Towards Understanding Model Quantization for Reliable Deep Neural Network Deployment

Qiang Hu[1], Yuejun Guo[2], Maxime Cordy[1], Xiaofei Xie[3], Wei Ma[4], Mike Papadakis[1], and Yves Le Traon[1]
[1]University of Luxembourg, Luxembourg
[2]Luxembourg Institute of Science and Technology, Luxembourg
[3]Singapore Management University, Singapore
[4]Nanyang Technological University, Singapore

Abstract—**Deep Neural Networks (DNNs) have gained considerable attention in the past decades due to their astounding performance in different applications, such as natural language modeling, self-driving assistance, and source code understanding. With rapid exploration, more and more complex DNN architectures have been proposed along with huge pre-trained model parameters. A common way to use such DNN models in user-friendly devices (e.g., mobile phones) is to perform model compression before deployment. However, recent research has demonstrated that model compression, e.g., model quantization, yields accuracy degradation as well as output disagreements when tested on unseen data. Since the unseen data always include distribution shifts and often appear in the wild, the quality and reliability of models after quantization are not ensured. In this paper, we conduct a comprehensive study to characterize and help users understand the behaviors of quantization models. Our study considers four datasets spanning from image to text, eight DNN architectures including both feed-forward neural networks and recurrent neural networks, and 42 shifted sets with both synthetic and natural distribution shifts. The results reveal that 1) data with distribution shifts lead to more disagreements than without. 2) Quantization-aware training can produce more stable models than standard, adversarial, and Mixup training. 3) Disagreements often have closer top-1 and top-2 output probabilities, and *Margin* is a better indicator than other uncertainty metrics to distinguish disagreements. 4) Retraining the model with disagreements has limited efficiency in removing disagreements. We release our code and models as a new benchmark for further study of model quantization.**

I. Introduction

Thanks to the massively available data released and powerful hardware devices supported, Deep Learning (DL) gains considerable attention and achieves ever better performance than humans on different tasks [1]. Many security-critical DL systems have been deployed in recent years, e.g., self-driving car [2] and face recognition application [3]. Besides, inspired by the usage of DNNs for natural language processing, researchers also employ DNNs for source code-related tasks, e.g., code summarization [4] and problem classification [5]. As the backbone of DL systems, Deep Neural Networks (DNNs) follow the data-driven paradigm to learn knowledge from the labeled data automatically and make predictions for incoming unlabelled ones. Correspondingly, the study of the development of DNNs into DL systems, which is a key step in MLOps [6], is interesting to both security and software engineering communities.

However, a factor that limits the development of DNNs is that DNNs are usually large-size and require strong computing resources. For example, the famous language prediction model GPT-3 [7] has 175 billion parameters, which is hard to be deployed in our daily used devices. For code tasks, the recently released model GraphCodeBERT [8] occupies 124M of storage memory, which is also difficult to be plugged into the generally used IDEs. Furthermore, with the rapid research progress, more and more complex DNNs are being developed, which makes DNN deployment even more challenging.

To solve the above deployment issue, instead of directly migrating DNNs to devices, one typical process is to reduce the size of DNN models by model compression techniques for lighter and easier deployment. There are different ways to perform model compression, e.g., model pruning which removes useless parameters from the model, and model quantization which degrades float-level parameters to lower-level parameters (integer-level). In general, the compression process is important and must preserve the performance of original models as much as possible. The reason is that after compression, it is hard to further change the model when unexpected problems occur, e.g., retraining a model deployed on a mobile device is impractical because this model is packaged.

Unfortunately, recent research has revealed two problems with model compression. First, [9] shows that a compressed model could have a big accuracy difference (more than 5%) compared to its original model on the newly synthesized data (by using fuzzing techniques). Second [10], [11] demonstrate that it is common to find inputs that trigger different predictions by a compressed model and its original model. As these studies reveal, it remains unclear to what extent model compression preserves prediction performance and under which conditions. However, although some studies have been conducted, the existing literature 1) only focuses on studying the compressed model on the artificially generated unseen data, the real-world unseen data are missed; 2) lacks a detailed analysis of the characteristics of data that cause the performance difference and disagreements; 3) ignores the exploration of how to fix the disagreements. These lacks, in turn, impede the reliable application of compression techniques.

In this paper, we fill this gap and empirically characterize the behavior of compressed models under various experimental settings in order to better understand the limitations of compression techniques. We specifically consider quantization as this approach is mostly applied in practice [12], and use the term *compressed model* to represent the model after quantization in this paper. We focus our study on the DL models compressed by TensorFLowLite [13] and CoreML [14] which are widely adopted in the industry. For example, Google uses TensorFLowLite for model deployment on Android devices and Apple applies CoreML for IOS devices. In total, our experimental settings include four datasets ranging from image to text, eight different DNNs including both Feed-forward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs), 42 different test sets with both synthetic and natural distribution shifts. Accompanied by our study, we provide the first benchmark DNN models for further quantization study. With this material, we explore four research questions that existing studies have overlooked:

RQ1: How do compressed models react to distribution shifts? Real applications of DL systems often witness data distribution shifts – changes in data distribution that typically cause drops in model performance [15]. Given the practical predominance of this phenomenon, research [16]–[18] has emphasized the need to consider distribution shifts when evaluating DL models. We, therefore, study the impact of model quantization in the case of distribution shifts. We evaluate the compressed models against two types of distribution shift datasets: synthetic (based on image transformations) and natural (reported in the literature). We compare the original and the compressed models in terms of accuracy difference and predicted label differences, i.e. disagreements.

RQ2: How does the training strategy influence the behavior of compressed models? We explore the influence of different training strategies: standard training which is the basic way to prepare pretrained model, quantization-aware training [19] which is specifically designed for model quantization, adversarial training [20] and mixup training [21], which are the commonly used data augmentation training strategies. We apply each strategy to train original models and then quantize these models. We compare the pairs of models in terms of accuracy differences and disagreements.

RQ3: What are the characteristics of the data on which original and compressed models disagree? We aim to find discriminating factors that can help identify the disagreement inputs. In particular, we investigate whether the most uncertain data are the most likely to produce disagreements. Based on different uncertainty metrics, we train simple classifiers based on logistic regression and evaluate their capabilities to predict disagreements.

RQ4: Can model retraining reduce disagreements? We investigate whether retraining – a common approach to improve DL models – can efficiently fix disagreements. Specifically, we explore whether retraining the original model (for additional epochs) with disagreement inputs can help preserve the knowledge of these inputs through the quantization process, and make the compressed model classify these inputs correctly.

In summary, the main novel contributions of this paper are:

- We show that synthetic distribution shift has a significant impact on compressed models; it increases the accuracy change by up to 3.03% and the percentage of disagreements by 5.28%.
- We empirically confirm that quantization-aware training is the best method to alleviate performance loss and disagreements after quantization.
- We demonstrate that data uncertainty – as captured by the *Margin* metric – is a suitable factor to discriminate disagreement data. A simple classifier based on *Margin* reaches an AUC-ROC of 0.63 to 0.97.
- We illustrate that retraining on disagreement inputs does not decrease the total level of disagreements between original and compressed models because it has the side effect of introducing new disagreements.
- We build the first model quantization benchmark models [22] to support future research on studying and improving the reliability of model deployment.

