parameters from the perspective of interpretability. Previous studies pointed out that the robustness of the quantization network is related to both the quantization bucket (range) and the magnitude of the noise (Lin et al., 2018), where the former depends on the bit-width and quantizer and the latter depends on the input noise. Specifically, when the magnitude of the noise is small, the quantization bucket is capable of reducing the errors by eliminating small perturbations; however, when the magnitude of perturbation is larger than a certain threshold, quantization instead amplifies the errors. This fact precisely leads to the natural advantages of the binary network in the presence of natural noise. (1) 1-bit binarization enjoys the largest quantization bucket among all bits quantization. In binarization, the application of the sign function allows the binarizer to be regarded as consisting of a semi-open closed interval $[0, +\infty)$ and an open interval $(-\infty, 0)$ quantization bucket. So binarization is with the largest quantization bucket among all bit-width quantization (most of their quantization buckets are with closed intervals with limited range) and brings the largest noise tolerance to BNNs. (2) Natural data corruption (we evaluated in BiBench) is typically considered to have a smaller magnitude than adversarial noise which is always considered worst-case (Ren et al., 2022). (1) and (2) show that when BNNs encounter corrupted inputs, the high discretization of parameters makes them more robust.

5.2. Efficiency Tracks

We analyze the efficiency metrics of training consumption, theoretical complexity, and hardware inference (Table 3).

5.2.1. Training Consumption: Binarization Could Be Stable yet Generally Expensive

We thoroughly examine the training cost of binarization algorithms on ResNet18 for CIFAR10 and present the sensitivity and training time results for different binarization algorithms in Table 3 and Figure 3, respectively.

“Binarization≠sensitivity”: existing techniques can stabilize binarization-aware training. It is commonly believed that the training of binarized networks is more sensitive to training settings than full-precision networks due to the representation limitations and gradient approximation errors introduced by the high degree of discretization. However, we find that the hyperparameter sensitivities of existing binarization algorithms are polarized, with some being even more hyperparameter-stable than the training of full-precision networks, while others fluctuate greatly. This variation is due to the different techniques used by the binarized operators of these algorithms. Hyperparameter-stable binarization algorithms often share the following characteristics: (1) *channel-wise scaling factors* based on learning or statistics; (2) *soft approximation* to reduce gradient error. These stable algorithms may not necessarily outperform others, but they can simplify the tuning process in production and provide reliable accuracy with a single training.

The preference for hyperparameter settings is also clear and stable binarized networks. Statistical results in Figure 2 show that training with Adam optimizer, the learning rate equaling to the full-precision network ($1\times$), and the CosineAnnealingLR scheduler is more stable than other settings. Based on this, we use this setting as part of the standard training pipelines when evaluating binarization.

Soft approximation in binarization leads to a significant increase in training time. Comparing the time consumed by each binarization algorithm, we found that the training time of algorithms using custom gradient approximation techniques such as Bi-Real and ReActNet increased signifi-

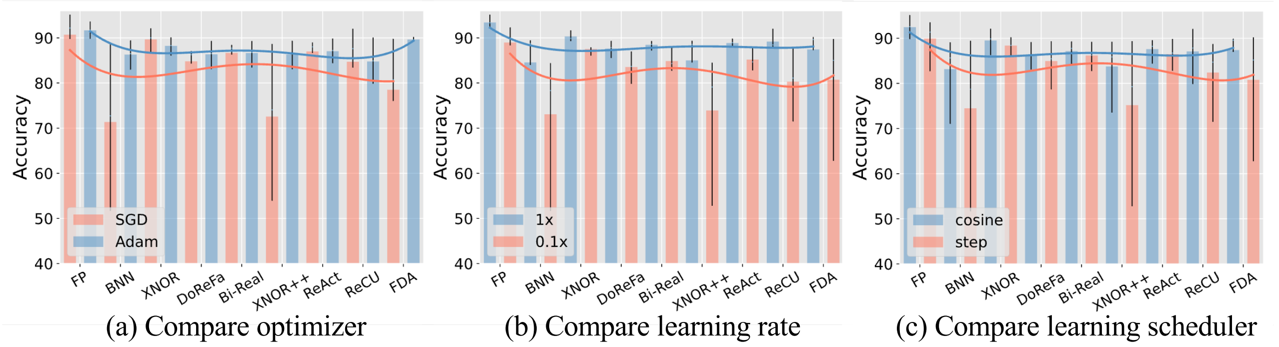


Figure 2: Comparisons of accuracy under different training settings.

cantly. The metric about the training time of FDA is even as high as 20.62%, meaning it is almost 5× the training time of a full-precision network.

5.2.2. Theoretical Complexity: Different Algorithms have Similar Complexity

There is a minor difference in theoretical complexity among binarization algorithms. The leading cause of the difference in compression rate is each model's definition of the static scaling factor. For example, BNN does not use any factors and has the highest compression. In terms of theoretical acceleration, the main difference comes from two factors: the reduction in the static scaling factor also improves theoretical speedup, and real-time re-scaling and mean-shifting for activation add additional computation, such as in the case of ReActNet, which reduces the speedup by 0.11 times. In general, the theoretical complexity of each method is similar, with overall metrics in the range of $[12.71, 12.94]$. These results suggest that binarization algorithms should have similar inference efficiency.

5.2.3. Hardware Inference: Immense Potential on Edge Devices Despite Limited Supports

One of the advantages of the hardware inference track is that it provides valuable insights into the practicalities of deploying binarization in real-world settings. This track stands out from other tracks in this regard.

Limited inference libraries lead to almost fixed paradigms of binarization deployment. The availability of open-source inference libraries that support the deployment of binarization algorithms on hardware is quite limited. After investigating the existing options, we found that only Larq (Geiger & Team, 2020) and daBNN (Zhang et al., 2019) offer complete deployment pipelines and primarily support deployment on ARM devices. As shown in Table 4, both libraries support channel-wise scaling factors in the floating-point form that must be fused into the Batch Normalization (BN) layer. However, neither of them supports dynamic activation statistics or re-scaling during inference. Larq also includes support for mean-shifting activation with a fixed bias. These limitations in the available inference libraries' deployment capabilities have significantly impacted the practicality of deploying binarization algorithms. For example, the scale factor shape of XNOR++ caused its deployment to fail and XNOR also failed due to its activation re-scaling technique; BNN and DoReFa have different theoretical complexities but have the exact same true computation efficiency when deployed. These constraints have resulted in a situation where the vast majority of binarization methods have almost identical inference performance, with the mean-shifting operation of ReActNet on activation having only a slight impact on efficiency. As a result, binarized models must adhere to fixed deployment paradigms and have almost identical efficiency performance.

Born for the edge: more promising for lower-power edge computing. After evaluating the performance of binarized models on a range of different chips, we found that the average speedup of the binarization algorithm was higher on chips with lower computing power (Figure 3). This counter-intuitive result is likely since higher-performance chips tend to have more acceleration from multi-threading when running floating-point models, leading to a relatively slower speedup of binarized models on these chips. In contrast, binarization technology is particularly effective on edge chips with lower performance and cost. Its extreme compression and acceleration capabilities can enable the deployment of advanced neural networks at the edge. These findings suggest that binarization is well-suited for low-power, cost-sensitive edge devices.

5.3. Suggested Paradigm of Binarization Algorithm

Based on our evaluation and analysis, we propose the following paradigm for achieving accurate and efficient network binarization using existing techniques: (1) **Soft gradient approximation** presents great potential. It improves performance by increasing training rather than deployment cost.

Table 4: Deployment capability of different inference libraries on real hardware.

Infer. Lib.	Provider	s Granularity	s Form	Flod BN	Act. Re-scaling	Act. Mean-shifting
Larq	Larq	Channel-wise	FP32	$\surd$	$\times$	$\surd$
daBNN	JD	Channel-wise	FP32	$\surd$	$\times$	$\times$
Algorithm	**Deployable**	**s Granularity**	**s Form**	**Flod BN**	**Act. Re-scaling**	**Act. Mean-shifting**
BNN	$\surd$	N/A	N/A	N/A	$\times$	$\times$
XNOR	$\times$	Channel-wise	FP32	$\surd$	$\surd$	$\times$
DoReFa	$\surd$	Channel-wise	FP32	$\surd$	$\times$	$\times$
Bi-Real	$\surd$	Channel-wise	FP32	$\surd$	$\times$	$\times$
XNOR++	$\times$	Spatial-wise	FP32	$\times$	$\times$	$\times$
ReActNet	$\surd$	Channel-wise	FP32	$\surd$	$\times$	$\surd$
ReCU	$\surd$	Channel-wise	FP32	$\surd$	$\times$	$\times$
FDA	$\surd$	Channel-wise	FP32	$\surd$	$\times$	$\times$

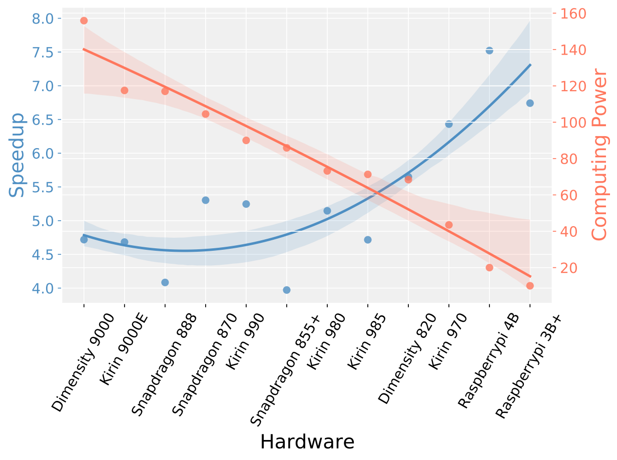


Figure 3: The lower the chip's computing power, the higher the inference speedup of deployed binarized models.

And all accuracy-winning algorithms adopt this technique, *i.e.*, ReActNet, ReCU, and FDA. By further exploring this technique, FDA outperforms previous algorithms on the architecture track, indicating the great potential of the soft gradient approximation technique. (2) **Channel-wise scaling factors** are currently the optimal option for binarization. The gain from the floating-point scaling factor is demonstrated in accuracy tracks, and deployable consideration limits its form to channel-wise. This means a balanced trade-off between accuracy and efficiency. (3) **Pre-binarization parameter redistributing** is an optional but beneficial operation that can be implemented as a mean-shifting for weights or activations before binarization. Our findings indicate that this technique can significantly enhance accuracy with little added inference cost, as seen in ReActNet and ReCU.

It is important to note that, despite the insights gained from benchmarking on evaluation tracks, **none of the binarization techniques or algorithms work well across all scenarios so far**. Further research is needed to overcome the current limitations and mutual restrictions between production and deployment, and to develop binarization algorithms that consider both deployability and efficiency. Additionally, it would be helpful for inference libraries to support more advanced binarized operators. In the future, the focus of binarization research should be on addressing these issues.

6. Discussion

In this paper, we propose BiBench, a versatile and comprehensive benchmark toward the fundamentals of network binarization. BiBench covers 8 network binarization algorithms, 9 deep learning datasets (including a corruption one), 13 different neural architectures, 2 deployment libraries, 14 real-world hardware, and various hyperparameter settings. Based on these scopes, we develop evaluation tracks to measure the accuracy under multiple conditions and efficiency when deployed on actual hardware. By benchmark results and analysis, BiBench summarizes an empirically optimized paradigm with several critical considerations for designing accurate and efficient binarization algorithms. BiBench aims to provide a comprehensive and unbiased resource for researchers and practitioners working in model binarization. We hope BiBench can facilitate a fair comparison of algorithms through a systematic investigation with metrics that reflect the fundamental requirements for model binarization and serve as a foundation for applying this technology in broader and more practical scenarios.

Acknowledgement. Supported by the National Key Research and Development Plan of China (2021ZD0110503), the National Natural Science Foundation of China (No. 62022009), the State Key Laboratory of Software Development Environment (SKLSDE-2022ZX-23), NTU NAP, MOE AcRF Tier 2 (T2EP20221-0033), and under the RIE2020 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s).

References

Alizadeh, M., Fernández-Marqués, J., Lane, N. D., and Gal, Y. A systematic study of binary neural networks' optimisation. In *ICLR*, 2019.

AMD. Amd64 architecture programmer's manual. https://developer.arm.com/documentation/ddi0596/2020-12/SIMD-FP-Instructions/CNT--Population-Count-per-byte-, 2022.

Arm. Arm a64 instruction set architecture. https://www.amd.com/system/files/TechDocs/24594.pdf, 2020.

Bai, H., Zhang, W., Hou, L., Shang, L., Jin, J., Jiang, X., Liu, Q., Lyu, M. R., and King, I. Binarybert: Pushing the limit of bert quantization. In *ACL*, 2021.

Bethge, J., Yang, H., Bornstein, M., and Meinel, C. Back to simplicity: How to train accurate bnns from scratch? *arXiv preprint arXiv:1906.08637*, 2019.

Bethge, J., Bartz, C., Yang, H., Chen, Y., and Meinel, C. Meliusnet: Can binary neural networks achieve mobilenet-level accuracy? *arXiv preprint arXiv:2001.05936*, 2020.

Bulat, A., Tzimiropoulos, G., and Center, S. A. Xnor-net++: Improved binary neural networks. In *BMVC*, 2019.

Bulat, A., Martinez, B., and Tzimiropoulos, G. High-capacity expert binary networks. In *ICLR*, 2020.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Chen, H., Wen, Y., Ding, Y., Yang, Z., Guo, Y., and Qin, H. An empirical study of data-free quantization's tuning robustness. In *CVPR*, 2022.

Chen, Y., Zhang, Z., and Wang, N. Darkrank: Accelerating deep metric learning via cross sample similarities transfer. In *AAAI*, 2018.

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.

Choukroun, Y., Kravchik, E., Yang, F., and Kisilev, P. Low-bit quantization of neural networks for efficient inference. In *ICCVW*, 2019.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016a.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016b.

Curtis, R. O. and Marshall, D. D. Why quadratic mean diameter? *WJAF*, 2000.

Cygert, S. and Czyżewski, A. Robustness in compressed neural networks for object detection. In *IJCNN*, 2021.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Li, F. F. Imagenet: a large-scale hierarchical image database. In *CVPR*, 2009.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *NeurIPS*, 2014.

developers, O. R. Onnx runtime. https://onnxruntime.ai/, 2021.

Diffenderfer, J. and Kailkhura, B. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. *arXiv preprint arXiv:2103.09377*, 2021.

Ding, X., Hao, T., Tan, J., Liu, J., Han, J., Guo, Y., and Ding, G. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *CVPR*, 2021.

Ding, Y., Qin, H., Yan, Q., Chai, Z., Liu, J., Wei, X., and Liu, X. Towards accurate post-training quantization for vision transformer. In *ACM MM*, 2022.

Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. Learned step size quantization. In *ICLR*, 2019.

Geiger, L. and Team, P. Larq: An open-source library for training binarized neural networks. *Journal of Open Source Software*, 2020.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. In *LPCV*, 2021.

Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., Yu, F., and Yan, J. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *IEEE ICCV*, 2019.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. D. Compressing deep convolutional networks using vector quantization. *arXiv: Computer Vision and Pattern Recognition*, 2014.

Gu, J., Li, C., Zhang, B., Han, J., Cao, X., Liu, J., and Doermann, D. Projection convolutional neural networks for 1-bit cnns via discrete back propagation. In *AAAI*, 2019.

Guo, J., Ouyang, W., and Xu, D. Multi-dimensional pruning: A unified framework for model compression. In *CVPR*, 2020a.

Guo, J., Zhang, W., Ouyang, W., and Xu, D. Model compression using progressive channel pruning. *IEEE TCSVT*, 2020b.

Guo, J., Liu, J., and Xu, D. Jointpruning: Pruning networks along multiple dimensions for efficient point cloud processing. *IEEE TCSVT*, 2021.

Guo, J., Bao, W., Wang, J., Ma, Y., Gao, X., Xiao, G., Liu, A., Dong, J., Liu, X., and Wu, W. A comprehensive evaluation framework for deep model robustness. *Pattern Recognition*, 2023a.

Guo, J., Xu, D., and Lu, G. Cbanet: Towards complexity and bitrate adaptive deep image compression using a single network. *IEEE TIP*, 2023b.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. *ICML*, 2015.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *NeurIPS*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, X., Mo, Z., Cheng, K., Xu, W., Hu, Q., Wang, P., Liu, Q., and Cheng, J. Proxybnn: Learning binarized neural networks via proxy matrices. In *ECCV*, 2020.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *IEEE ICCV*, 2017.

Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K.-T., and Nusselder, R. Latent weights do not exist: Rethinking binarized neural network optimization. *NeurIPS*, 2019.

Hendrycks, D. and Dietterich, T. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2018.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv: Machine Learning*, 2015.

Hisilicon. Kirin chipsets, 2022. URL `https://www.hisilicon.com/cn/products/Kirin`.

Hoiem, D., Divvala, S. K., and Hays, J. H. Pascal voc 2008 challenge. *World Literature Today*, 2009.

Hou, L., Yao, Q., and Kwok, J. T. Loss-aware binarization of deep networks. In *ICLR*, 2017.

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.

Hu, Q., Li, G., Wang, P., Zhang, Y., and Cheng, J. Training binary weight networks via semi-binary decomposition. In *ECCV*, 2018.

HUAWEI. Acl: Quantization factor file, 2022. URL `https://support.huaweicloud.com/intl/en-us/ti-mc-A200_3000/altasmodelling_16_043.html`.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *NeurIPS*, 2016.

J. Guo, W. Ouyang, and D. Xu. Channel pruning guided by classification loss and feature importance. In *AAAI*, 2020.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

Juefei-Xu, F., Naresh Boddeti, V., and Savvides, M. Local binary convolutional neural networks. In *CVPR*, 2017.

Kenton, J. D. M.-W. C. and Toutanova, L. K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

Kim, M. and Smaragdis, P. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 2012.

Krizhevsky, A., Nair, V., and Hinton, G. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

Lebedev, V. and Lempitsky, V. Fast convnets using group-wise brain damage. In *CVPR*, 2016.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I. V., and Lempitsky, V. S. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*, 2015.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Li, Y., Dong, X., and Wang, W. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *ICLR*, 2019.

Lin, J., Gan, C., and Han, S. Defensive quantization: When efficiency meets robustness. In *ICLR*, 2018.

Lin, M., Ji, R., Xu, Z., Zhang, B., Wang, Y., Wu, Y., Huang, F., and Lin, C.-W. Rotated binary neural network. *NeurIPS*, 2020.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *ECCV*, 2014.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *NeurIPS*, 2017.

Liu, A., Liu, X., Fan, J., Ma, Y., Zhang, A., Xie, H., and Tao, D. Perceptual-sensitive gan for generating adversarial patches. In *AAAI*, 2019.

Liu, A., Liu, X., Yu, H., Zhang, C., Liu, Q., and Tao, D. Training robust deep neural networks via adversarial noise propagation. *IEEE TIP*, 2021a.

Liu, A., Guo, J., Wang, J., Liang, S., Tao, R., Zhou, W., Liu, C., Liu, X., and Tao, D. X-adv: Physical adversarial object attacks against x-ray prohibited item detection. In *USENIX Security*, 2023.

Liu, C., Chen, P., Zhuang, B., Shen, C., Zhang, B., and Ding, W. Sa-bnn: State-aware binary neural network. In *AAAI*, 2021b.

Liu, J., Guo, J., and Xu, D. Apsnet: Toward adaptive point sampling for efficient 3d action recognition. *IEEE TIP*, 2022a.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. In *ECCV*, 2016.

Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., and Cheng, K.-T. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, 2018a.

Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., and Cheng, K.-T. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, 2018b.

Liu, Z., Shen, Z., Savvides, M., and Cheng, K.-T. Reactnet: Towards precise binary neural network with generalized activation functions. In *ECCV*, 2020.

Liu, Z., Oguz, B., Pappu, A., Xiao, L., Yih, S., Li, M., Krishnamoorthi, R., and Mehdad, Y. Bit: Robustly binarized multi-distilled transformer. In *NeurIPS*, 2022b.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

Martinez, B., Yang, J., Bulat, A., and Tzimiropoulos, G. Training binary neural networks with real-to-binary convolutions. In *ICLR*, 2019.

MediaTek. Mediatek 5g, 2022. URL `https://i.mediatek.com/mediatek-5g`.

NVIDIA. Tensorrt: A c++ library for high performance inference on nvidia gpus and deep learning accelerators, 2022. URL `https://github.com/NVIDIA/TensorRT`.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

Qin, H. Hardware-friendly deep learning by network quantization and binarization. *arXiv preprint arXiv:2112.00737*, 2021.

Qin, H., Cai, Z., Zhang, M., Ding, Y., Zhao, H., Yi, S., Liu, X., and Su, H. Bipointnet: Binary neural network for point clouds. In *ICLR*, 2020a.

Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., and Song, J. Forward and backward information retention for accurate binary neural networks. In *CVPR*, 2020b.

Qin, H., Ding, Y., Zhang, M., Qinghua, Y., Liu, A., Dang, Q., Liu, Z., and Liu, X. Bibert: Accurate fully binarized bert. In *ICLR*, 2021.

Qin, H., Ma, X., Ding, Y., Li, X., Zhang, Y., Tian, Y., Ma, Z., Luo, J., and Liu, X. Bifsmn: Binary neural network for keyword spotting. In *IJCAI*, 2022a.

Qin, H., Zhang, X., Gong, R., Ding, Y., Xu, Y., and Liu, X. Distribution-sensitive information retention for accurate binary neural network. *IJCV*, 2022b.

Qin, H., Ding, Y., Zhang, X., Wang, J., Liu, X., and Lu, J. Diverse sample generation: Pushing the limit of generative data-free quantization. *IEEE TPAMI*, 2023a.

Qin, H., Ma, X., Ding, Y., Li, X., Zhang, Y., Ma, Z., Wang, J., Luo, J., and Liu, X. Bifsmnv2: Pushing binary neural networks for keyword spotting to real-network performance. *IEEE TNNLS*, 2023b.

Qualcomm. Snpe: Qualcomm neural processing sdk for ai, 2022. URL https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

Ren, J., Pan, L., and Liu, Z. Benchmarking and analyzing point cloud classification under corruptions. In *ICML*, 2022.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *NeurIPS*, 2015.

Rusci, M., Capotondi, A., and Benini, L. Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers. *MLSys*, 2020.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Shang, Y., Xu, D., Duan, B., Zong, Z., Nie, L., and Yan, Y. Lipschitz continuity retained binary neural network. In *ECCV*, 2022a.

Shang, Y., Xu, D., Zong, Z., Nie, L., and Yan, Y. Network binarization via contrastive learning. In *ECCV*, 2022b.

Shen, Z., Liu, Z., Qin, J., Huang, L., Cheng, K.-T., and Savvides, M. S2-bnn: Bridging the gap between self-supervised real and 1-bit neural networks via guided distribution calibration. In *CVPR*, 2021.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Singh, M. P. and Jain, M. K. Evolution of processor architecture in mobile phones. *IJCA*, 2014.

Tseng, V.-S., Bhattachara, S., Fernández-Marqués, J., Alizadeh, M., Tong, C., and Lane, N. D. Deterministic binary filters for convolutional neural networks. IJCAI, 2018.

Vanhoucke, V., Senior, A. W., and Mao, M. Z. Improving the speed of neural networks on cpus. 2011.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2018a.

Wang, J. Adversarial examples in physical world. In *IJCAI*, 2021.

Wang, J., Liu, A., Bai, X., and Liu, X. Universal adversarial patch attack for automatic checkout using perceptual and attentional bias. *IEEE TIP*, 2021a.

Wang, J., Liu, A., Yin, Z., Liu, S., Tang, S., and Liu, X. Dual attention suppression attack: Generate adversarial camouflage in physical world. In *CVPR*, 2021b.

Wang, J., Yin, Z., Hu, P., Liu, A., Tao, R., Qin, H., Liu, X., and Tao, D. Defensive patches for robust recognition in the physical world. In *CVPR*, 2022a.

Wang, P., He, X., Li, G., Zhao, T., and Cheng, J. Sparsity-inducing binarized neural networks. In *AAAI*, 2020a.

Wang, X., Zhang, B., Li, C., Ji, R., Han, J., Cao, X., and Liu, J. Modulated convolutional networks. In *CVPR*, 2018b.

Wang, Y., Wang, J., Yin, Z., Gong, R., Wang, J., Liu, A., and Liu, X. Generating transferable adversarial examples against vision transformers. In *ACM MM*, 2022b.

Wang, Z., Lu, J., Tao, C., Zhou, J., and Tian, Q. Learning channel-wise interactions for binary convolutional neural networks. In *CVPR*, 2019.

Wang, Z., Wu, Z., Lu, J., and Zhou, J. Bidet: An efficient binarized object detector. In *CVPR*, 2020b.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

Wikipedia. Apple m1 - wikipedia, 2022a. URL https://en.wikipedia.org/wiki/Apple_M1.

Wikipedia. Raspberry pi - wikipedia, 2022b. URL https://en.wikipedia.org/wiki/Raspberry_Pi.

Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015.

Xiao, Y., Zhang, T., Liu, S., and Qin, H. Benchmarking the robustness of quantized models. *arXiv preprint arXiv:2304.03968*, 2023.

Xu, Y., Han, K., Xu, C., Tang, Y., Xu, C., and Wang, Y. Learning frequency domain approximation for binary neural networks. *NeurIPS*, 2021a.

Xu, Z., Hsu, Y., and Huang, J. Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks. In *ICLR*, 2018.

Xu, Z., Lin, M., Liu, J., Chen, J., Shao, L., Gao, Y., Tian, Y., and Ji, R. Recu: Reviving the dead weights in binary neural networks. In *IEEE ICCV*, 2021b.

Ye, S., Xu, K., Liu, S., Cheng, H., Lambrechts, J.-H., Zhang, H., Zhou, A., Ma, K., Wang, Y., and Lin, X. Adversarial robustness vs. model compression, or both? In *IEEE ICCV*, 2019.

Yim, J., Joo, D., Bae, J., and Kim, J. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *CVPR*, 2017.

Yin, Z., Wang, J., Ding, Y., Xiao, Y., Guo, J., Tao, R., and Qin, H. Improving generalization of deepfake detection with domain adaptive batch normalization. In *ACM MMW*, 2021.

Yuan, C. and Agaian, S. S. A comprehensive review of binary neural network. *arXiv preprint arXiv:2110.06804*, 2021.

Zagoruyko, S. and Komodakis, N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *ICLR*, 2017.

Zhang, C., Zhang, M., Zhang, S., Jin, D., Zhou, Q., Cai, Z., Zhao, H., Liu, X., and Liu, Z. Delving deep into the generalization of vision transformers under distribution shifts. In *CVPR*, 2022a.

Zhang, J., Pan, Y., Yao, T., Zhao, H., and Mei, T. dabnn: A super fast inference framework for binary neural networks on arm devices. In *ACM MM*, 2019.

Zhang, S., Liu, C., Jiang, H., Wei, S., Dai, L., and Hu, Y. Feedforward sequential memory networks: A new structure to learn long-term dependency. *arXiv preprint arXiv:1512.08301*, 2015.

Zhang, S., Lei, M., Yan, Z., and Dai, L. Deep-fsmn for large vocabulary continuous speech recognition. In *ICASSP*, 2018a.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018b.

Zhang, X., Qin, H., Ding, Y., Gong, R., Yan, Q., Tao, R., Li, Y., Yu, F., and Liu, X. Diversifying sample generation for accurate data-free quantization. In *CVPR*, 2021a.

Zhang, Y., Pan, J., Liu, X., Chen, H., Chen, D., and Zhang, Z. Fracbnn: Accurate and fpga-efficient binary neural networks with fractional activations. In *FPGA*, 2021b.

Zhang, Y., Zhang, Z., and Lew, L. Pokebnn: A binary pursuit of lightweight accuracy. In *CVPR*, 2022b.

Zhao, M., Dai, S., Zhu, Y., Tang, H., Xie, P., Li, Y., Liu, C., and Zhang, B. Pb-gcn: Progressive binary graph convolutional networks for skeleton-based action recognition. *Neurocomputing*, 2022a.

Zhao, R., Song, W., Zhang, W., Xing, T., Lin, J.-H., Srivastava, M., Gupta, R., and Zhang, Z. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *FPGA*, 2017.

Zhao, Z., Xu, S., Zhang, C., Liu, J., and Zhang, J. Bayesian fusion for infrared and visible images. *Signal Processing*, 2020a.

Zhao, Z., Xu, S., Zhang, C., Liu, J., Zhang, J., and Li, P. Didfuse: Deep image decomposition for infrared and visible image fusion. In *IJCAI*, 2020b.

Zhao, Z., Bai, H., Zhang, J., Zhang, Y., Xu, S., Lin, Z., Timofte, R., and Gool, L. V. Cddfuse: Correlation-driven dual-branch feature decomposition for multi-modality image fusion. *CoRR*, abs/2211.14461, 2022b.

Zhao, Z., Xu, S., Zhang, J., Liang, C., Zhang, C., and Liu, J. Efficient and model-based infrared and visible image fusion via algorithm unrolling. *IEEE TCSVT*, 2022c.

Zhao, Z., Zhang, J., Xu, S., Lin, Z., and Pfister, H. Discrete cosine transform network for guided depth map super-resolution. In *CVPR*, 2022d.

Zhao, Z., Bai, H., Zhu, Y., Zhang, J., Xu, S., Zhang, Y., Zhang, K., Meng, D., Timofte, R., and Gool, L. V. DDFM: denoising diffusion model for multi-modality image fusion. 2023a.

Zhao, Z., Zhang, J., Gu, X., Tan, C., Xu, S., Zhang, Y., Timofte, R., and Gool, L. V. Spherical space feature decomposition for guided depth map super-resolution. *CoRR*, abs/2303.08942, 2023b.