II. Background

A. Deep Learning

Deep learning [23] is a machine learning technique that uses intermediate layers to progressively obtain knowledge from raw data, and deep neural networks form the backbone of deep learning. A typical deep neural network consists of an input layer, several hidden layers, and an output layer. Each layer includes neurons that mimic the neurons in human brains and undertake specific computations, such as sigmoid and rectifier. The connections between successive layers establish the data flow. In brief, training a deep neural network is to tune the parameters (importance of neurons) of the connections, and testing is to ensure accuracy and reliability during deployment in real-world applications.

B. Model Quantization

Model quantization is one of the most used model compression techniques that aims at transforming the higher-bit level weights to lower-bit level weights, e.g., from float32 weights to 8-bit integer weights, to reduce the size of the model for easy model deployment. Multiple quantization approaches [19], [24]–[26] have been proposed given its importance in DL-based engineering. An important part of quantization methods is the mapping between the two parts of weights. This mapping can be constructed by using a simple linear function to find the scale for two levels of weights, or by different clustering metrics (e.g., k-means cluster used in CoreML) to find the lookup table quantization of weights.

Figure 1 gives an example of basic linear quantization. We assume the left side is the float32-level weights, and the range of these weights is [-1, 1]. We plan to convert the float weights to 8-bit integer weights ranging in [-128, 127]. Thus, the $scale$ here is 128 and the compressed weights are calculated by $Round(weights_{float}/scale)$. Generally, to further reduce the memory usage of compressed models, the quantization only keeps positive weights.

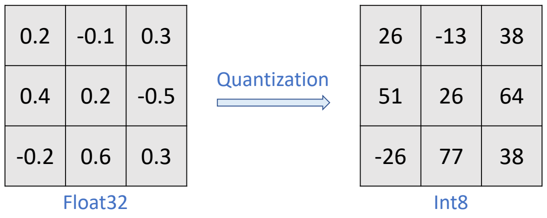


Fig. 1. An example of weights quantization. Each weight in Float32 format is converted into Int8.

C. Distribution Shift

Distribution shift refers to the change of data distribution in the test dataset compared to the training dataset. Generally, benchmark datasets [27], [28] are designed to include training and test data following the same distribution. However, in real-world deployments, the test data can be from the same or a different distribution, which raises the security concern [16]. **A motivating example can be found on our project site [22].**

Generally, there are two types of distribution shifts, synthetic and natural [15]. Synthetic distribution shift considers possible perturbations in the real world. In addition, considering the severity of corruption, data can have various levels of noise, which covers many different situations. As a result, synthetic distribution shift is always taken as a starting point to evaluate the performance of a DNN under different settings. A wide range of visual corruptions has been developed in the image domain [29], [30]. For example, adding motion blur into an image can mimic the scenario of a moving object, and inserting the fog effect can simulate the condition of foggy weather. Different from synthetic shift, natural distribution shift comes from natural variations in datasets. For instance, in the widely used text dataset, IMDb [28], data (movie reviews) are collected from IMDb. When testing, the reviews can be from another movie review website.

III. Overview

A. Study Design

Figure 2 gives an overview of our study. Overall, instead of only analyzing the behavior of compressed models produced from pre-trained models, we also follow the general MLOps to explore how the training process affects the compressed model in the DNN development phase, and if the model repair process can enhance the compressed model in the DNN maintenance phase. Specifically, following the common DL systems development process, we prepare the original model *DNN* by standard model training (Section III-D) using the collected datasets. Then, we use quantization techniques (e.g., TensorflowLite, CoreML) to compress the model and prepare the optimized model *DNN'* for further deployment. Afterward, to study whether the quantization is reliable or not, we prepare two types of test data, the ID test set, and the OOD test set. Remark that the ID test set is the original test data from each dataset, which is in distribution compared to the training data. The OOD test set is the data with distribution shifts. We compare the performance of the original *DNN* and compressed *DNN'* on these two types of test sets and check the *differences* to answer **RQ1**. In our study, we consider two types of distribution shifts, synthetic and natural.

In the development phase, in addition to standard training, some other training strategies are often used to prepare pre-trained models. Thus, it is essential to explore the potential factor that could influence the behaviors of compressed models – training strategy. We utilize three additional training strategies to train models and then analyze the behaviors of their compressed versions to answer **RQ2**. Specifically, we include quantization-aware training [19], which is specifically designed for solving the problem of accuracy decline after quantization, adversarial training [20] and Mixup training [21] which aim to improve the generalization of a DNN model.

After analyzing the behaviors of compressed models, we obtain multiple models that are waiting for repair with their disagreements. Before trying to remove the disagreements and repair the compressed models, the first step should be to investigate the properties of the data that cause disagreements between *DNN* and *DNN'*. We utilize the uncertainty metric as an indicator to check if it can represent the properties of disagreements and answer **RQ3**. Specifically, for each test dataset (ID/OOD) and model, we collect all the disagreements that have at least once been predicted differently by the original model and the compressed model. Then, we randomly select the same number as the disagreements of normal inputs where the predictions before and after quantization are consistent. Afterward, we obtain the output probabilities of these two (disagreements and normal inputs) sets and calculate their uncertainty scores by different uncertainty metrics as the input data of the logistic regression classifier. We assign the label of disagreement and normal input as 1 and 0, respectively. We then combine and shuffle the two sets and split them into training data and test data following the ratio 9:1. Finally, we train the classifier using the training data and calculate the AUC-ROC score of the classifiers using the prediction of test data with a threshold of 95%. The AUC-ROC score is used to determine the best uncertainty metric that is discriminative between disagreements and normal inputs significantly.

Finally, we make the first step to repairing the compressed model. We verify if model retraining is helping to alleviate disagreements to answer **RQ4**. Model retraining is the most straightforward and commonly used method during deployment to specifically let a pre-trained model work on unlearnt features [31]. However, its effectiveness on model quantization is uncovered. After retraining, we follow the same procedure as RQ1 to produce the compressed model and check if the disagreements decreased. Remarkably, we consider both the existing and newly generated disagreements.