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.

Appendix for BiBench

A. Details and Discussions of Benchmark

A.1. Details of Binarization Algorithm

Model compression methods including quantization (Xiao et al., 2023; Liu et al., 2021a; Guo et al., 2020a; Qin et al., 2022b; J. Guo, W. Ouyang, and D. Xu, 2020; Qin et al., 2023a; Chen et al., 2022; Qin, 2021; Zhang et al., 2021a; Guo et al., 2023b; Liu et al., 2022a; 2019) have been widely used in various deep learning fields (Wang, 2021; Zhao et al., 2020b; 2023a; 2020a; Guo et al., 2023a; Zhao et al., 2023b; Guo et al., 2021; Liu et al., 2023), including computer vision (Guo et al., 2020b; Wang et al., 2021a; Ding et al., 2022; Zhao et al., 2022d; Wang et al., 2021b; Zhao et al., 2022b; Wang et al., 2022a; Yin et al., 2021; Zhao et al., 2022c), language understanding (Bai et al., 2021; Wang et al., 2022b), speech recognition (Qin et al., 2023b), etc. Previous research has deemed lower bit-width quantization methods as more aggressive (Rusci et al., 2020; Choukroun et al., 2019; Qin et al., 2022a), as they often provide higher compression and faster processing at the cost of lower accuracy. Among all quantization techniques, 1-bit quantization (binarization) is considered the most aggressive (Qin et al., 2022a), as it poses significant accuracy challenges but offers the greatest compression and speed benefits.

Training. When training a binarized model, the sign function is commonly used in the forward pass, and gradient approximations such as STE are applied during the backward pass to enable the model to be trained. Since the parameters are quantized to binary, network binarization methods typically use a simple sign function as the quantizer rather than using the same quantizer as in multi-bit (2-8 bit) quantization (Gong et al., 2019; Gholami et al., 2021). Specifically, as (Gong et al., 2019) describes, for multi-bit uniform quantization, given the bit width b and the floating-point activation/weight x following in the range (l, u), the complete quantization-dequantization process of uniform quantization can be defined as

$$Q_U(\boldsymbol{x}) = \text{round}\left(\frac{\boldsymbol{x}}{\Delta}\right)\Delta, \tag{10}$$

where the original range (l, u) is divided into $2^b - 1$ intervals Pi, $i \in (0, 1, \cdots, 2^b - 1)$, and $\Delta = \frac{u-l}{2^b-1}$ is the interval length. When $b = 1$, the $Q_U(\boldsymbol{x})$ equals the sign function, and the binary function is expressed as

$$Q_B(\boldsymbol{x}) = \text{sign}(\boldsymbol{x}). \tag{11}$$

Therefore, binarization can be regarded formally as the 1-bit specialization of quantization.

Deployment. For efficient deployment on real-world hardware, binarized parameters are grouped in blocks of 32 and processed simultaneously using 32-bit instructions, which is the key principle for achieving acceleration. To compress binary algorithms, instructions such as XNOR (or the combination of EOR and NOT) and popcount are used to enable the deployment of binarized networks on real-world hardware. The XNOR (exclusive-XOR) gate is a combination of an XOR gate and an inverter, and XOR (also known as EOR) is a common instruction that has long been available in assembly instructions for all target platforms. The popcount instruction, or Population Count per byte, counts the number of bits with a specific value in each vector element in the source register, stores the result in a vector and writes it to the destination register (Arm, 2020). This instruction is used to accelerate the inference of binarized networks (Hubara et al., 2016; Rastegari et al., 2016) and is widely supported by various hardware, such as the definitions of popcount in ARM and x86 in (Arm, 2020) and (AMD, 2022), respectively.

Comparison with other compression techniques. Current network compression technologies primarily focus on reducing the size and computation of full-precision models. Knowledge distillation, for instance, guides the training of small (student) models using the intermediate features and/or soft outputs of large (teacher) models (Hinton et al., 2015; Xu et al., 2018; Chen et al., 2018; Yim et al., 2017; Zagoruyko & Komodakis, 2017). Model pruning (Han et al., 2015; 2016; He et al., 2017) and low-rank decomposition (Denton et al., 2014; Lebedev et al., 2015; Jaderberg et al., 2014; Lebedev & Lempitsky, 2016) also reduce network parameters and computation via pruning and low-rank approximation. Compact model design, on the other hand, creates a compact model from scratch (Howard et al., 2017; Sandler et al., 2018; Zhang et al., 2018b; Ma et al., 2018). While these compression techniques effectively decrease the number of parameters, the compressed models still use 32-bit floating-point numbers, which leaves scope for additional compression using model quantization/binarization techniques. Compared to multi-bit (2-8 bit) model quantization, which compresses parameters to integers(Gong et al., 2014; Wu et al., 2016; Vanhoucke et al., 2011; Gupta et al., 2015), binarization directly applies the sign function to compress the

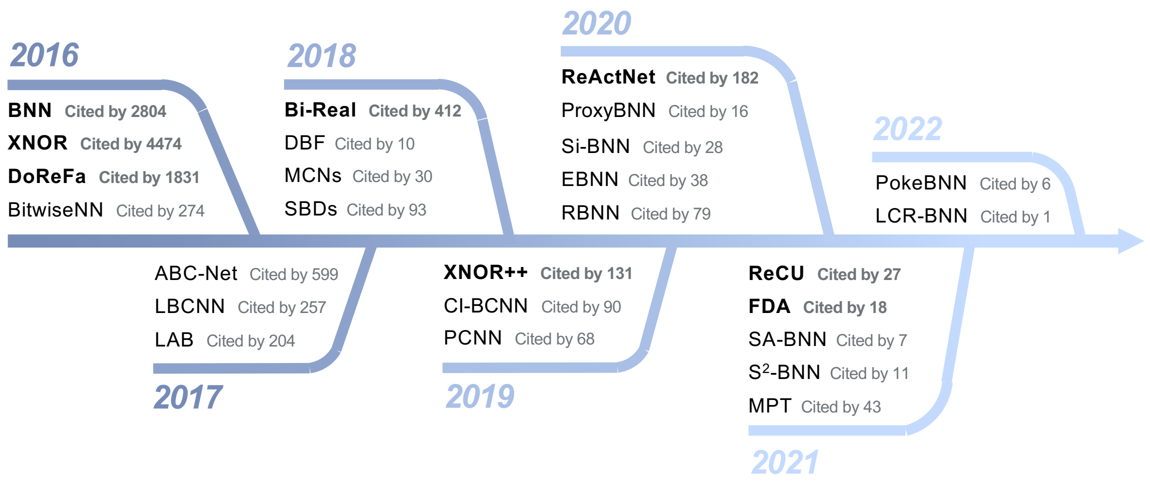


Figure 4: Timeline of the operator-level binarization algorithms we have considered. The algorithms selected for BiBench are in bold, and the citation is counted till 2023/01/25.

model to a more compact 1-bit (Rusci et al., 2020; Choukroun et al., 2019; Qin et al., 2022a; Shang et al., 2022b; Qin et al., 2020b). Additionally, due to the use of binary parameters, bitwise operations (XNOR and popcount) can be applied during inference at deployment instead of integer multiply-add operations in 2-8 bit model quantization. As a result, binarization is considered to be more hardware-efficient and can achieve greater speedup than multi-bit quantization.

Selection Rules:

When creating the BiBench, we considered various binarization algorithms with enhanced operator techniques in binarization research, and the timeline of considered algorithms is Figure 4 and we list their details in Table 5. We follow two general rules when selecting algorithms for our benchmark:

(1) **The selected algorithms should function on binarized operators** that are the fundamental components for binarized networks (as discussed in Section 2.1). We exclude algorithms and techniques that require specific local structures or training pipelines to ensure a fair comparison.

(2) **The selected algorithms should have an extensive influence to be representative**, *i.e.*, selected from widely adopted algorithms or the most advanced ones.

Specifically, we chose algorithms based on the following detailed criteria to ensure representativeness and fairness in evaluations: Operator Techniques (Yes/No), Year, Conference, Citations (up to 2023/01/25), Open source availability (Yes/No), and Specific Structure / Training-pipeline requirements (Yes/No/Optional).

We analyze the techniques proposed in these works. Following the general rules we mentioned, all considered binarization algorithms should have significant contributions to the improvement of the binarization operator (Operator Techniques: Yes) and should not include techniques that are bound to specific architectures and training pipelines to complete well all the evaluations of the learning task, neural architecture, and training consumption tracks in BiBench (Specified Structure / Training-pipeline: No/Optional, Optional means the techniques are included but can be decoupled with binarized operator totally). We also consider the impact and reproducibility of these works. We prioritized the selection of works with more than 100 citations, which means they are more discussed and compared in binarization research and thus have higher impacts. Works in 2021 and later are regarded as the SOTA binarization algorithms and prioritized. Furthermore, we hope the selected works have official open-source implementations for reproducibility.

Based on the above selections, eight binarization algorithms, *i.e.*, BNN, XNOR-Net, DoReFa-Net, Bi-Real Net, XNOR-Net++, ReActNet, FDA, and ReCU, stand out and are fully evaluated by our BiBench.

Algorithm Details:

BNN (Courbariaux et al., 2016b): During the training process, BNN uses the straight-through estimator (STE) to calculate gradient $\boldsymbol{g_x}$ which takes into account the saturation effect:

Table 5: The considered operator-level binarization algorithms and our final selections in BiBench. Bold means that the algorithm has an advantage in that column.

Algorithm	Year	Conference	Citation (2023/01/25)	Operator Techniques	Open Source	Specified Structure / Training-pipeline
BitwiseNN (Kim & Smaragdis, 2016)	2016	ICMLW	**274**	**Yes**	No	**No**
DoReFa (Zhou et al., 2016)	2016	ArXiv	**1831**	**Yes**	**Yes**	**No**
XNOR-Net (Rastegari et al., 2016)	2016	ECCV	**4474**	**Yes**	**Yes**	**No**
BNN (Courbariaux et al., 2016a)	2016	NeurIPS	**2804**	**Yes**	**Yes**	**No**
LBCNN (Juefei-Xu et al., 2017)	2017	CVPR	**257**	**Yes**	**Yes**	Yes
LAB (Hou et al., 2017)	2017	ICLR	**204**	**Yes**	**Yes**	Yes
ABC-Net (Lin et al., 2017)	2017	NeurIPS	**599**	**Yes**	**Yes**	Yes
DBF (Tseng et al., 2018)	2018	IJCAI	10	**Yes**	No	Yes
MCNs (Wang et al., 2018b)	2018	CVPR	30	**Yes**	No	Yes
SBDs (Hu et al., 2018)	2018	ECCV	93	**Yes**	No	**No**
Bi-Real Net (Liu et al., 2018a)	2018	ECCV	**412**	**Yes**	**Yes**	**Opt**
PCNN (Gu et al., 2019)	2019	AAAI	68	**Yes**	No	Yes
CI-BCNN (Wang et al., 2019)	2019	CVPR	90	**Yes**	**Yes**	Yes
XNOR-Net++ (Bulat et al., 2019)	2019	BMVC	**131**	**Yes**	**Yes**	**No**
ProxyBNN (He et al., 2020)	2020	ECCV	16	**Yes**	No	Yes
Si-BNN (Wang et al., 2020a)	2020	AAAI	28	**Yes**	No	**No**
EBNN (Bulat et al., 2020)	2020	ICLR	38	**Yes**	**Yes**	Yes
RBNN (Lin et al., 2020)	2020	NeurIPS	79	**Yes**	**Yes**	**No**
ReActNet (Liu et al., 2020)	2020	ECCV	**182**	**Yes**	**Yes**	**Opt**
SA-BNN (Liu et al., 2021b)	**2021**	AAAI	7	**Yes**	No	**No**
S^2-BNN (Shen et al., 2021)	**2021**	CVPR	11	**Yes**	**Yes**	Yes
MPT (Diffenderfer & Kailkhura, 2021)	**2021**	ICLR	43	**Yes**	**Yes**	Yes
FDA (Xu et al., 2021a)	**2021**	NeurIPS	18	**Yes**	**Yes**	**No**
ReCU (Xu et al., 2021b)	**2021**	ICCV	27	**Yes**	**Yes**	**No**
LCR-BNN (Shang et al., 2022a)	**2022**	ECCV	1	**Yes**	**Yes**	Yes
PokeBNN (Zhang et al., 2022b)	**2022**	CVPR	6	**Yes**	**Yes**	Yes

$$\texttt{sign}(\boldsymbol{x}) = \begin{cases} +1, & \text{if } \boldsymbol{x} \geq 0 \\ -1, & \text{otherwise} \end{cases} \qquad \boldsymbol{g_x} = \begin{cases} \boldsymbol{g_b}, & \text{if } \boldsymbol{x} \in (-1,1) \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

And during inference, the computation process is expressed as

$$\boldsymbol{o} = \operatorname{sign}(\boldsymbol{a}) \circledast \operatorname{sign}(\boldsymbol{w}), \tag{13}$$

where $\circledast$ indicates a convolutional operation using XNOR and bitcount operations.

XNOR-Net (Rastegari et al., 2016): XNOR-Net obtains the channel-wise scaling factors $\boldsymbol{\alpha} = \frac{\|\boldsymbol{w}\|}{|\boldsymbol{w}|}$ for the weight and $\boldsymbol{K}$ contains scaling factors β for all sub-tensors in activation $\boldsymbol{a}$. We can approximate the convolution between activation $\boldsymbol{a}$ and weight $\boldsymbol{w}$ mainly using binary operations:

$$\boldsymbol{o} = (\operatorname{sign}(\boldsymbol{a}) \circledast \operatorname{sign}(\boldsymbol{w})) \odot \boldsymbol{K}\boldsymbol{\alpha}, \tag{14}$$

where $\boldsymbol{w} \in \mathbb{R}^{c\times w\times h}$ and $\boldsymbol{a} \in \mathbb{R}^{c\times w_{\text{in}}\times h_{\text{in}}}$ denote the weight and input tensor, respectively. And the STE is also applied in the backward propagation of the training process.

DoReFa-Net (Zhou et al., 2016): DoReFa-Net applies the following function for 1-bit weight and activation:

$$\boldsymbol{o}=(\operatorname{sign}(\boldsymbol{a})\circledast\operatorname{sign}(\boldsymbol{w}))\odot\boldsymbol{\alpha}. \tag{15}$$

And the STE is also applied in the backward propagation with the full-precision gradient.

Bi-Real Net (Liu et al., 2018b): Bi-Real Net proposes a piece-wise polynomial function as the gradient approximation function:

$$\operatorname{bireal}(\boldsymbol{a})=\begin{cases}-1 & \text{if } \boldsymbol{a}<-1\\ 2\boldsymbol{a}+\boldsymbol{a}^2 & \text{if } -1\leqslant\boldsymbol{a}<0\\ 2\boldsymbol{a}-\boldsymbol{a}^2 & \text{if } 0\leqslant\boldsymbol{a}<1\\ 1 & \text{otherwise}\end{cases},\quad \frac{\partial\operatorname{bireal}(\boldsymbol{a})}{\partial\boldsymbol{a}}=\begin{cases}2+2\boldsymbol{a} & \text{if } -1\leqslant\boldsymbol{a}<0\\ 2-2\boldsymbol{a} & \text{if } 0\leqslant\boldsymbol{a}<1\\ 0 & \text{otherwise}\end{cases}. \tag{16}$$

And the forward propagation of Bi-Real Net is the same as Eq. (15).