B. Datasets and Models

Table I presents the details of datasets and models. In this study, we consider four widely studied datasets over image and text domains. For each dataset, we build two different models. More specifically, MNIST [27] is a gray-scale image dataset containing digit numbers from 0 to 9. We train LeNet-

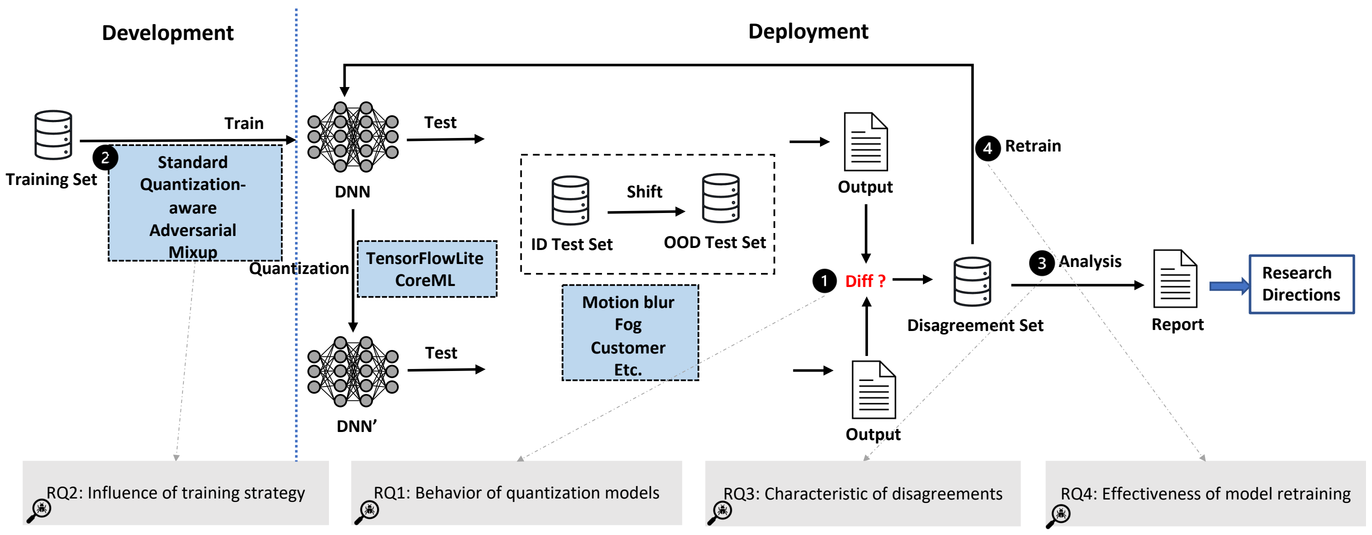


Fig. 2. Overview of the experimental design.

TABLE I
DETAILS OF DATASETS AND DNNs

Dataset	Classes	Training	Test	Model	Parameters	Accuracy (%)	Distribution Shift
MNIST	10	60000	10000	LeNet-1	7206	98.62	MNIST-C
				LeNet-5	107786	98.87	(Synthetic)
CIFAR-10	10	50000	10000	ResNet20	274442	87.44	CIFAR-10-C
				NiN	972658	88.27	(Synthetic)
iWildCam	182	129809	8154	ResNet-50	23960630	75.78	Camera Traps
				Densenet-101	7224054	76.01	(Natural)
IMDb	2	5000	5000	LSTM	2694206	83.78	CR, Yelp
				GRU	2661694	83.14	(Natural)

1 and LeNet-5 from the LeNet [27] family. CIFAR-10 [32] contains color images of airplanes and birds. For this dataset, we build two models, Network in Network (NiN) [33] and ResNet-20 [34]. iWildCam is a dataset from the distribution shift benchmark Wilds [15]. It consists of color images of different animals, e.g., cows, wild horses, and giraffes. We follow the recommendation of the benchmark to build ResNet-50 [34] for iWildCam and add one more model, DenseNet-121 [35], in our study. IMDb [28] is a text dataset collected from the popular movie review website IMDb. This dataset is mainly used for sentiment analysis, i.e., the reviewer holds a positive or negative opinion of a movie. We build two well-known RNN models, LSTM [36] and GRU [37], for IMDb.

Test data with distribution shift. For synthetic distribution shift, we test on MNIST and CIFAR-10 with benchmark datasets MNIST-C [29] and CIFAR-10-C [30], respectively. Both benchmarks include several groups of noisy images synthesized by different image transformation methods, e.g., image rotation and image scale. MNIST-C contains 16 types of transformations and CIFAR-10-C has 19 types. For natural distribution shift, we test on iWildCam and IMDb using the Wilds benchmark. The distribution shift comes from the change of camera traps in iWildCam and the difference in websites and customers in IMDb.

C. Quantization Techniques

TensorflowLite [13] is a component of the deep learning framework – TensorFlow, which is developed and maintained by Google. It provides interfaces to convert TensorFlow models into Lite models to promote the deployment in different low-computing devices, such as Android mobile phones. Currently, TensorFLowLite supports both 8-bit integer and 16-bit float quantizations for most DNNs except 8-bit integer quantization for RNNs [38]. In our experiments, we only apply 16-bit float quantization for IMDb-related models.

CoreML [14] is an Apple framework that converts models from third-party frameworks (e.g., TensorFlow and PyTorch) to Mlmodel. Mlmodel is a specific deep learning model format for IOS platforms. CoreML also provides post-training quantization interfaces to compress models. Different from TensorflowLite, CoreML supports all bits level quantization for all types of DNNs.

D. Training Strategies

In addition to standard training, we consider three representative training strategies from different perspectives, quantization-aware [19], adversarial [20], and Mixup [21].

Standard training is the baseline to evaluate the other training strategies. In this setting, we train the model without any modification in the model (e.g., quantization-aware) or data (e.g., Mixup).

Quantization-aware training is designed by the TensorFlow group, which is used for preserving the accuracy of the model after post-training quantization in the training process. It simulates the quantization effects in the forward pass of training. Namely, during training, the parameters of the model will be updated by both the normal operations and the injected quantization operations. In this way, the trained model can learn the knowledge for quantization.

Adversarial training is one of the most effective defenses for promoting model robustness by adversarially data augmentation. Compared to standard training, adversarial examples crafted from raw inputs are fed to train the model during each epoch. As a result, the training dataset is augmented successively.

Mixup training is a data augmentation technique that generates new samples by weighted combinations of random training data and their labels. It has been empirically proven to be effective in improving the generalization of DNNs and has several variants, such as AugMix [39]. In this paper, we consider the original Mixup.

E. Evaluation Measures

We consider both the **accuracy** and **disagreement** to evaluate the performance of DNNs and use **AUC-ROC** to evaluate the performance of logistic regression classifiers.

Accuracy is the basic criterion to quantify the quality of a DNN model, which refers to the ratio of correct predictions.

Number of disagreements is defined in [10] to characterize the difference between two DNNs. A disagreement is an input that triggers different outputs by the original model and its compressed version. By measuring the number of disagreements in the test data, one can observe the model's behavior change after quantization.

Area Under the Receiver Operating Characteristic Curve (AUC-ROC) [40] is a threshold-independent evaluation metric. In RQ3, we utilize the AUC-ROC score to measure the performance of the trained logistic regression classifiers.

F. Uncertainty Metrics

In RQ3, we utilize uncertainty metrics to estimate the characteristics of the disagreement inputs. Following previous studies [41], [42], we select 4 commonly used output-based uncertainty metrics in our study. Given a classification task, let DNN be a C-class model and x be an input. $p_i\left(x\right)$ denotes the predicted probability of x belonging to the ith class, $0 \leq i \leq C$. **Entropy** score [43] quantifies the uncertainty of x by Shannon entropy: $Entropy(x)$ = -$\sum_{i=1}^{C} p_i\left(x\right) \log p_i\left(x\right)$. **Gini** [44] score is calculated as: $Gini(x)$ = $1-\sum_{i=1}^{C}\left(p_i\left(x\right)\right)^2$. *Margin* [45] score is based on the top-2 prediction probabilities: $Margin(x) = Margin\left(x\right) = p_k\left(x\right) - p_j\left(x\right)$, where $k = \underset{i=1:C}{\arg\max}\left(p_i\left(x\right)\right)$ and $j = \underset{i=\{1:C\}/k}{\arg\max}\left(p_i\left(x\right)\right)$. **Least Confidence (LC)** [46] score is the difference between the most confident prediction and 100% confidence. $LC(x)$ = 1 - $p_k\left(x\right)$, where $k = \underset{i=1:C}{\arg\max}\left(p_i\left(x\right)\right)$.

IV. Configuration

Environments. We undertake model training and retraining on an NVIDIA Tesla V100 16G SXM2 GPU. For the TensorFlowLite model evaluation, we run experiments on a 2.6 GHz Intel Xeon Gold 6132 CPU. For the CoreML model evaluation, we conduct experiments on a MacBook Pro laptop with macOS Big Sur 11.0.1 with a 2GHz GHz QuadCore Intel Core i5 CPU with 16GB RAM.

Quantization. We apply the interfaces provided by TensorFLowLite and CoreML to accomplish post-training model quantization. For IMDb-related models, we only apply 16-bit float quantization by TensorFlowLite and utilize both 8-bit integer and 16-bit float quantization by CoreML. For other models, we conduct 8-bit integer and 16-bit float quantization using both techniques.

Model training. For the quantization-aware training, we mask layers (e.g., the BatchNormalization layer) that are not supported by the current TensorFlow framework. In addition, since TensorFlow does not support RNNs [47], we skip IMDb-related models in this experiment. Regarding the adversarial training, we employ the commonly used PGD-based [48] adversarial training for image datasets, and PWWS-based [49] adversarial training for text datasets. For the Mixup training, we follow the recommendation by the original paper to set the mixup parameter α as 0.2. For the training of the regression model used in RQ3, we use the default setting of *sklearn* framework and set the number of maximum iterations as 5000.

Model retraining. Following the same setting from the empirical study of model retraining [31], we add all disagreements into original training data to train the pre-trained model with additional several epochs (5 epochs for MNIST, IMDb, and iWildsCam, 10 epochs for CIFAR-10). All the detailed configurations can be found at our project site [22].

V. Experimental Results

A. RQ1: Behavior of Compressed Models

Table II presents the results of the behaviors of compressed models on ID test data and OOD test data with synthetic distribution shifts. We can see the accuracy in most cases degraded due to the loss of information during quantization, which is also demonstrated by the existing studies [9], [41]. However, surprisingly, almost 30% of (86 out of 292) opposite cases where compressed models hold higher accuracy than their original models. Particularly, in the case of $ResNet20, Zoom\_blur$, the compressed model has an improvement of 1.98%. On the other hand, this phenomenon also happens to the natural distribution shift (10 out of 16 cases in Table III). Regarding shifted data as natural adversarial examples, our finding confirms the conclusion from a recent research [50] that the quantization process can be useful to promote the model's adversarial robustness. In addition, the distribution shift can lead to larger change and should be taken into account during deployment. For example, in *MNIST, TF-8*, the compressed model has an accuracy change of 0.04% on ID test data but 0.78% under the Fog shift (Table II). And comparing the ID and OOD test sets, we found the synthetic distribution shift can increase the accuracy change by up to 3.03% (ResNet20-Gaussian_noise-CM-8).

Considering the disagreement, the results demonstrate that even if the compressed model maintains accuracy, there may exist disagreements. For example, in the case of *LeNet1, CM-8*, the accuracy change is 0, but the number of disagreements is 6. Even worse, in *DenseNet-121, CM-16*, 216 disagreements appear without any accuracy change. This calls for the attention that the behaviors of compressed models can not be exactly reflected by only comparing the test accuracy. Thus, during deployment, using accuracy only to evaluate the quality and reliability of compressed DNNs is insufficient.

TABLE II

BEHAVIOR OF COMPRESSED MODELS UNDER SYNTHETIC DISTRIBUTION SHIFT. NON-HIGHLIGHTED VALUE: ACCURACY CHANGE (%), HIGHLIGHTED VALUE: NUMBER OF DISAGREEMENTS. A LOW VALUE INDICATES A SMALL DIFFERENCE BETWEEN THE ORIGINAL AND COMPRESSED MODELS. ID REFERS TO THE ID TEST DATA AND THE OTHERS ARE OOD TEST DATA. TF: TENSORFLOWLITE. CM: COREML. —AVERAGE—: THE AVERAGE OF ABSOLUTE CHANGES.