XNOR-Net++ (Bulat et al., 2019): XNOR-Net++ proposes to re-formulate Eq. (14) as:

$$\boldsymbol{o}=(\operatorname{sign}(\boldsymbol{a})\circledast\operatorname{sign}(\boldsymbol{w}))\odot\boldsymbol{\Gamma}, \tag{17}$$

and we adopt the $\boldsymbol{\Gamma}$ as the following form in experiments (achieve the best performance in the original paper):

$$\boldsymbol{\Gamma}=\boldsymbol{\alpha}\otimes\boldsymbol{\beta}\otimes\boldsymbol{\gamma},\quad \boldsymbol{\alpha}\in\mathbb{R}^{\boldsymbol{o}},\boldsymbol{\beta}\in\mathbb{R}^{h_{\text{out}}},\boldsymbol{\gamma}\in\mathbb{R}^{w_{\text{out}}}, \tag{18}$$

where $\otimes$ denotes the outer product operation, and $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ are learnable during training.

ReActNet (Liu et al., 2020): ReActNet defines an RSign as a binarization function with channel-wise learnable thresholds:

$$\boldsymbol{x}=\operatorname{rsign}(\boldsymbol{x})=\begin{cases}+1, & \text{if } \boldsymbol{x}>\boldsymbol{\alpha}\\ -1, & \text{if } \boldsymbol{x}\leq\boldsymbol{\alpha}\end{cases}. \tag{19}$$

where $\boldsymbol{\alpha}$ is a learnable coefficient controlling the threshold. And the forward propagation is

$$\boldsymbol{o}=(\operatorname{rsign}(\boldsymbol{a})\circledast\operatorname{sign}(\boldsymbol{w}))\odot\boldsymbol{\alpha}. \tag{20}$$

where $\boldsymbol{\alpha}$ denotes the channel-wise scaling factors of weights. Parameters in RSign are optimized end-to-end with other parameters in the network. The gradient of $\boldsymbol{\tau}$ in RSign can be simply derived by the chain rule as:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\tau}}=\frac{\partial\mathcal{L}}{\partial h(\boldsymbol{x})}\frac{\partial h(\boldsymbol{x})}{\partial\boldsymbol{\tau}}, \tag{21}$$

where $\mathcal{L}$ represents the loss function and $\frac{\partial\mathcal{L}}{\partial h(\boldsymbol{x})}$ denotes the gradients from deeper layers. The derivative $\frac{\partial h(\boldsymbol{x})}{\partial\boldsymbol{\tau}}$ can be easily computed as

$$\frac{\partial h(\boldsymbol{x})}{\partial\boldsymbol{\tau}}=-1. \tag{22}$$

The estimator of the sign function for activation is

$$\operatorname{react}(\boldsymbol{a})=\begin{cases}-1 & \text{if } \boldsymbol{a}<-1;\\ 2\boldsymbol{a}+\boldsymbol{a}^2 & \text{if } -1\leq\boldsymbol{a}<0;\\ 2\boldsymbol{a}-\boldsymbol{a}^2; & \text{if } 0\leq\boldsymbol{a}<1;\\ 1 & \text{otherwise,}\end{cases} \tag{23}$$

$$\frac{\partial \operatorname{react}(\boldsymbol{a})}{\partial \boldsymbol{a}}=\begin{cases}2+2\boldsymbol{a} & \text{if } -1 \leq \boldsymbol{a}<0; \\ 2-2\boldsymbol{a}; & \text{if } 0 \leq \boldsymbol{a}<1; \\ 0 & \text{otherwise.}\end{cases} \tag{24}$$

And the estimator for weight is the STE form straightforwardly as Eq. (12).

ReCU (Xu et al., 2021b): As described in their paper, ReCU is formulated as

$$\operatorname{recu}(\boldsymbol{w})=\max\left(\min\left(\boldsymbol{w}, Q_{(\tau)}\right), Q_{(1-\tau)}\right), \tag{25}$$

where $Q_{(\tau)}$ and $Q_{(1-\tau)}$ denote the τ quantile and $1-\tau$ quantile of $\boldsymbol{w}$, respectively. After applying balancing, weight standardization, and the ReCU to $\boldsymbol{w}$, the generalized probability density function of $\boldsymbol{w}$ can be written as follows

$$f(w)=\begin{cases}\frac{1}{2b}\exp\left(\frac{-|\boldsymbol{w}|}{b}\right), & \text{if } |\boldsymbol{w}|<Q(\tau); \\ 1-\tau, & \text{if } |\boldsymbol{w}|=Q(\tau); \\ 0, & \text{otherwise,}\end{cases} \tag{26}$$

where b is obtained via

$$b=\operatorname{mean}(|\boldsymbol{w}|), \tag{27}$$

where $\operatorname{mean}(|\cdot|)$ returns the mean of the absolute values of the inputs. The gradient of the weights is obtained by applying the chain derivation rule to the above equation, where the estimator applied to the sign function is STE. As for the gradient w.r.t. the activations, ReCU considers the piecewise polynomial function as follows

$$\frac{\partial L}{\partial \operatorname{sign}(\boldsymbol{a})}=\frac{\partial L}{\partial \operatorname{sign}(\boldsymbol{a})}\cdot\frac{\partial \operatorname{sign}(\boldsymbol{a})}{\partial \boldsymbol{a}}\approx\frac{\partial L}{\partial \operatorname{sign}(\boldsymbol{a})}\cdot\frac{\partial F(\boldsymbol{a})}{\partial \boldsymbol{a}}, \tag{28}$$

where

$$\frac{\partial F(\boldsymbol{a})}{\partial \boldsymbol{a}}=\begin{cases}2+2\boldsymbol{a}, & \text{if } -1 \leq \boldsymbol{a}<0; \\ 2-2\boldsymbol{a}, & \text{if } 0 \leq \boldsymbol{a}<1; \\ 0, & \text{otherwise .}\end{cases} \tag{29}$$

And other implementations also strictly follow the original paper and official code.

FDA (Xu et al., 2021a): FDA applies the following forward propagation in the operator:

$$\boldsymbol{o}=(\operatorname{rsign}(\boldsymbol{a})\circledast\operatorname{sign}(\boldsymbol{w}))\odot\boldsymbol{\alpha}, \tag{30}$$

and computes the gradients of $\boldsymbol{o}$ in the backward propagation as:

$$\frac{\partial \ell}{\partial \mathbf{t}}=\frac{\partial \ell}{\partial \boldsymbol{o}}\boldsymbol{w}_2^{\top}\odot\left(\left(\mathbf{t}\boldsymbol{w}_1\right)\geq 0\right)\boldsymbol{w}_1^{\top}+\frac{\partial \ell}{\partial \boldsymbol{o}}\eta'(\mathbf{t})+\frac{\partial \ell}{\partial \boldsymbol{o}}\odot\frac{4\omega}{\pi}\sum_i\cos((2i+1)\omega\mathbf{t}), \tag{31}$$

where $\frac{\partial \ell}{\partial \boldsymbol{o}}$ is the gradient from the upper layers, $\odot$ represents element-wise multiplication, and $\frac{\partial \ell}{\partial \mathbf{t}}$ is the partial gradient on $\mathbf{t}$ that backward propagates to the former layer. And $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are weights in the original models and the noise adaptation modules respectively. FDA updates them as

$$\frac{\partial \ell}{\partial \boldsymbol{w}_1} = \mathbf{t}^\top \frac{\partial \ell}{\partial \boldsymbol{o}} \boldsymbol{w}_2^\top \odot \left((\mathbf{t}\boldsymbol{w}_1) \geq 0 \right), \qquad \frac{\partial \ell}{\partial \boldsymbol{w}_2} = \sigma \left(\mathbf{t}\boldsymbol{w}_1 \right)^\top \frac{\partial \ell}{\partial \boldsymbol{o}}. \tag{32}$$

In Table 6, we detail the techniques included in the selected binarization algorithms.

Table 6: The details of the selected binarization algorithms in BiBench.

Algorithm	Scaling Factor		Parameter Redistribution		Gradient Approximation	
	weight	activation	weight	activation	weight	activation
BNN	w/o	w/o	w/o	w/o	STE	STE
XNOR	Statistics by Channel	Statistics by Channel	w/o	w/o	STE	STE
DoReFa	Statistics by Layer	w/o	w/o	w/o	STE	STE
Bi-Real	Statistics by Channel	w/o	w/o	w/o	STE	Differentiable Piecewise Polynomial Function
XNOR++	Learned by Custom-size ($o \times h_{\text{out}} \times w_{\text{out}}$)	w/o	w/o	w/o	STE	STE
ReActNet	Statistics by Channel	w/o	w/o	w/o	STE	Differentiable Piecewise Polynomial Function
ReCU	Statistics by Channel	w/o	balancing (mean-shifting)	w/o	Rectified Clamp Unit	Rectified Clamp Unit
FDA	Statistics by Channel	w/o	w/o	mean-shifting	Decomposing Sign with Fourier Series	Decomposing Sign with Fourier Series

[1] "STE" indicates the Straight Through Estimator, and "w/o" means no special technique is used.

A.2. Details of Learning Tasks

Selection Rules:

To evaluate the performance of the binarization algorithm in a wide range of learning tasks, we must select a variety of representative tasks. Firstly, we choose representative perception modalities in deep learning, including (2D/3D) vision, text, and speech. These modalities have seen rapid progress and broad impact, so we select specific tasks and datasets within these modalities. Specifically, (1) in the 2D vision modality, we select the essential image classification task and the prevalent object detection task, with datasets including CIFAR10 and ImageNet for the former and Pascal VOC and COCO for the latter. ImageNet and COCO datasets are more challenging and significant, while CIFAR10 and Pascal VOC are more fundamental. For other modalities, binarization is still challenging even with the underlying tasks and datasets in the field, since there are few related binarization studies: (2) in the 3D vision modality, the basic point cloud classification ModelNet40 dataset is selected to evaluate the binarization performance, which is regarded as one of the most fundamental tasks in 3D point cloud research and is widely studied. (3) In the text modality, the General Language Understanding Evaluation (GLUE) benchmark is usually recognized as the most popular dataset, including nine sentence- or sentence-pair language understanding tasks. (4) The keyword spotting task was chosen as the base task in the speech modality, specifically the Google Speech Commands classification dataset.

Based on the above reasons and rules, we have selected a series of challenging and representative tasks for BiBench to evaluate binarization comprehensively and have obtained a series of reliable and informative conclusions and experiences.

Dataset Details:

CIFAR10 (Krizhevsky et al., 2014): The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images commonly used to train machine learning and computer vision algorithms. This dataset is widely used for image classification tasks. There are 60,000 color images, each of which measures 32x32 pixels. All images are categorized into 10 different classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each class has 6000 images, where 5000 are for training and 1000 are for testing. The evaluation metric of the CIFAR-10 dataset is accuracy, defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{33}$$

where *TP* (True Positive) means cases correctly identified as positive, *TN* (True Negative) means cases correctly identified as negative, *FP* (False Positive) means cases incorrectly identified as positive and *FN* (False Negative) means cases incorrectly identified as negative. To estimate the accuracy, we should calculate the proportion of *TP* and *TN* in all evaluated cases.

ImageNet (Krizhevsky et al., 2012): ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. The images are collected from the web and labeled by human labelers using a crowd-sourced image labeling service called Amazon Mechanical Turk. As part of the Pascal Visual Object Challenge, ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) was established in 2010. There are approximately 1.2 million training images, 50,000 validation images, and 150,000 testing images in total in ILSVRC. ILSVRC uses a subset of ImageNet, with about 1000 images in each of the 1000 categories.

ImageNet also uses accuracy to evaluate the predicted results, which is defined above.

Pascal VOC07 (Hoiem et al., 2009): The PASCAL Visual Object Classes 2007 (VOC07) dataset contains 20 object categories including vehicles, households, animals, and other: airplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, TV/monitor, bird, cat, cow, dog, horse, sheep, and person. As a benchmark for object detection, semantic segmentation, and object classification, this dataset contains pixel-level segmentation annotations, bounding box annotations, and object class annotations. The VOC07 dataset uses mean average precision(mAP) to evaluate results, which is defined as:

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k \tag{34}$$

where AP_k denotes the average precision of the k-th category, which calculates the area under the precision-recall curve:

$$AP_k = \int_0^1 p_k(r)dr. \tag{35}$$

Especially for VOC07, we apply 11-point interpolated AP, which divides the recall value to $\{0.0, 0.1, \ldots, 1.0\}$ and then computes the average of maximum precision value for these 11 recall values as:

$$AP = \frac{1}{11}\sum_{r\in\{0.0,\ldots,1.0\}} AP_r = \frac{1}{11}\sum_{r\in\{0.0,\ldots,1.0\}} p_{\text{interp}} r. \tag{36}$$

The maximum precision value equals to the right of its recall level:

$$p_{\text{interp}} r = \max_{\tilde{r}\geq r} p(\tilde{r}). \tag{37}$$

COCO17 (Lin et al., 2014): The MS COCO (Microsoft Common Objects in Context) dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images. According to community feedback, in the 2017 release, the training/validation split was changed from 83K/41K to 118K/5K. And the images and annotations are the same. The 2017 test set is a subset of 41K images from the 2015 test set. Additionally, 123K images are included in the unannotated dataset. The COCO17 dataset also uses mean average precision (mAP) as defined above PASCAL VOC07 uses, which is defined as above.

ModelNet40 (Wu et al., 2015): The ModelNet40 dataset contains point clouds of synthetic objects. As the most widely used benchmark for point cloud analysis, ModelNet40 is popular due to the diversity of categories, clean shapes, and well-constructed dataset. In the original ModelNet40, 12,311 CAD-generated meshes are divided into 40 categories, where 9,843 are for training, and 2,468 are for testing. The point cloud data points are sampled by a uniform sampling method from mesh surfaces and then scaled into a unit sphere by moving to the origin. The ModelNet40 dataset also uses accuracy as the metric, which has been defined above in CIFAR10.

ShapeNet (Chang et al., 2015): ShapeNet is a large-scale repository for 3D CAD models developed by researchers from Stanford University, Princeton University, and the Toyota Technological Institute in Chicago, USA.

Using WordNet hypernym-hyponym relationships, the repository contains over 300M models, with 220,000 classified into 3,135 classes. There are 31,693 meshes in the ShapeNet Parts subset, divided into 16 categories of objects (*i.e.*, tables, chairs, planes, *etc.*). Each shape contains 2-5 parts (with 50 part classes in total).