Test Data	MNIST															
	LeNet1								LeNet5							
	TF-8		TF-16		CM-8		CM-16		TF-8		TF-16		CM-8		CM-16	
ID	-0.04	14	-0.04	9	0	2	0	0	0.02	4	0.01	3	0.01	1	0	0
Brightness	0.51	172	0.28	67	-0.04	53	0.01	7	-0.27	175	-0.56	100	-0.79	100	0.03	6
Canny_edges	0.77	172	0.5	86	0.02	51	-0.01	4	0.16	73	-0.06	35	-0.01	17	0.02	2
Dotted_line	-0.21	38	-0.06	24	-0.03	8	0	0	-0.01	26	-0.06	13	-0.04	12	0	0
Fog	0.78	542	0.11	133	-0.17	112	0	6	0.31	321	-0.43	112	-0.6	120	0.01	8
Glass_blur	-0.05	41	-0.04	18	-0.05	10	0	0	0.09	44	-0.1	23	-0.03	17	0	0
Identity	-0.04	14	-0.04	9	0	2	0	0	0.02	4	0.01	3	0.01	1	0	0
Impulse_noise	-0.23	77	0.06	28	-0.11	33	-0.04	4	-0.02	50	-0.13	35	0.01	23	-0.03	3
Motion_blur	0.14	79	0.08	29	-0.07	20	0.01	1	0.18	59	-0.11	23	-0.01	17	0.01	1
Rotate	-0.07	62	-0.03	30	0	13	-0.02	2	-0.11	30	-0.09	18	-0.05	10	0	0
Scale	-0.28	102	-0.15	43	-0.04	29	0	0	-0.05	53	-0.02	22	-0.02	8	0	0
Shear	0	22	0	8	0.01	11	0.01	2	0	22	-0.03	11	-0.01	5	0	0
Shot_noise	-0.06	26	-0.02	11	0	10	0	0	0.06	16	-0.01	7	-0.03	6	0	0
Spatter	-0.05	31	0.06	13	0.02	6	0	0	0.04	14	-0.02	10	0	8	-0.01	1
Stripe	-0.42	113	-0.1	70	0.18	103	-0.04	6	-0.03	88	-0.03	36	-0.19	53	0	1
Translate	-0.18	159	-0.08	70	-0.04	67	0	4	0.15	135	-0.05	64	0	47	-0.01	2
Zigzag	0.03	88	-0.03	41	-0.01	34	-0.04	7	-0.06	66	-0.17	34	-0.08	30	-0.01	1
—Average—	0.23	103	0.10	41	0.05	33	0.01	3	0.09	69	0.11	32	0.11	28	0.01	1
	CIFAR-10															
	NiN								ResNet20							
	TF-8		TF-16		CM-8		CM-16		TF-8		TF-16		CM-8		CM-16	
ID	-0.95	514	-0.1	24	0.03	45	0.02	7	0.04	456	-0.4	54	0.36	181	0.03	7
Brightness	-0.02	70	-0.01	50	0.03	44	0.01	9	-0.02	190	0.1	51	-0.08	170	-0.03	5
Contrast	0.04	78	-0.06	42	-0.06	48	-0.04	6	-0.12	250	0.04	49	0.02	187	-0.06	10
Defocus_blur	-0.1	51	-0.05	34	-0.06	37	-0.04	5	-0.21	205	0.01	53	0.02	175	-0.03	15
Elastic_transform	0.03	107	0.02	65	0.06	70	0.01	10	0.02	342	0	84	0.83	302	-0.05	18
Fog	-0.02	57	0.02	29	-0.06	37	-0.03	6	-0.07	236	0.01	45	-0.07	212	0	16
Frost	0.02	81	-0.05	53	0	50	0	10	-0.28	260	0.05	68	-0.76	317	0.01	14
Gaussian_blur	-0.05	56	-0.06	29	-0.04	36	-0.02	5	-0.18	207	0.05	60	0.04	161	0.03	12
Gaussian_noise	0.06	96	-0.03	57	0.06	65	-0.06	8	-0.35	343	0.06	106	-2.67	499	-0.15	29
Glass_blur	-0.36	164	-0.42	122	-0.02	90	-0.08	20	-0.09	559	0.12	168	-1.56	754	-0.14	57
Impulse_noise	-0.28	108	-0.34	86	-0.11	82	-0.05	18	-0.12	275	0.04	72	-1.1	329	-0.01	20
Jpeg_compression	-0.26	82	-0.15	55	-0.21	57	-0.05	12	-0.15	259	-0.08	75	-0.94	315	-0.06	21
Motion_blur	0	87	-0.06	43	0.14	67	-0.05	13	0.46	336	-0.04	83	1.69	325	-0.02	21
Pixelate	0.03	84	0	49	-0.06	53	-0.01	6	0.08	251	0.05	61	-0.32	239	-0.02	10
Saturate	-0.13	89	0	49	-0.05	49	-0.02	8	-0.16	262	-0.02	66	0.11	238	-0.01	22
Shot_noise	-0.06	105	-0.08	60	-0.09	69	-0.02	8	-0.18	302	0.03	79	-2.01	409	-0.12	23
Snow	-0.14	698	-0.03	67	-0.02	66	-0.02	5	-0.56	255	0.04	68	-0.84	291	-0.08	23
Spatter	-1.06	604	-0.02	43	0.08	52	0.02	4	-0.29	228	0.06	64	-0.38	239	-0.01	15
Speckle_noise	-1.89	631	-0.12	49	-0.06	65	0.01	5	-0.34	269	-0.01	70	-1.98	398	-0.04	13
Zoom_blur	0.6	883	-0.07	56	0.1	77	-0.04	12	0.25	415	-0.04	125	1.98	392	0.01	24
—Average—	0.31	232	0.08	53	0.07	58	0.03	9	0.20	295	0.06	75	0.89	307	0.05	19

TABLE III

BEHAVIOR OF COMPRESSED MODELS UNDER NATURAL DISTRIBUTION SHIFT. NON-HIGHLIGHTED VALUE: ACCURACY CHANGE (%), HIGHLIGHTED VALUE: NUMBER OF DISAGREEMENTS. A LOW VALUE INDICATES A SMALL DIFFERENCE BETWEEN THE ORIGINAL AND COMPRESSED MODELS. ID REFERS TO THE ID TEST DATA AND THE OTHERS ARE OOD TEST DATA. TF: TENSORFLOWLITE. CM: COREML. —AVERAGE—: THE AVERAGE OF ABSOLUTE CHANGES.

Test Data	iWildCam															
	DenseNet-121								ResNet50							
	TF-8		TF-16		CM-8		CM-16		TF-8		TF-16		CM-8		CM-16	
ID	-18.96	2830	-0.12	167	-8.34	2035	0.04	34	0	326	-0.11	187	-0.21	226	0.06	16
OOD	-10.91	14279	-0.42	1105	-5.18	11095	0	216	1.09	2158	0.6	1270	-0.33	1811	-0.01	128
—Average—	14.94	8555	0.27	636	6.76	6565	0.02	125	0.55	1242	0.35	729	0.27	1019	0.04	72
	IMDb															
	LSTM								GRU							
	TF-8		TF-16		CM-8		CM-16		TF-8		TF-16		CM-8		CM-16	
ID	-	-	-0.08	8	0	6	0	0	-	-	-0.06	3	0.04	2	0	0
CR	-	-	-0.04	8	-0.06	9	0	0	-	-	0.28	30	0.2	24	-0.02	1
Yelp	-	-	-0.12	14	0.02	7	0	0	-	-	0.06	9	0.08	8	0	0
—Average—	-	-	0.08	10	0.03	7	0.00	0	-	-	0.13	14	0.11	11	0.01	0

Moreover, comparing the number of disagreements from the ID test data and OOD test data, we observe that the distribution shift tends to lead to more disagreements. In 82% cases (241 of 294), the number of disagreements from OOD test data is greater than from ID test data, the difference can be by up to 5.28% (LeNet1, Fog, TF-8). However, after the model has been deployed and used in the wild, test data are more likely to have distribution shifts which raises a big concern that model quantization may bring unexpected errors.

Next, we compare the two quantization techniques considering the accuracy change. On average, regardless of the dataset, DNN, and quantization level, CoreML produces more stable compressed models (smaller change) than TensorFlowLite in most cases (12 out of 14). Concretely, in 16-bit float quantization, CoreML always outperforms TensorFlowLite. Take *iWildCam, DenseNet-121* as an example, in 16-bit level quantization, the average accuracy change is 0.27% by TensorFlowLite but only 0.02% by CoreML. This difference of 0.25% could cause the CoreML-compressed model to correctly predict 188 more data than the TensorFlowLite-compressed model, which is a considerable difference. In 8-bit integer quantization, CoreML can still outperform TensorFlowLite in most cases (4 out of 6). Additionally, we found an extreme case (*iWildCam, DensetNet-121*) where the accuracy of compressed models by both techniques drops a lot. This finding raises the concern that both quantization tools have room for improvement and require a thorough test. On the other hand, considering the number of disagreements, the models compressed by CoreML have fewer disagreement inputs than those by TensorFlowLite in most cases (13 out of 14).

Answer to RQ1: Under synthetic distribution shift, the accuracy change and the number of disagreements between the original and compressed models increase by up to 3.03% and 5.28%. Regardless of the dataset, DNN, and distribution shift, CoreML keeps the behaviors of original DNNs better than TensorFlowLite during deployment.

B. RQ2: Influence of Training Strategy

In this section, we explore how different training strategies influence the behaviors of compressed models. Here, we only report the results of one model from each dataset (MNIST-LeNet5, CIFAR-10-ResNet20, IMDb-LSTM, and iWildsCam-ResNet50). The whole results are available at our project site.

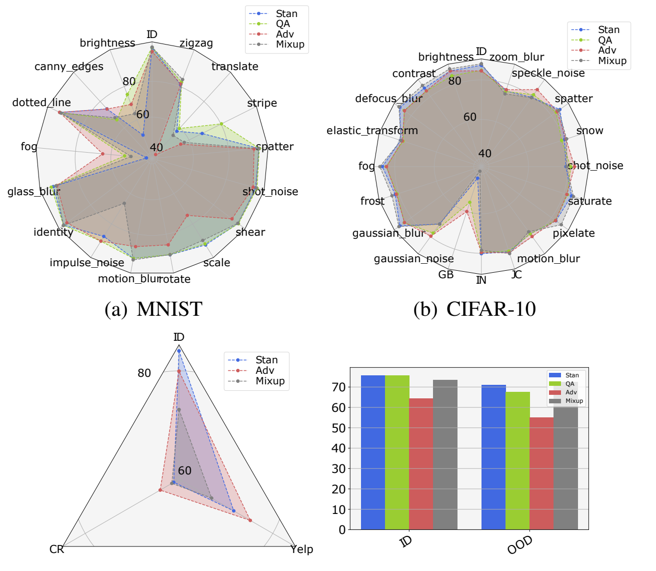
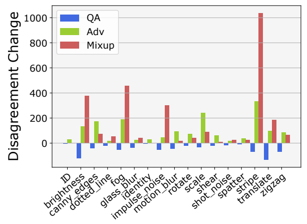
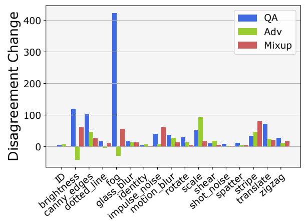


Fig. 3. Accuracy (%) of models (before quantization) trained by different training strategies. ID represents the accuracy on ID test datasets, and the others are on OOD test datasets. Stan: standard training. QA: quantization-aware training. Adv: adversarial training. Mixup: Mixup training. In CIFAR-10, GB, IN, and JC represent glass_blur, impulse_noise, and jpeg_compression.

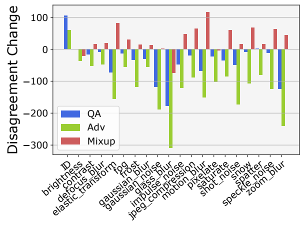
First, we evaluate the performance of each training strategy considering the distribution shift before model quantization. Figure 3 shows the results. Under synthetic distribution shift, for MNIST, there are 12, 5, and 6 cases out of 17 that using quantization-aware, adversarial, and Mixup training, respectively, improve the accuracy compared to using standard



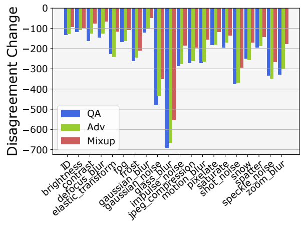
(a) MNIST, TensorFlowLite



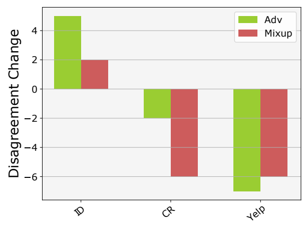
(b) MNIST, CoreML



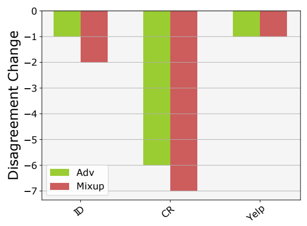
(c) CIFAR-10, TensorFlowLite



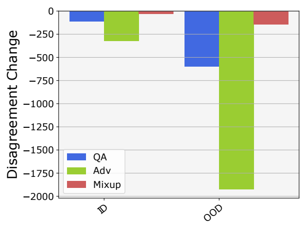
(d) CIFAR-10, CoreML



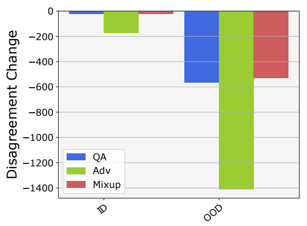
(e) IMDb, TensorFlowLite



(f) IMDb, CoreML



(g) iWildCam, TensorFlowLite



(h) iWildCam, CoreML

Fig. 4. The disagreement change of models trained by different training strategies compared to by standard training. QA: quantization-aware training. Adv: adversarial training. Mixup: Mixup training. $y-$axis: the difference in the number of disagreements between a training strategy and standard training.

TABLE IV
AVERAGE ACCURACY CHANGE (%) OF MODELS BY QUANTIZATION. A LOW AVERAGE VALUE INDICATES A SMALL DIFFERENCE BETWEEN THE ORIGINAL AND COMPRESSED MODELS. HIGHLIGHTED VALUES INDICATE THAT THE ACCURACY CHANGE BY THE CORRESPONDING TRAINING STRATEGY IS THE SAME AS OR SMALLER THAN BY STANDARD TRAINING.

Dataset	**Quantization**	**Training Strategy**			
		Standard	**Quantization-aware**	**Adversarial**	**Mixup**
MNIST	TensorFlowLite-8	0.09	0.06	0.36	0.53
	TensorFlowLite-16	0.11	0.01	0.09	0.24
	CoreML-8	0.11	0.06	0.11	0.09
	CoreML-16	0.01	0.01	0.01	0.02
CIFAR-10	TensorFlowLite-8	0.20	0.77	1.02	1.48
	TensorFlowLite-16	0.06	0.06	0.21	0.15
	CoreML-8	0.89	0.05	0.20	0.12
	CoreML-16	0.05	0.03	0.16	0.04
IMDb	TensorFlowLite-16	0.08	-	0.03	0.05
	CoreML-8	0.03	-	0.03	0.03
	CoreML-16	0.01	-	0.01	0.01
iWildCam	TensorFlowLite-8	0.55	0.47	0.24	0.29
	TensorFlowLite-16	0.35	0.05	0.10	0.07
	CoreML-8	0.27	0.19	0.04	0.62
	CoreML-16	0.04	0.11	0.04	0.06

training. While the result for CIFAR-10 changes to 5, 8, and 12 cases of 20 correspondingly. We conclude that none of these three training strategies can consistently deal with the issue of accuracy degradation under synthetic distribution shifts. On the other hand, under natural distribution shift, interestingly, when performing adversarial training for IMDb models, the accuracy of models on both distribution-shifted datasets (CR and $Yelp$) has been improved. We conjecture that the features of text adversarial examples are more likely to appear in the real-world OOD test dataset. For example, the original sentence *"a wonderful...are terribly **well done*** and its adversarial sentence *"a wonderful...are terribly **considerably perform**"* only have a two-word difference, but the model predicts them differently. The words *considerably* and *perform* are both in the vocabulary of OOD data. For iWildCam, only the Mixup training can improve the accuracy of models on shifted data.