GLUE (Wang et al., 2018a): General Language Understanding Evaluation (GLUE) benchmark is a collection of nine natural language understanding tasks, including single-sentence tasks CoLA and SST-2, similarity and paraphrasing tasks MRPC, STS-B and QQP, and natural language inference tasks MNLI, QNLI, RTE, and WNLI. Among them, SST-2, MRPC, QQP, MNLI, QNLI, RTE, and WNLI use accuracy as the metric, which is defined in CIFAR10. CoLA is measured by Matthews Correlation Coefficient (MCC), which is better in binary classification since the number of positive and negative samples are extremely unbalanced:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}. \tag{38}$$

And STS-B is measured by Pearson/Spearman Correlation Coefficient:

$$r_{\textit{Pearson}} = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{X_i - \bar{X}}{s_X}\right)\left(\frac{Y_i - \bar{Y}}{s_Y}\right), r_{\textit{Spearman}} = 1 - \frac{6\sum d_i^2}{n(n^2-1)}, \tag{39}$$

where n is the number of observations, s_X and s_Y indicate the sum of squares of X and Y respectively, and d_i is the difference between the ranks of corresponding variables.

SpeechCom. (Warden, 2018): As part of its training and evaluation process, SpeechCom provides a collection of audio recordings containing spoken words. Its primary goal is to provide a way to build and test small models that detect a single word that belongs to a set of ten target words. Models should detect as few false positives as possible from background noise or unrelated speech while providing as few false positives as possible. The accuracy metric for SpeechCom is also the same as CIFAR10.

CIFAR10-C (Hendrycks & Dietterich, 2018): CIFAR10-C is a dataset generated by adding 15 common corruptions and 4 extra corruptions to the test images in the Cifar10 dataset. It benchmarks the frailty of classifiers under corruption, including noise, blur, weather, and digital influence. And each type of corruption has five levels of severity, resulting in 75 distinct corruptions. We report the accuracy of the classifiers under each level of severity and each corruption. Meanwhile, we use the mean and relative corruption error as metrics. Denote the error rate of *Network* under *Settings* as $E^{\textit{Network}}_{\textit{Settings}}$. The classifier's aggregate performance across the five severities of the corruption types. The Corruption Errors of a certain type of *Corruption* is computed with the formula:

$$CE^{\textit{Network}}_{\textit{Corruption}} = \sum_{s=1}^{5} E^{\textit{Network}}_{s,\textit{Corruption}} / \sum_{s=1}^{5} E^{\textit{AlexNet}}_{s,\textit{Corruption}}. \tag{40}$$

To make Corruption Errors comparable across corruption types, the difficulty is usually adjusted by dividing by AlexNet's errors.

A.3. Details of Neural Architectures

ResNet (He et al., 2016): Residual Networks, or ResNets, learn residual functions concerning the layer inputs instead of learning unreferenced functions. Instead of making stacked layers directly fit a desired underlying mapping, residual nets let these layers fit a residual mapping. There is empirical evidence that these networks are easier to optimize and can achieve higher accuracy with considerably increased depth.

VGG (Simonyan & Zisserman, 2015): VGG is a classical convolutional neural network architecture. It is proposed by an analysis of how to increase the depth of such networks. It is characterized by its simplicity: the network utilizes small 3×3 filters, and the only other components are pooling layers and a fully connected layer.

MobileNetV2 (Sandler et al., 2018): MobileNetV2 is a convolutional neural network architecture that performs well on mobile devices. This model has an inverted residual structure with residual connections between the bottleneck layers. The intermediate expansion layer employs lightweight depthwise convolutions to filter features as a source of nonlinearity. In MobileNetV2, the architecture begins with an initial layer of 32 convolution filters, followed by 19 residual bottleneck layers.

Faster-RCNN (Ren et al., 2015): Faster R-CNN is an object detection model that improves Fast R-CNN by utilizing a region proposal network (RPN) with the CNN model. The RPN shares full-image convolutional features with the detection network, enabling nearly cost-free region proposals. A fully convolutional network is used to predict the bounds and objectness scores of objects at each position simultaneously. RPNs use end-to-end training to produce region proposals of high quality and instruct the unified network where to search. Sharing their convolutional features allows RPN and Fast R-CNN to be combined into a single network. Faster R-CNN consists of two modules. The first module is a deep, fully convolutional network that proposes regions, and the second is the detector that uses the proposals for giving the final prediction boxes.

SSD (Liu et al., 2016): SSD is a single-stage object detection method that discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. During prediction, each default box is adjusted to match better the shape of the object based on its scores for each object category. In addition, the network automatically handles objects of different sizes by combining predictions from multiple feature maps with different resolutions.

BERT (Kenton & Toutanova, 2019): BERT, or Bidirectional Encoder Representations from Transformers, improves upon standard Transformers by removing the unidirectionality constraint using a masked language model (MLM) pre-training objective. By masking some tokens from the input, the masked language model attempts to estimate the original vocabulary id of the masked word based solely on its context. An MLM objective differs from a left-to-right language model in that it enables the representation to integrate the left and right contexts, which facilitates pre-training a deep bidirectional Transformer. Additionally, BERT uses a next-sentence prediction task that pre-trains text-pair representations along with the masked language model. Note that we replace the direct binarized attention with a bi-attention mechanism to prevent the model from completely crashing (Qin et al., 2021).

PointNet (Qi et al., 2017): PointNet is a unified architecture for applications ranging from object classification and part segmentation to scene semantic parsing. The architecture directly receives point clouds as input and outputs either class labels for the entire input or point segment/part labels. **PointNet-Vanilla** is a variant of PointNet, which drops off the T-Net module. And for all PointNet models, we apply the EMA-Max (Qin et al., 2020a) as the aggregator, because directly following the max pooling aggregator will cause the binarized PointNets to fail to converge.

FSMN (Zhang et al., 2015): Feedforward sequential memory networks or FSMN is a novel neural network structure to model long-term dependency in time series without using recurrent feedback. It is a standard fully connected feedforward neural network containing some learnable memory blocks. As a short-term memory mechanism, the memory blocks encode long context information using a tapped-delay line structure.

Deep-FSMN (Zhang et al., 2018a): The Deep-FSMN architecture is an improved feedforward sequential memory network (FSMN) with skip connections between memory blocks in adjacent layers. By utilizing skip connections, information can be transferred across layers, and thus the gradient vanishing problem can be avoided when building very deep structures.

A.4. Details of Hardware

Hisilicon Kirin (Hisilicon, 2022): Kirin is a series of ARM-based systems-on-a-chip (SoCs) produced by HiSilicon. Their products include Kirin 970, Kirin 980, Kirin 985, *etc.*

MediaTek Dimensity (MediaTek, 2022): Dimensity is a series of ARM-based systems-on-a-chip (SoCs) produced by MediaTek. Their products include Dimensity 820, Dimensity 8100, Dimensity 9000, *etc.*

Qualcomm Snapdragon (Singh & Jain, 2014): Snapdragon is a family of mobile systems-on-a-chip (SoC) processor architecture provided by Qualcomm. The original Snapdragon chip, the Scorpio, was similar to the ARM Cortex-A8 core based upon the ARMv7 instruction set, but it was enhanced by the use of SIMD operations, which provided higher performance. Qualcomm Snapdragon processors are based on the Krait architecture. They are equipped with an integrated LTE modem, providing seamless connectivity across 2G and 3G LTE networks.

Raspberrypi (Wikipedia, 2022b): Raspberry Pi is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. Raspberry Pi was originally designed to promote the teaching of basic computer science in schools and in developing countries. As a result of its low cost, modularity, and open design, it is used in many applications, including weather monitoring, and is sold outside the intended market. It is typically used by computer hobbyists and electronic enthusiasts due to the adoption of HDMI and USB standards.

Apple M1 (Wikipedia, 2022a): Apple M1 is a series of ARM-based systems-on-a-chip (SoCs) designed by Apple Inc. As a central processing unit (CPU) and graphics processing unit (GPU) for Macintosh desktops and notebooks, as well as iPad Pro and iPad Air tablets. In November 2020, Apple introduced the M1 chip, followed by the professional-oriented M1 Pro and M1 Max chips in 2021. Apple launched the M1 Ultra in 2022, which combines two M1 Max chips in a single package. The M1 Max is a higher-performance version of the M1 Pro, with larger GPU cores and memory bandwidth.

A.5. Discussion of Novelty and Significance

We emphasize that our BiBench includes the following significant contributions: (1) the ***first*** systematic benchmark that enables a new view to quantitative evaluate binarization algorithms at the operator level, and (2) revealing a practical binarized operator design paradigm.

(1) BiBench is the first effort to facilitate systematic and comprehensive comparisons between binarized algorithms. It provides a brand new perspective to decouple the binarized operators from the architectures for quantitative evaluations at the operator level. In existing works, the binarized operator and specific structure/training pipeline are often designed simultaneously or coupled closely (up to 15 (in 26) algorithms in Table 5 of the manuscript), the latter is often difficult to generalize to various architectures and tasks. Their training and inference settings are also different from each other. This makes the direct comparison among binarization algorithms difficult. Our BiBench enables a new approach towards a fair comparison of binarization algorithms by building a unified evaluation track for each algorithm on learning tasks and neural architectures.

(2) BiBench reveals a practical paradigm of binarized operator designing. Based on the systemic and quantitative evaluation, superior techniques for better binarization operators can emerge, which is essential for pushing binarization algorithms to be accurate and efficient. For instance, after excluding the bi-real shortcut in Bi-Real Net, the difference between it and the DoReFa operator is solely the soft estimator in the backward propagation, yet this change yields a 2% difference in the learning task track (as Table 2 in the manuscript shows). These unprecedented quantitative insights identify which techniques are effective and low-cost for improving binarization operators, which will strongly facilitate the emergence of more powerful generic binarization algorithms. We summarize and present these operator-level binarization techniques in Section 5.3.

A.6. Discussion of Bit-width Constraints

In real practice, applying flexible bit-width or specialized quantizers in certain layers can lead to significant improvements in binarization, enabling a better balance between task performance and inference efficiency. There is a common practice to keep the input and output layers at 32-bit full precision since binarizing these two layers brings a severe task performance drop while having a relatively low efficiency gain (Rastegari et al., 2016; Liu et al., 2018a; Zhou et al., 2016). And we also follow this practice in our BiBench. Moreover, some binarization algorithms optimized for practical deployment also compress these layers while maintaining the task performance, they usually make the two layers fixed-point (*e.g.*, 8-bit) or design specific quantizers for them (Zhang et al., 2021b; Zhao et al., 2017; Courbariaux et al., 2016b). As mentioned by the reviewer, FracBNN (Zhang et al., 2021b) is one of the most representative works among them, which employs a dual-precision activation scheme to compute features with up to two bits using an additional sparse binary convolution. In practice, these algorithms can help BNNs achieve better accuracy-efficiency trade-offs.

We would also highlight that there is significant value in the exploration of full binarization (weight and activation) algorithms at the operator level, which is the focus of our BiBench. Since the application of the most aggressive bit-widths, the binarized parts in BNNs can be implemented by extremely efficient bitwise instructions, which makes it a more efficient solution compared to other bit-width quantization and partial binarization. Hence, the weight and activation binarization is adopted by various papers that propose binarization algorithms as a standard setting (Rastegari et al., 2016; Zhou et al., 2016; Liu

Table 7: Results of the structure binarization.

	Size\Dim	64	128	256	512	Mean
Attention (Token)	64	0.97	0.99	1.02	1.03	0.98
	128	0.97	0.98	1.01	1.03	
	256	0.94	0.96	0.99	1.01	
	512	0.89	0.92	0.97	0.99	
CNN (W×H)	14×14	0.59	0.59	0.59	0.59	0.59
	28×28	0.59	0.58	0.59	0.59	
	56×56	0.59	0.59	0.59	0.59	
	112×112	0.59	0.59	0.59	0.59	
MLP (#Point)	64	0.84	0.85	0.84	0.86	0.84
	128	0.84	0.84	0.85	0.84	
	256	0.84	0.84	0.84	0.84	
	512	0.84	0.84	0.84	0.84	

et al., 2018a; 2020). Therefore, our BiBench applies binarization operators to the entire network to allow us to fairly reflect the accuracy and efficiency of various binarization operators. Furthermore, the goal of BiBench is to systematically study various binarization algorithms through the results of each evaluation track (including learning tasks, neural architecture, *etc*.). So we only consider generic binarization techniques at the operator level when benchmarking, while techniques like special structural design and quantization for specific layers are excluded. We believe that our results provide valuable insights into the practical trade-offs and challenges of using binary activations, which can serve as a foundation for future research in this area.

A.7. Discussion of Binarized Architectures

To empirically verify our analysis in Section 5.1.2, we design the following experiments to demonstrate that the impact of different architectures on binarization is mainly caused by different local structures: (1) Local structures definition. We first abstract the typical local structures of CNN-based, transformer-based, and MLP-based architectures. The first is the bottleneck unit in ResNet, which is mainly composed of three 2D convolutions and a shortcut. The second is the self-attention structure in BERT, which is mainly composed of three linear units generating the query, key, and value, and two multiplication operations between activations (without weights). The last one is the sub-MLP structure in PointNet, which is mainly formed by stacking 3 linear units in series. All binarized structure examples unified contain three convolution or linear units to ensure consistency. (2) Initialization and binarization. To get the impact of the structure level on binarization, we randomly initialize the weights and inputs in all structures and then compare the differences between their full precision and binarized versions. We use the most basic BNN binarization to reveal essential effects. (3) Metric definition. We define 16 inputs of different sizes for each structure, and compare the average error caused by binarization on them:

$$E_f = \mathbb{E}_{\boldsymbol{x}\in\mathcal{X}}\left(\left|\frac{f(\boldsymbol{x})}{\mathrm{std}(f(\boldsymbol{x}))} - \frac{\hat{f}(\boldsymbol{x})}{\mathrm{std}(\hat{f}(\boldsymbol{x}))}\right|_{\ell 1}\right), \tag{41}$$

where $\boldsymbol{x}\in\mathcal{X}$ denotes the input set, $\|\cdot\|_{\ell 1}$ denotes L1 norm, and $f(\cdot)$ and $\hat{f}(\cdot)$ denote the full-precision and binarized structures, respectively. As shown in the results in Table 7, the transformer exhibits a larger binarization error at the local structure level. This empirically validates our analysis that binarization has a greater impact on the forward propagation of transformers with the attention mechanism compared to other architectures.

A.8. Discussion of Binarization Robustness

we first compare corruption robustness between full-precision architectures of different sizes to determine whether smaller model sizes necessarily lead to better robustness. We compare the full-precision ResNet-20 evaluated by BiBench. ResNet-20 has a consistent residual structure compared with ResNet-18, and the latter has a smaller parameter amount (about 1/40 of the former). However, our results in Table 8 counter-intuitively show that the robustness indicator of full-precision

Table 8: Robustness indicators of binarization algorithms and full-precision ResNet-20.