Second, we check the accuracy change of each model trained by different training strategies after quantization. Table IV presents the results of the average accuracy change of all test datasets of each model. Compared to standard training, the compressed models by using the quantization-aware training are more stable where the accuracy change in most cases (10 out of 12) is the same as or smaller. For example, in *CIFAR-10, CM-8*, by standard training, the compressed model has an average of 0.89% difference compared to its original model. However, by quantization-aware training, the difference can decline to only 0.05%. By contrast, both adversarial and Mixup training can result in more stable (11 out of 15, 8 out of 15 cases) compressed models than standard training but not as well as quantization-aware training. In short, quantization-aware training outperforms adversarial and Mixup training considering minimizing the accuracy change during deployment.

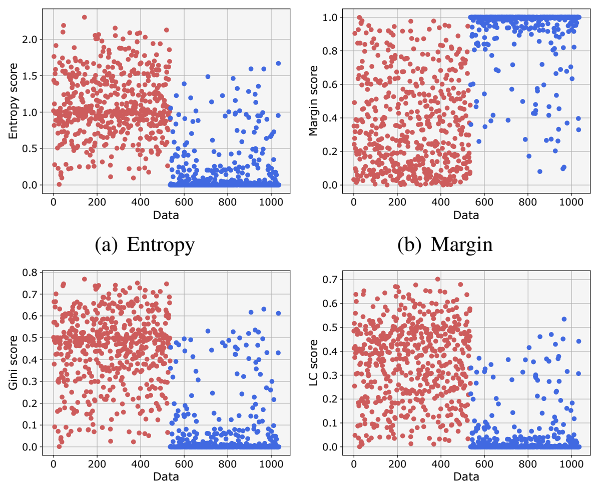
In addition, similar to the findings in RQ1, we observe that under synthetic distribution shift (MNIST and CIFAR-10), most (7 out of 8) of the accuracy change improvements happen in the models compressed by TensorFlowLite. And for the data with natural distribution shifts, the accuracy change increase only happens in the models compressed by CoreML. This phenomenon indicates that in terms of accuracy change, quantization-aware training produces more stable models than standard, adversarial, and Mixup training. TensorFlowLite is more suitable to deal with natural distribution shifts, while CoreML performs better for synthetic distribution shifts.

Finally, we check the disagreements that occur during model quantization. Figure 4 shows the disagreement change of models trained by different strategies compared to the standard training. Given all OOD test datasets, the quantization-aware equipped with TensorFlowLite can efficiently decrease the number of disagreements. Under synthetic distribution shift only, after TensorFlowLite quantization, the models trained by Mixup training lead to more disagreements. On the other hand, under natural distribution shift, all these tree training strategies are useful to reduce disagreements (negative disagreement change in Figures 4(e) - 4(h)) regardless of the quantization technique. We can conclude that under synthetic distribution shift, quantization-aware training is useful to remove disagreements for TensorFlowLite-compressed models. While under natural distribution shift, all three training strategies are efficient to reduce disagreements.

Answer to RQ2: Generally, quantization-aware training can produce more stable models with small accuracy changes and fewer disagreements after model quantization. For data with natural distribution shifts, both quantization-aware training and basic data augmentation training (adversarial training and Mixup training) can reduce the disagreements.

C. RQ3: Characteristic of Disagreements

Since disagreements are usually close to the decision boundaries of the model [10], we try to characterize the disagreements from the perspective of output uncertainty. Concretely, after quantization, the decision boundary of a model may slightly move due to the precision of parameter change. As a result, the data that are close to the boundary might cross over the boundary and cause disagreements. Generally, those data are uncertain to the model and could be identified by uncertainty metrics. Many metrics have been proposed but which one can be used to more precisely distinguish the disagreements and normal inputs is unclear. In our study, we consider four (Entropy, Margin, Gini, Least Confidence) widely used uncertainty metrics only based on the output of the model to determine the best one to present the property of disagreements.



(a) Entropy (b) Margin (c) Gini (d) Least Confidence

Fig. 5. An example (CIFAR-10, ResNet20, ID test data) of the distributions of output uncertainty scores. Red: disagreements. Blue: normal inputs.

Figure 5 gives an example (CIFAR-10, ResNet20) of the distribution of uncertainty scores of the disagreements and normal inputs. First of all, regardless of the uncertainty metric, the result confirms that disagreements are more uncertain for a model than normal inputs as they usually have higher (lower in Margin) uncertainty scores. Thus, output-based uncertainty is a promising indicator to distinguish disagreements and normal inputs. Take the least confidence as an example, most normal inputs have LC scores near 0. According to the definition of LC, the result demonstrates that the model is confident (with almost 100%) in the top-1 predictions for these inputs. In detail, the number of inputs having LC scores in the ranges of [0, 0.2], (0.2, 0.4], (0.4, 0.6], (0.6, 0.8], and (0.8, 1] are 462, 31, 7, 0, and 0 respectively. In contrast, for the disagreement inputs, most of them have high uncertain scores. Specifically, the number of inputs that the LC scores in the ranges of [0, 0.2], (0.2, 0.4], (0.4, 0.6], (0.6, 0.8], and (0.8, 1] are 110, 175, 225, 26, and 0 respectively.

TABLE V
$AUC - ROC$ SCORE OF THE LOGISTIC REGRESSION CLASSIFIERS TRAINED BY USING DIFFERENT UNCERTAINTY SCORES. THE BEST RESULTS AMONG THE FOUR UNCERTAINTY METRICS ARE HIGHLIGHTED.