Arch	BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA	ResNet 20 (FP32)
CIFAR10-C	95.26	100.97	81.43	96.56	92.69	94.01	103.29	98.35	89.55

ResNet-20 is only 89.55 in terms of robustness, which is worse than not only up to 7 (out of 8) binarization methods we evaluated but also the larger-scale full-precision ResNet-18 (whose corresponding indicator is 100.00). This suggests that the reason for being robust to data corruption cannot be directly attributed to the smaller model scale, but is more likely to be a unique characteristic of some binarization algorithms. More detailed results are in Table 9.

Table 9: Original results of full-precision ResNet-20 on CIFAR10-C.

Origin	91.99	Overall	68.68																
gaussian_noise1	69.91	shot_noise1	80.41	gaussian_blur1	91.44	glass_blur1	47.34	zoo_blur1	81.48	fog1	91.71	snow1	84.26	elastic_transfor1	85.65	pixelate1	88.95	spatter1	88.02
gaussian_noise2	48.26	shot_noise2	68.26	gaussian_blur2	81.83	glass_blur2	48.49	zoo_blur2	76.91	fog2	89.92	snow2	70.81	elastic_transfor2	85.39	pixelate2	82.4	spatter2	80.3
gaussian_noise3	32.12	shot_noise3	44.64	gaussian_blur3	67.53	glass_blur3	51.63	zoo_blur3	69.56	fog3	87.18	snow3	75.67	elastic_transfor3	80.06	pixelate3	76.87	spatter3	76.17
gaussian_noise4	27.33	shot_noise4	37.75	gaussian_blur4	52.48	glass_blur4	40.26	zoo_blur4	62.59	fog4	82.64	snow4	72.39	elastic_transfor4	73.32	pixelate4	59.62	spatter4	79.82
gaussian_noise5	23.29	shot_noise5	28.72	gaussian_blur5	31.23	glass_blur5	41.77	zoo_blur5	52.02	fog5	61.65	snow5	64.81	elastic_transfor5	69.55	pixelate5	43.48	spatter5	69.63
ipulse_nosie1	80.6	speckle_noise1	80.58	defocus_blur1	91.39	otion_blur1	85.17	brighness1	91.78	frost1	85.77	contrast1	91.37	jpeg_copression1	83.15	saturate1	89.24		
ipulse_nosie2	67.92	speckle_noise2	63.85	defocus_blur2	89.24	otion_blur2	75.86	brighness2	91.27	frost2	78.21	contrast2	85.99	jpeg_copression2	77.7	saturate2	86.71		
ipulse_nosie3	58.22	speckle_noise3	56.01	defocus_blur3	81.91	otion_blur3	65.86	brighness3	90.52	frost3	65.87	contrast3	79.12	jpeg_copression3	75.65	saturate3	91.01		
ipulse_nosie4	43.37	speckle_noise4	44.78	defocus_blur4	70.09	otion_blur4	65.56	brighness4	89.4	frost4	63.54	contrast4	64.85	jpeg_copression4	72.6	saturate4	87.96		
ipulse_nosie5	31.27	speckle_noise5	34.8	defocus_blur5	47.18	otion_blur5	56.16	brighness5	86.63	frost5	51.87	contrast5	22.13	jpeg_copression5	69.08	saturate5	83.69		

B. Full Results

B.1. Evaluation Results of All Tracks

Table 10-11 shows the accuracy of different binarization algorithms on 2D and 3D vision tasks, including CIFAR10, ImageNet, PASCAL VOC07, COCO17 for 2D vision tasks and ModelNet40 for 3D vision task. And for each task, we cover several representative model architectures and binarize them with the binarization algorithms mentioned above.

We also evaluate binarization algorithms on language and speech tasks, for which we test TinyBERT (4 layers and 6 layers) on GLUE Benchmark and FSMN and D-FSMN on the SpeechCommand dataset. Results are listed in Table 12.

To demonstrate the robustness corruption of binarized algorithms, we show the results on the CIFAR10-C benchmark, which is used to benchmark the robustness to common perturbations in Table 13 and Table 14. It includes 15 kinds of noise, blur, weather, and digital corruption, each with five levels of severity.

The sensitivity of hyperparameters while training is shown in Table 15-16. For each binarization algorithm, we use SGD or Adam optimizer, $1\times$ or $0.1\times$ of the original learning rate, cosine or step learning scheduler, and 200 training epochs. Each case is tested five times to show the training stability. We also calculate the mean and standard deviation (std) of accuracy. The best accuracy and the lowest std for each binarization algorithm are bolded.

We conduct comprehensive deployment and inference on various kinds of hardware, including the Kirin series (970, 980, 985, 990, and 9000E), Dimensity series (820 and 9000), Snapdragon series (855+, 870 and 888), Raspberrypi (3B+ and 4B) and Apple M1 series (M1 and M1 Max). Limited to the support of frameworks, we can only test BNN and ReAct with Larq compute engine and only BNN with daBNN. We convert models to enable the actual inference on real hardware, including ResNet18/34 and VGG-Small on Larq, and only ResNet18/34 on daBNN. And we test 1, 2, 4, and 8 threads for each hardware and additionally test 16 threads for Apple Silicons on Larq. And daBNN only supports single-thread inference. Results are showcased in Table 17-20.

B.2. Comparison Results against Other Compression Technologies

We further evaluated representative multi-bit quantization algorithms (Zhou et al., 2016; Choi et al., 2018; Esser et al., 2019) (with INT2 and INT8) and dropout (pruning) algorithms (Li et al., 2016; Ding et al., 2021) in the limited time to demonstrate their accuracy and efficiency metrics and compare them to network binarization. The results show that compared with multi-bit quantization and dropout, binarization brings more significant compression and acceleration while facing greater challenges from the decline in accuracy.

To highlight the characteristics of network binarization, we compare it with other mainstream compression approaches, including multi-bit quantization and pruning, from accuracy and efficiency perspectives (Table 21). Overall, the results

Table 10: Accuracy on 2D and 3D Vision Tasks.

Task	Arch.	FP32	Binarization Algorithm							
			BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA
CIFAR10	ResNet20	91.99	85.31	85.53	85.18	85.56	85.41	86.18	86.42	86.38
	ResNet18	94.82	89.69	91.4	91.55	91.20	90.04	91.55	92.79	90.42
	ResNet34	95.34	90.82	89.58	90.95	92.50	90.59	92.69	93.64	89.59
	VGG-Small	93.80	89.66	89.65	89.66	90.25	89.34	90.27	90.84	89.48
ImageNet	ResNet18	69.90	52.99	53.99	53.55	54.79	52.43	54.97	54.51	54.63
VOC07	Faster-RCNN	76.06	58.54	56.75	58.07	60.90	56.60	61.90	62.10	60.10
	SSD300	77.34	9.09	33.72	30.70	31.90	9.41	38.41	9.80	43.68
COCO17	Faster-RCNN	27.20	21.20	20.50	21.30	22.20	21.60	22.80	23.30	22.40
ModelNet40	PointNet$_{\text{vanilla}}$	86.80	85.13	83.47	85.21	85.37	85.66	85.13	85.21	85.49
	PointNet	88.20	9.08	80.75	78.77	77.71	63.25	76.50	81.12	79.62

express the intuitive conclusion that there is a trade-off between accuracy and efficiency for different compression approaches. The ultra-low bit-width of network binarization brings acceleration and compression beyond multi-bit quantization and pruning. For example, binarization achieves 12.32× FLOPs saving, while INT8 quantization achieves 1.87×. However, binarization algorithms also introduce a significant performance drop, the largest among all compression approaches (*e.g.*, CIFAR10-Res20 binary 85.74 *vs.* pruning 91.54). These results show that network binarization pursues a more radical efficiency improvement among existing compression approaches and is oriented for deployment on edge devices, which is consistent with the conclusions in BiBench.

Table 11: Accuracy on ShapeNet dataset.

Task	Arch.	Category	FP32	Binarization Algorithm							
				BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA
ShapeNet	PointNet	Airplane	83.7	37.5	74.14	67.2	67.61	30.36	66.12	31.61	65.34
		Bag	79.6	44.2	49	55.34	47.11	37.44	50.28	38.58	48.62
		Cap	92.3	44.3	73.32	51.21	61.41	40.37	56.73	40.13	56
		Car	76.8	24.3	55.27	52.24	49.39	24.07	49.11	23.92	58.5
		Chair	90.9	61.6	85.62	83.96	83.6	41.89	83.83	41.5	83.27
		Earphone	70.2	38.5	30.97	34.94	35.24	26.3	36.72	23.01	34.46
		Guitar	91.1	32.9	69.17	67.9	65.99	23.45	64.18	28.38	78.69
		Knife	85.7	43	78.16	76.16	75.53	37.62	75.01	38.81	77.07
		Lamp	82	51.2	69	68.75	60	49.45	66.13	48.41	67.45
		Laptop	95.5	49.4	93.29	92.93	92.79	41.89	92.93	42.28	93.66
		Motorbike	64.4	16.3	19.04	18.88	18.69	13.18	18.59	11.26	20.38
		Mug	93.6	49.1	64.32	53.56	52.01	47.58	52.51	46.83	53.48
		Pistol	80.8	25.5	62.29	59.15	51.43	26.96	53.79	27.81	62.61
		Rocket	54.4	26.9	30.95	27.92	26.61	22.38	26.01	19.32	23.08
		Skateboard	70.7	41.2	45.7	50.15	43.78	28.63	43.74	26.71	45.81
		Table	81.4	51.3	73.68	72.69	69.72	45.74	69.56	45.21	73.45
		Overall	80.81875	39.82	60.87	58.31	56.31	33.58	56.58	33.36	58.68

Table 12: Accuracy on Language and Speech Tasks.

Task		Arch.	FP32	Binarization Algorithm							
				BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA
GLUE	MNLI-m	BERT-Tiny$_{4L}$	82.81	36.90	41.20	52.31	55.09	37.27	55.52	38.55	59.41
		BERT-Tiny$_{6L}$	84.76	37.01	51.17	63.09	66.81	37.98	66.47	37.95	68.46
		BERT-Base	84.88	35.45	41.40	60.67	62.47	35.45	60.22	35.45	63.49
	MNLI-mm	BERT-Tiny$_{4L}$	83.08	36.54	41.55	53.01	55.57	36.07	55.89	37.62	59.76
		BERT-Tiny$_{6L}$	84.42	36.47	50.92	63.87	66.82	38.11	67.64	36.91	68.98
		BERT-Base	85.45	35.22	41.18	60.96	63.17	35.22	61.19	35.22	63.72
	QQP	BERT-Tiny$_{4L}$	90.47	66.19	73.69	75.79	77.38	64.97	76.92	67.32	78.92
		BERT-Tiny$_{6L}$	85.98	63.18	78.90	80.93	82.42	63.19	82.95	63.3	83.19
		BERT-Base	91.51	63.18	71.93	77.07	80.01	63.18	81.16	63.18	83.26
	QNLI	BERT-Tiny$_{4L}$	87.46	51.71	60.59	61.15	61.92	52.79	62.67	53.99	62.29
		BERT-Tiny$_{6L}$	90.79	52.22	62.75	66.88	69.72	51.84	70.27	51.32	72.72
		BERT-Base	92.14	51.8	60.29	70.78	70.14	54.07	69.44	51.87	72.43
	SST-2	BERT-Tiny$_{4L}$	92.43	52.98	79.93	82.45	84.06	54.01	84.17	54.24	86.12
		BERT-Tiny$_{6L}$	90.25	58.14	84.74	86.23	87.73	69.38	87.95	52.40	87.72
		BERT-Base	93.23	52.29	78.78	86.01	86.35	53.32	84.4	52.40	87.93
	CoLA	BERT-Tiny$_{4L}$	49.61	6.55	7.22	12.69	16.86	0	14.71	6.25	17.80
		BERT-Tiny$_{6L}$	54.17	2.57	12.57	15.97	17.94	0	15.24	2.24	22.21
		BERT-Base	59.71	4.63	0	4.74	15.95	0	4.63	0.40	4.63
	STS-B	BERT-Tiny$_{4L}$	86.35	4.31	18.05	18.74	22.65	7.45	22.73	8.20	27.56
		BERT-Tiny$_{6L}$	89.79	1.04	14.72	22.31	24.59	5.70	23.40	8.22	37.15
		BERT-Base	90.06	6.94	12.19	18.26	20.76	4.99	8.73	6.59	10.14
	MRPC	BERT-Tiny$_{4L}$	85.50	68.30	71.74	71.99	71.74	68.30	71.74	71.25	71.49
		BERT-Tiny$_{6L}$	87.71	68.30	70.76	71.74	71.49	68.30	71.74	69.04	71.74
		BERT-Base	86.24	68.30	68.3	70.02	70.27	68.30	71.25	68.30	69.04
	RTE	BERT-Tiny$_{4L}$	65.34	56.31	53.43	56.31	55.59	54.15	57.76	61.01	59.20
		BERT-Tiny$_{6L}$	68.95	56.31	54.51	54.51	58.12	49.09	53.43	58.84	54.87
		BERT-Base	72.20	53.43	57.04	55.23	54.51	54.87	54.51	55.23	55.23
Speech Commands		FSMN	94.89	56.45	56.45	68.65	73.60	75.04	73.80	56.45	56.45
		D-FSMN	97.51	88.32	92.03	78.92	85.11	56.77	83.80	92.11	93.91

Table 13: Results for Robustness Corruption on CIFAR10-C Dataset with Different Binarization Algorithms (1/2).

Noise	FP32	Binarization Algorithm							
		BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA
Origin	94.82	89.69	91.40	91.55	91.20	90.04	91.55	92.79	90.42
gaussian_noise-1	78.23	74.22	76.00	74.97	74.95	75.07	75.15	78.25	77.36
gaussian_noise-2	56.72	56.73	62.44	55.94	58.33	57.52	55.97	61.32	60.48
gaussian_noise-3	36.93	42.69	47.58	39.56	43.47	40.79	37.99	43.32	44.26
gaussian_noise-4	31.03	38.35	41.43	33.24	36.65	34.68	31.47	35.91	37.30
gaussian_noise-5	25.54	34.05	36.22	28.66	31.78	30.13	25.49	30.19	32.09
ipulse_nosie-1	82.54	84.57	86.94	84.68	87.30	85.72	86.89	88.73	85.8
ipulse_nosie-2	70.12	77.13	80.74	77.35	81.14	79.62	80.25	82.80	80.3
ipulse_nosie-3	59.88	70.58	75.01	69.20	74.59	71.82	72.16	76.05	72.6
ipulse_nosie-4	40.59	54.42	59.39	49.48	56.66	52.61	49.79	58.44	56.45
ipulse_nosie-5	26.03	39.86	41.54	32.72	37.28	35.12	28.42	38.26	39.98
shot_noise-1	85.75	81.51	81.31	81.84	81.58	80.66	81.88	83.98	82.42
shot_noise-2	76.61	72.04	74.02	72.21	72.81	73.03	72.32	76.70	75.1
shot_noise-3	52.21	53.90	57.08	50.66	54.59	53.76	51.31	57.22	56.56
shot_noise-4	44.13	47.58	51.29	43.59	48.36	46.64	44.21	48.78	48.91
shot_noise-5	32.73	39.93	40.79	33.80	38.50	36.47	31.79	36.46	37.8
speckle_noise-1	86.30	81.29	81.94	80.93	80.77	81.14	82.17	84.17	82.62
speckle_noise-2	71.94	68.07	70.14	67.5	69.22	69.35	68.26	72.94	71.70
speckle_noise-3	64.47	62.12	64.13	60.24	63.44	62.50	61.14	66.89	64.27
speckle_noise-4	49.81	51.93	53.77	47.93	52.75	50.59	48.39	54.13	52.40
speckle_noise-5	38.70	44.25	43.60	38.65	43.16	42.09	37.78	42.13	42.57
gaussian_blur-1	94.17	89.03	90.5	89.33	90.56	89.00	91.05	92.16	89.33
gaussian_blur-2	87.04	78.3	81.98	78.81	80.42	77.75	81.20	84.80	78.93
gaussian_blur-3	75.15	67.74	68.27	67.67	68.16	64.54	67.42	73.62	66.29
gaussian_blur-4	59.5	55.17	53.63	55.74	54.08	52.44	52.72	60.32	53.37
gaussian_blur-5	36.03	37.31	33.96	37.50	37.54	36.77	34.08	39.22	34.93
defocus_blur-1	94.2	88.73	91.06	89.1	90.32	88.91	90.92	91.98	89.58
defocus_blur-2	92.75	85.97	88.99	86.59	88.31	85.58	87.91	90.47	87.01
defocus_blur-3	87.38	79.02	82.43	78.88	80.71	77.58	80.88	84.85	79.52
defocus_blur-4	76.99	69.13	71.02	68.29	68.33	65.96	68.42	74.40	68.22
defocus_blur-5	52.09	48.85	51.99	48.82	49.17	48.45	46.92	55.70	48.27
glass_blur-1	54.93	56.57	51.72	57.94	56.78	57.29	56.27	58.82	58.92
glass_blur-2	56.37	57.93	53.46	60.42	59.21	59.32	58.03	60.25	60.56
glass_blur-3	59.21	61.43	56.98	64.11	61.72	62.41	60.39	62.84	63.32
glass_blur-4	45.65	46.50	42.72	48.48	47.19	47.83	46.88	49.09	49.23
glass_blur-5	49.19	49.52	46.40	52.06	49.83	50.02	49.08	51.14	51.82
otion_blur-1	89.40	81.57	83.27	82.00	83.11	81.48	84.19	86.21	82.83
otion_blur-2	81.95	71.52	74.71	73.38	72.48	70.99	74.35	77.75	74.09
otion_blur-3	72.48	61.87	66.21	63.86	63.39	61.57	63.85	68.31	64.36
otion_blur-4	72.79	62.40	66.18	63.94	62.84	62.03	64.54	67.88	64.13
otion_blur-5	63.91	54.14	57.98	56.07	55.90	54.35	55.60	59.71	56.65
zoo_blur-1	87.36	78.25	81.31	78.56	79.45	77.20	80.22	83.69	78.55
zoo_blur-2	83.89	74.84	77.73	75.39	75.88	72.83	75.74	80.46	74.72
zoo_blur-3	77.73	69.00	70.98	68.81	69.03	66.56	68.34	74.33	68.21
zoo_blur-4	71.39	64.12	65.21	63.79	62.81	61.01	62.47	68.67	62.58
zoo_blur-5	60.60	55.15	55.83	55.4	54.38	52.66	52.24	59.51	53.94
brighness-1	94.31	89.29	90.84	89.53	90.8	89.30	90.97	92.06	89.74
brighness-2	94.03	88.25	90.42	88.71	89.66	88.50	90.64	91.64	88.77
brighness-3	93.53	87.40	89.38	87.38	89.17	87.31	89.63	90.72	87.84
brighness-4	92.74	85.45	88.27	86.12	87.78	85.44	88.16	89.58	86.39
brighness-5	90.36	80.95	85.22	81.65	83.79	80.99	84.45	86.53	82.04

Table 14: Results for Robustness Corruption on CIFAR10-C Dataset with Different Binarization Algorithms (2/2).

Noise	FP32	Binarization Algorithm							
		BNN	XNOR	DoReFa	Bi-Real	XNOR++	ReActNet	ReCU	FDA
fog-1	94.04	88.17	90.89	88.84	89.91	88.51	90.84	92.08	89.43
fog-2	93.03	84.58	88.85	85.48	87.26	84.77	88	89.87	86.76
fog-3	90.69	78.07	85.2	80.07	83.32	78.94	83.77	86.82	82.78
fog-4	86.72	69.56	78.92	72.27	75.89	71.01	77.96	81.56	75.62
fog-5	68.6	49.04	53.9	52.33	52.88	49.68	57.67	62.29	55.18
frost-1	89.97	83.66	84.7	84.07	85.76	83.64	85.51	87.75	84.85
frost-2	84.42	77.88	78.97	77.47	78.96	77.26	79.14	81.4	79.16
frost-3	74.85	67.67	69.3	67.14	68.76	66.03	69.58	72.54	70.14
frost-4	73.32	65.93	67.14	65.97	68.52	65.37	67.93	71.44	69.41
frost-5	62.13	55.02	56.67	55.62	56.77	54.26	57.69	61.11	59.88
snow-1	89.26	84.58	85.44	84.59	86.43	85.07	85.95	87.82	85.67
snow-2	78.96	72.01	73.37	73.14	73.42	72.43	73.65	78.05	73.84
snow-3	82.85	75.6	76.84	75.74	76.95	75.74	77.76	79.98	75.95
snow-4	80.29	70.84	72.56	71.93	72.39	71.25	72.97	76.12	72.16
snow-5	74.94	63.85	66.94	65.9	65.71	64.24	66.29	70.33	66.53
contrast-1	93.82	87.23	89.96	87.93	89.68	87.88	90.16	91.53	88.84
contrast-2	90.53	76.02	84	77.27	82.1	76.3	82.8	86.02	81.37
contrast-3	85.84	63.97	77.1	65.77	72.77	64.18	74.62	79.30	71.89
contrast-4	75.08	44.07	62.37	47.35	55.57	44.94	59.87	65.72	55.14
contrast-5	29.36	20	25.67	20.18	22.18	21.04	25.28	25.66	24.04
elastic_transfor-1	89.97	82.97	84.5	83.38	84.74	82.48	84.42	86.54	83.25
elastic_transfor-2	89.43	82.12	84.79	82.44	84.07	81.97	84.61	86.20	83.15
elastic_transfor-3	85.52	77.56	80.71	77.92	79.11	77	79.53	82.27	78.35
elastic_transfor-4	79.48	73.75	74.83	73.92	73.41	72.77	73.98	77.53	74.17
elastic_transfor-5	75.02	70.97	70.22	71.36	71.03	71.63	71.65	75.31	71.49
jpeg_copression-1	87.36	83.28	83.93	84.07	83.83	83.77	84.3	85.65	83.63
jpeg_copression-2	81.68	80.09	79.66	79.77	80.03	80.29	80.59	81.66	79.8
jpeg_copression-3	79.98	78.55	78.21	78.32	78.27	78.99	78.51	79.94	78.44
jpeg_copression-4	77.17	77.12	75.78	77.44	77.04	77.46	77.08	77.67	76.71
jpeg_copression-5	73.85	74.51	73.04	74.65	74.13	75.26	74.16	74.85	74.19
pixelate-1	92.57	86.97	88.19	87.39	88.73	87.47	88.17	89.42	87.26
pixelate-2	88.23	81.91	80.95	82.37	82.82	81.93	81.99	83.80	80.95
pixelate-3	84	78.4	75.25	78.63	78.03	77.15	77.28	78.89	75.30
pixelate-4	68.51	64.11	58.11	62.49	60.89	61.43	60.06	61.71	59.95
pixelate-5	50.57	50.68	44.77	48.18	45.44	46.74	43.27	47.29	45.62
saturate-1	92.41	84.98	88.38	85.38	87.26	85.39	88.23	89.34	86.82
saturate-2	90.12	80.74	85.26	81.57	82.68	80.74	84.22	86.06	83.09
saturate-3	93.83	87.89	90.45	88.12	89.53	88.03	90.15	90.97	88.73
saturate-4	91.61	82.5	86.88	82.6	84.66	82.44	86.04	87.56	83.70
saturate-5	87.48	76.03	82.76	75.53	78.3	75.85	80.62	82.64	77.00
spatter-1	91.17	87.5	89.75	87.83	89.34	87.9	89.42	91.00	88.98
spatter-2	85.2	83.85	85.98	83.52	85.64	84.72	85.88	87.59	85.00
spatter-3	80.63	77.94	80.33	77.95	80.19	79.41	80.07	82.65	79.50
spatter-4	94.68	84.57	86.71	84.77	86.51	85.14	86.72	88.22	85.32
spatter-5	74.07	78.85	80.77	78.71	80.94	79.71	80.51	83.14	79.48
Overall	74.11	69.43	71.51	69.36	70.76	69.09	70.31	73.56	70.70

Table 15: Sensitivity to Hyper Parameters in Training (1/2).

Algorithm	Epoch	Optimizer	Learning Rate	Scheduler	Acc.	Acc.$_1$	Acc.$_2$	Acc.$_3$	Acc.$_4$	mean	std
FP32	200	SGD	0.1	cosine	94.58	94.6	95.05	94.64	94.84	94.74	0.20
	200	SGD	0.1	step	92.63	92.42	92.15	92.62	92.38	92.44	0.20
	200	SGD	0.01	cosine	92.23	91.76	91.76	91.99	92.17	91.98	0.22
	200	SGD	0.01	step	83.94	83.50	82.80	84.13	83.89	83.65	0.53
	200	Adam	0.001	cosine	93.51	92.94	93.12	93.35	92.86	93.16	0.27
	200	Adam	0.001	step	93.37	93.15	93.32	93.41	93.35	93.32	0.10
	200	Adam	0.0001	cosine	89.97	89.92	89.96	89.9	89.92	89.93	0.03
	200	Adam	0.0001	step	90.57	89.91	90.43	90.25	90.31	90.29	0.25
BNN	200	SGD	0.1	cosine	87.62	87.53	87.99	88.86	87.84	87.97	0.53
	200	SGD	0.1	step	70.87	73.86	71.83	73.1	72.87	72.51	1.17
	200	SGD	0.01	cosine	73.52	72.62	72.82	71.14	72.59	72.54	0.87
	200	SGD	0.01	step	52.85	51.77	52.00	52.34	53.14	52.42	0.57
	200	Adam	0.001	cosine	88.76	88.99	88.67	88.84	88.81	88.81	0.12
	200	Adam	0.001	step	88.85	89.34	88.77	89.02	89.00	89.00	0.22
	200	Adam	0.0001	cosine	83.46	83.09	83.20	83.70	83.20	83.33	0.25
	200	Adam	0.0001	step	84.08	84.11	84.20	84.31	83.56	84.05	0.29
XNOR	200	SGD	0.1	cosine	91.83	91.99	91.87	92.01	91.56	91.85	0.18
	200	SGD	0.1	step	90.02	90.01	90.12	89.82	89.7	89.93	0.17
	200	SGD	0.01	cosine	90.09	89.68	90.01	89.97	90.00	89.95	0.16
	200	SGD	0.01	step	86.86	86.66	87.21	86.98	86.61	86.86	0.24
	200	Adam	0.001	cosine	89.39	89.81	89.73	89.91	89.75	89.72	0.20
	200	Adam	0.001	step	89.92	89.79	89.73	90.01	89.61	89.81	0.16
	200	Adam	0.0001	cosine	86.18	86.29	87.03	86.36	86.62	86.50	0.34
	200	Adam	0.0001	step	86.32	87.04	86.68	86.99	87.18	86.84	0.34
DoReFa	200	SGD	0.1	cosine	85.64	85.67	85.89	86.00	85.79	85.80	0.15
	200	SGD	0.1	step	86.95	86.98	86.69	86.62	86.65	86.78	0.17
	200	SGD	0.01	cosine	86.56	86.59	86.52	86.69	86.88	86.65	0.14
	200	SGD	0.01	step	78.76	79.97	80.73	79.94	80.47	79.97	0.76
	200	Adam	0.001	cosine	88.85	89.06	88.92	88.87	88.75	88.89	0.11
	200	Adam	0.001	step	89.08	89.16	88.93	89.23	88.84	89.05	0.16
	200	Adam	0.0001	cosine	83.56	83.17	83.65	83.60	83.66	83.53	0.20
	200	Adam	0.0001	step	83.70	83.74	84.27	84.19	84.01	83.98	0.26
Bi-Real	200	SGD	0.1	cosine	87.55	87.81	88.06	87.30	87.88	87.72	0.30
	200	SGD	0.1	step	87.95	88.35	88.13	87.73	88.25	88.08	0.25
	200	SGD	0.01	cosine	87.76	87.93	87.73	87.72	87.64	87.76	0.11
	200	SGD	0.01	step	83.75	82.91	82.82	82.91	83.39	83.16	0.40
	200	Adam	0.001	cosine	88.78	89.15	89.06	89.00	89.2	89.04	0.16
	200	Adam	0.001	step	88.89	88.98	88.78	89.11	89.05	88.96	0.13
	200	Adam	0.0001	cosine	83.96	84.17	84.37	83.54	84.07	84.02	0.31
	200	Adam	0.0001	step	84.63	84.48	84.32	84.75	84.29	84.49	0.20

Table 16: Sensitivity to Hyper Parameters in Training (2/2).

Algorithm	Epoch	Optimizer	Learning Rate	Scheduler	Acc.	Acc.$_1$	Acc.$_2$	Acc.$_3$	Acc.$_4$	mean	std
XNOR++	200	SGD	0.1	cosine	87.82	88.42	88.12	88.55	88.19	88.22	0.28
	200	SGD	0.1	step	73.55	73.11	75.06	74.05	73.78	73.91	0.73
	200	SGD	0.01	cosine	74.03	75.06	73.64	74.53	74.71	74.39	0.56
	200	SGD	0.01	step	53.55	54.16	54.01	52.91	54.36	53.80	0.58
	200	Adam	0.001	cosine	88.77	88.65	89.10	88.61	88.81	88.79	0.19
	200	Adam	0.001	step	89.18	89.05	89.27	88.93	89.00	89.09	0.14
	200	Adam	0.0001	cosine	83.86	83.49	83.56	83.16	83.62	83.54	0.25
	200	Adam	0.0001	step	83.46	83.77	84.40	84.06	83.82	83.90	0.35
ReActNet	200	SGD	0.1	cosine	88.60	88.53	88.38	88.48	88.89	88.58	0.19
	200	SGD	0.1	step	88.42	88.01	88.10	88.02	88.43	88.20	0.21
	200	SGD	0.01	cosine	87.75	87.86	88.00	87.80	88.02	87.89	0.12
	200	SGD	0.01	step	83.29	82.89	83.65	83.76	83.27	83.37	0.35
	200	Adam	0.001	cosine	89.47	89.29	89.01	89.05	89.14	89.19	0.19
	200	Adam	0.001	step	89.27	89.74	89.48	89.40	89.39	89.46	0.18
	200	Adam	0.0001	cosine	84.65	84.93	84.48	84.65	84.67	84.68	0.16
	200	Adam	0.0001	step	84.69	84.55	84.93	84.94	85.38	84.90	0.32
ReCU	200	SGD	0.1	cosine	91.72	91.94	91.68	91.69	91.81	91.77	0.11
	200	SGD	0.1	step	87.73	88.14	87.81	88.02	87.91	87.92	0.16
	200	SGD	0.01	cosine	87.32	87.28	87.53	87.48	87.32	87.39	0.11
	200	SGD	0.01	step	71.86	71.72	71.78	72.26	71.59	71.84	0.25
	200	Adam	0.001	cosine	88.24	89.98	88.26	88.48	88.13	88.62	0.77
	200	Adam	0.001	step	88.36	88.48	88.55	88.42	88.63	88.49	0.11
	200	Adam	0.0001	cosine	80.07	81.10	80.62	81.09	79.95	80.57	0.55
	200	Adam	0.0001	step	81.26	81.42	81.08	81.58	81.69	81.41	0.24
FDA	200	SGD	0.1	cosine	89.69	89.59	89.56	89.53	89.65	89.60	0.07
	200	SGD	0.1	step	80.38	80.34	80.83	80.52	80.52	80.52	0.19
	200	SGD	0.01	cosine	80.72	80.93	80.89	80.70	80.79	80.81	0.10
	200	SGD	0.01	step	63.41	62.85	63.04	63.04	63.14	63.10	0.20
	200	Adam	0.001	cosine	89.70	89.57	89.57	89.80	89.76	89.68	0.11
	200	Adam	0.001	step	89.84	89.85	90.10	89.79	90.01	89.92	0.13
	200	Adam	0.0001	cosine	89.59	89.10	89.34	89.31	89.51	89.37	0.19
	200	Adam	0.0001	step	89.52	89.59	89.52	89.64	89.58	89.57	0.05

Table 17: Inference Efficiency on Hardware (1/4).

Hardware	Threads	Arch.	Larq			daBNN	
			FP32	BNN	ReAct	FP32	BNN
Kirin 970	1	ResNet18	716.427	123.263	126.457	427.585	72.585
		ResNet34	1449.67	159.615	171.227	836.321	124.091
		VGG-Small	242.443	14.833	16.401	–	–
	2	ResNet18	372.642	72.697	78.605	–	–
		ResNet34	732.355	96.711	108.41	–	–
		VGG-Small	121.91	10.304	11.935	–	–
	4	ResNet18	191.517	42.986	47.182	–	–
		ResNet34	367.891	61.413	73.101	–	–
		VGG-Small	57.721	8.72	8.387	–	–
	8	ResNet18	96.937	37.457	56.017	–	–
		ResNet34	212.982	53.809	67.667	–	–
		VGG-Small	33.647	18.649	19.818	–	–
Kirin 980	1	ResNet18	307.624	49.009	50.018	158.363	31.803
		ResNet34	507.734	71.909	74.920	308.537	53.031
		VGG-Small	83.163	7.772	8.215	–	–
	2	ResNet18	187.494	52.057	54.285	–	–
		ResNet34	367.853	57.336	60.483	–	–
		VGG-Small	49.264	6.116	5.604	–	–
	4	ResNet18	104.076	29.556	35.539	–	–
		ResNet34	202.173	31.324	35.911	–	–
		VGG-Small	22.690	3.147	3.291	–	–
	8	ResNet18	60.307	45.683	56.416	–	–
		ResNet34	120.738	60.758	86.887	–	–
		VGG-Small	18.147	21.688	23.350	–	–
Kirin 985	1	ResNet18	173.238	27.429	30.626	164.556	34.528
		ResNet34	438.971	58.165	60.885	323.439	57.808
		VGG-Small	70.797	6.147	6.796	–	–
	2	ResNet18	103.621	25.672	35.477	–	–
		ResNet34	327.416	53.949	62.865	–	–
		VGG-Small	55.328	5.955	6.243	–	–
	4	ResNet18	92.387	26.728	34.778	–	–
		ResNet34	184.050	39.881	52.153	–	–
		VGG-Small	28.076	8.919	14.795	–	–
	8	ResNet18	130.972	82.772	89.766	–	–
		ResNet34	227.504	119.586	143.958	–	–
		VGG-Small	44.339	34.034	43.816	–	–
Kirin 990	1	ResNet18	114.238	21.235	22.066	144.205	29.239
		ResNet34	227.043	31.545	32.821	275.502	49.476
		VGG-Small	38.118	3.338	3.482	–	–
	2	ResNet18	59.329	13.911	14.179	–	–
		ResNet34	116.822	23.452	22.770	–	–
		VGG-Small	20.055	2.080	2.194	–	–
	4	ResNet18	38.403	10.280	12.208	–	–
		ResNet34	81.273	15.570	17.727	–	–
		VGG-Small	13.508	1.542	1.760	–	–
	8	ResNet18	37.703	25.360	31.365	–	–
		ResNet34	78.753	34.884	39.363	–	–
		VGG-Small	12.707	14.414	21.749	–	–

Table 18: Inference Efficiency on Hardware (2/4).

Hardware	Threads	Arch.	Larq			daBNN	
			FP32	BNN	ReAct	FP32	BNN
Kirin 9000E	1	ResNet18	118.059	19.865	20.547	129.270	24.781
		ResNet34	236.047	31.822	32.575	250.680	42.134
		VGG-Small	39.114	3.595	3.832	–	–
	2	ResNet18	68.351	16.821	16.115	–	–
		ResNet34	133.671	25.061	24.660	–	–
		VGG-Small	23.018	2.684	2.598	–	–
	4	ResNet18	45.592	17.452	18.847	–	–
		ResNet34	91.648	23.395	28.022	–	–
		VGG-Small	14.360	2.762	2.782	–	–
	8	ResNet18	43.363	61.263	42.328	–	–
		ResNet34	89.405	70.232	93.558	–	–
		VGG-Small	19.070	17.351	23.825	–	–
Dimensity 820	1	ResNet18	158.835	32.636	34.912	323.035	63.471
		ResNet34	328.020	57.133	60.807	629.493	102.443
		VGG-Small	82.417	5.958	6.420	–	–
	2	ResNet18	122.167	29.306	34.384	–	–
		ResNet34	250.088	43.306	50.143	–	–
		VGG-Small	51.320	4.670	5.053	–	–
	4	ResNet18	94.636	21.850	30.027	–	–
		ResNet34	177.757	33.809	40.816	–	–
		VGG-Small	45.056	4.223	4.546	–	–
	8	ResNet18	90.210	45.357	61.981	–	–
		ResNet34	166.989	68.444	74.286	–	–
		VGG-Small	32.971	21.344	23.706	–	–
Dimensity 9000	1	ResNet18	106.388	21.023	24.770	148.690	29.030
		ResNet34	210.665	32.841	34.590	284.438	49.854
		VGG-Small	42.057	4.410	5.530	–	–
	2	ResNet18	81.606	22.661	27.050	–	–
		ResNet34	143.349	27.666	37.910	–	–
		VGG-Small	26.512	2.273	2.410	–	–
	4	ResNet18	51.421	13.079	15.200	–	–
		ResNet34	100.249	23.314	25.920	–	–
		VGG-Small	17.735	3.015	3.770	–	–
	8	ResNet18	43.355	24.939	30.740	–	–
		ResNet34	84.182	30.212	39.990	–	–
		VGG-Small	14.857	14.258	17.540	–	–
Snapdragon 855+	1	ResNet18	90.430	19.769	20.530	163.293	31.174
		ResNet34	186.694	29.126	30.512	298.882	49.948
		VGG-Small	29.735	3.153	3.259	–	–
	2	ResNet18	58.510	25.780	26.331	–	–
		ResNet34	124.580	31.023	32.646	–	–
		VGG-Small	19.408	2.258	2.471	–	–
	4	ResNet18	39.269	19.865	23.297	–	–
		ResNet34	82.180	30.248	31.387	–	–
		VGG-Small	13.566	2.032	2.359	–	–
	8	ResNet18	36.630	49.060	85.861	–	–
		ResNet34	73.513	41.131	88.101	–	–
		VGG-Small	12.860	17.828	23.489	–	–

Table 19: Inference Efficiency on Hardware (3/4).

Hardware	Threads	Arch.	Larq FP32	Larq BNN	Larq ReAct	daBNN FP32	daBNN BNN
Snapdragon 870	1	ResNet18	88.145	16.527	17.020	126.762	25.240
		ResNet34	185.468	25.488	26.195	237.361	41.440
		VGG-Small	30.318	2.851	2.964	–	–
	2	ResNet18	63.829	18.351	19.575	–	–
		ResNet34	159.174	25.352	26.340	–	–
		VGG-Small	27.669	2.094	2.308	–	–
	4	ResNet18	42.796	17.578	21.083	–	–
		ResNet34	89.960	25.816	27.201	–	–
		VGG-Small	14.829	2.614	2.215	–	–
	8	ResNet18	46.798	19.192	28.579	–	–
		ResNet34	97.834	25.060	32.863	–	–
		VGG-Small	16.799	9.717	17.293	–	–
Snapdragon 888	1	ResNet18	77.205	15.547	16.111	123.618	25.240
		ResNet34	152.887	22.906	23.893	234.972	41.648
		VGG-Small	25.133	2.410	2.543	–	–
	2	ResNet18	46.297	19.309	19.321	–	–
		ResNet34	93.615	20.473	22.489	–	–
		VGG-Small	16.001	1.920	2.213	–	–
	4	ResNet18	33.524	13.699	14.332	–	–
		ResNet34	67.495	19.020	21.157	–	–
		VGG-Small	11.743	2.882	2.768	–	–
	8	ResNet18	33.761	26.108	58.989	–	–
		ResNet34	67.876	37.018	61.315	–	–
		VGG-Small	11.752	27.615	16.774	–	–
Raspberrypi 3B+	1	ResNet18	740.509	158.732	175.256	1460.723	241.713
		ResNet34	1536.915	240.606	254.810	2774.888	435.170
		VGG-Small	257.079	24.479	25.790	–	–
	2	ResNet18	667.012	143.716	106.894	–	–
		ResNet34	933.149	144.287	158.868	–	–
		VGG-Small	145.427	14.503	15.628	–	–
	4	ResNet18	562.567	108.585	116.640	–	–
		ResNet34	976.223	159.258	183.698	–	–
		VGG-Small	191.470	10.839	10.196	–	–
	8	ResNet18	877.026	279.660	356.239	–	–
		ResNet34	1638.035	389.924	485.260	–	–
		VGG-Small	399.338	110.448	142.978	–	–
Raspberrypi 4B	1	ResNet18	448.744	80.822	82.380	688.838	120.348
		ResNet34	897.735	112.837	119.536	1362.893	209.276
		VGG-Small	150.814	11.177	12.024	–	–
	2	ResNet18	261.861	49.079	55.279	–	–
		ResNet34	525.735	67.480	79.468	–	–
		VGG-Small	89.284	6.647	7.882	–	–
	4	ResNet18	270.191	36.331	45.903	–	–
		ResNet34	572.423	53.866	70.841	–	–
		VGG-Small	90.650	5.056	6.167	–	–
	8	ResNet18	466.585	168.844	226.771	–	–
		ResNet34	879.375	264.638	319.789	–	–
		VGG-Small	216.439	114.064	162.118	–	–

Table 20: Inference Efficiency on Hardware (4/4).

Hardware	Threads	Arch.	Larq			daBNN	
			FP32	BNN	ReAct	FP32	BNN
Apple M1	1	ResNet18	44.334	8.219	8.355	–	–
		ResNet34	88.334	12.505	12.771	–	–
		VGG-Small	14.093	1.446	1.465	–	–
	2	ResNet18	24.775	5.037	5.194	–	–
		ResNet34	47.179	7.425	7.690	–	–
		VGG-Small	7.398	0.829	0.854	–	–
	4	ResNet18	18.612	3.448	3.671	–	–
		ResNet34	27.515	4.965	5.254	–	–
		VGG-Small	4.294	0.526	0.551	–	–
	8	ResNet18	16.653	5.035	6.003	–	–
		ResNet34	27.680	6.445	6.953	–	–
		VGG-Small	3.996	0.735	0.712	–	–
	16	ResNet18	90.323	70.697	73.729	–	–
		ResNet34	162.057	130.907	125.362	–	–
		VGG-Small	25.366	23.050	23.194	–	–
Apple M1 Max	1	ResNet18	46.053	8.653	8.486	–	–
		ResNet34	91.861	12.593	13.039	–	–
		VGG-Small	14.285	1.454	1.488	–	–
	2	ResNet18	25.039	5.450	5.361	–	–
		ResNet34	51.860	7.579	8.925	–	–
		VGG-Small	7.657	0.855	0.896	–	–
	4	ResNet18	14.708	3.625	3.888	–	–
		ResNet34	27.933	5.266	6.021	–	–
		VGG-Small	4.292	0.576	0.620	–	–
	8	ResNet18	10.660	3.718	4.510	–	–
		ResNet34	18.988	4.745	5.457	–	–
		VGG-Small	3.432	0.560	0.629	–	–
	16	ResNet18	60.500	47.727	53.900	–	–
		ResNet34	120.449	91.464	96.356	–	–
		VGG-Small	21.354	13.868	15.311	–	–

Table 21: Accuracy and efficiency comparison among multi-bit quantization (2&8-bits), pruning, and binarization.

	Accuracy		Efficiency	
	CIFAR10-Res18	CIFAR10-Res20	FLOPs (Res20, G)	Param (Res20, K)
FP32	94.82	91.99	13.61	11690
Binary (overall)	91.08 (-3.74)	85.74 (-6.25)	1.105 (12.32×)	884 (13.22×)
DoReFa-INT2 (Zhou et al., 2016)	92.71	87.56	1.686	1681
PACT-INT2 (Choi et al., 2018)	92.98	88.12	1.686	1681
LSQ-INT2 (Esser et al., 2019)	93.11	89.26	1.686	1681
INT2 (overall)	92.93 (-1.89)	88.31 (-3.68)	1.686 (8.07×)	1681 (6.95×)
DoReFa-INT8 (Zhou et al., 2016)	94.79	91.83	7.278	4067
PACT-INT8 (Choi et al., 2018)	94.8	91.87	7.278	4067
LSQ-INT8 (Esser et al., 2019)	94.78	91.95	7.278	4067
INT8 (overall)	94.79 (-0.03)	91.88 (-0.11)	7.278 (1.87×)	4067 (2.87×)
(Li et al., 2016)	94.47	91.32	9.527	9936
ResRep (Ding et al., 2021)	94.53	91.76	6.805	7247
Pruning (overall)	94.50 (-0.32)	91.54 (-0.45)	8.166 (1.67×)	8591 (1.36×)