Dataset	DNN	Training Strategy	Uncertainty Measure			
			Entropy	Gini	Margin	LC
MNIST	Lenet1	Standard	83.67	85.00	94.76	89.78
		Quantization-aware	95.81	95.20	97.45	96.61
		Adversarial	71.41	73.58	96.51	82.49
		Mixup	79.74	84.34	94.06	89.76
		Average	82.66	84.53	95.70	89.66
	Lenet5	Standard	86.79	89.49	97.36	94.49
		Quantization-aware	80.39	83.3	94.82	88.48
		Adversarial	72.02	76.47	96.78	85.09
		Mixup	71.53	72.00	89.42	78.37
		Average	77.68	80.32	94.60	86.61
CIFAR10	ResNet20	Standard	95.42	93.58	94.54	94.28
		Quantization-aware	95.29	95.62	96.52	96.11
		Adversarial	92.63	94.01	97.05	96.04
		Mixup	87.03	90.31	95.28	93.27
		Average	92.59	93.38	95.85	94.93
	NiN	Standard	93.36	94.88	96.2	95.65
		Quantization-aware	85.23	85.74	87.47	86.31
		Adversarial	93.79	94.96	96.25	95.64
		Mixup	88.59	89.98	93.15	91.85
		Average	90.24	91.39	93.27	92.36
IMDb	LSTM	Standard	100	100	100	100
		Adversarial	100	83.33	100	100
		Mixup	100	100	100	100
		Average	100	94.44	100	100
	GRU	Standard	100	100	100	100
		Adversarial	100	50.00	100	100
		Mixup	100	100	100	100
		Average	100	83.33	100	100
iWildCam	Densenet	Standard	78.67	85.00	85.60	85.83
		Quantization-aware	75.64	75.73	76.51	76.18
		Adversarial	61.71	62.04	63.35	62.28
		Mixup	82.57	79.98	82.46	80.98
		Average	74.65	75.90	76.98	76.32
	Resnet50	Standard	87.00	93.41	95.95	94.44
		Quantization-aware	89.71	91.60	97.36	94.79
		Adversarial	88.54	85.81	96.77	89.88
		Mixup	87.73	89.68	96.23	93.10
		Average	88.25	90.13	96.58	93.05

Table V presents the AUC-ROC scores of the regression classifiers trained by using the uncertainty scores. Overall, in most cases (27 out of 30), classifiers trained by *Margin* score have greater AUC-ROC scores than other classifiers, which means that the disagreement inputs and normal inputs have a bigger difference based on the *Margin* score. Specifically, in 23 (out of 30) cases, the classifiers trained using *Margin* score as the training data have greater than 90% AUC-ROC scores, which indicates the classifiers are useful to distinguish the normal inputs and disagreements. Besides, most IMDb classifiers have 100% AUC-ROC scores, the perfect results could come from the limited number of disagreements but can still prove the output-based uncertainty score is a promising indicator to represent the property of disagreements.

Figure 5 also shows a few disagreements where the model has high confidence. We call them **extreme disagreements**. We utilize the *Margin* score to set the threshold and analyze how many extreme disagreements exist and where do they come from. Concretely, we define the disagreements with *Margin* > 0.95 as extreme. We observe that there are 3, 226, 0, and 9 extreme disagreements in MNIST, CIFAR-10, IMDb, and iWildsCam, respectively. Interestingly, all the extreme disagreements come from the disagreements between TensorFlowLite-8bit compressed model and the original model, which means this quantization moves the decision boundary a lot in some *areas*. A deeper analysis could be an interesting research direction.

Answer to RQ3: Disagreements have closer top-1 and top-2 output probabilities than normal inputs. Compared to Entropy, Gini, and Least Confidence, Margin is a better metric to distinguish disagreements and normal inputs.

D. RQ4: Effectiveness of Retraining

In RQ3, we observe that the disagreements are data where the model has low confidence in the prediction. We investigate if model retraining, an efficient method to improve confidence, can ensure a stable compressed model during quantization.

Table VI presents the number of disagreements from the ID test data before and after model retraining. In most cases (18 out of 26 cases that have disagreements before retraining), the number of disagreements decreases after model retraining. However, surprisingly, there are some exceptions that the disagreements increase. For example, in *MNIST, LeNet1, CoreML-8*, 6 more disagreements appear after retraining. This phenomenon indicates that retraining the model using disagreements cannot always remove the disagreements.

TABLE VI
NUMBER OF DISAGREEMENTS BEFORE AND AFTER MODEL RETRAINING. IN TOTAL: DISAGREEMENTS IN A TEST DATASET REGARDLESS OF THE QUANTIZATION TECHNIQUE. VALUES IN BRACKETS ARE THE DIFFERENCE. STUBBORN: DISAGREEMENTS CANNOT BE REMOVED BY RETRAINING. NEW: DISAGREEMENTS APPEARING AFTER RETRAINING.

	Before	**After**	**Before**	**After**
MNIST	**LeNet1**		**LeNet5**	
TensorFlowLite-8	14	7(-7)	4	3(-1)
TensorFlowLite-16	9	4(-5)	3	1(-2)
CoreML-8	2	8(+6)	1	1(0)
CoreML-16	0	0(0)	0	0(0)
In total	15	16(+1)	6	4(-2)
Stubborn	1		0	
New	15		4	
CIFAR-10	**NiN**		**ResNet20**	
TensorFlowLite-8	514	371(-143)	456	439(-17)
TensorFlowLite-16	24	26(+2)	54	49(-5)
CoreML-8	45	31(-14)	181	56(-125)
CoreML-16	7	4(-3)	7	13(+6)
In Total	540	401(-139)	536	480(-56)
Stubborn	47		100	
New	354		380	
IMDb	**LSTM**		**GRU**	
TensorFlowLite-16	8	7(-1)	3	1(-2)
CoreML-8	6	2(-4)	2	0(-2)
CoreML-16	0	0(0)	0	0(0)
In Total	13	8(-5)	5	1(-4)
Stubborn	0		0	
New	8		1	
iWildCam	**DenseNet**		**ResNet50**	
TensorFlowLite-8	2830	3319(+489)	326	373(+17)
TensorFlowLite-16	167	12(-155)	187	143(-44)
CoreML-8	2035	7(-2028)	226	101(-125)
CoreML-16	34	2(-32)	16	27(+11)
In Total	3834	3324 (-510)	469	462 (-7)
Stubborn	2230		24	
New	1094		438	

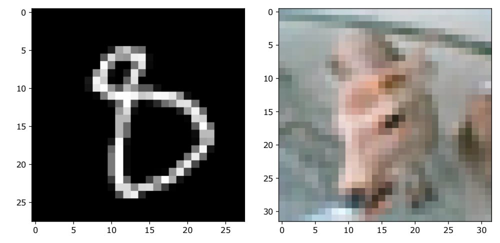

(a) MNIST-LeNet1 (b) CIFAR-10-ResNet20

Fig. 6. Examples of two stubborn disagreements. MNIST: predicted label before retraining: 1, 100% confidence, after: 8, 100% confidence. CIFAR-10: prediction before retraining: cat, 59% confidence, after: deer, 92% confidence.

In addition, we study whether the old disagreements are really removed by model retraining or not. To this end, we compare if the disagreements remain the same after retraining. For simplicity, we define the **stubborn disagreement** as the disagreement appearing both before and after retraining, and **new disagreement** as the disagreement introduced by retraining. Figure 6 gives two examples of stubborn disagreements. For the MNIST image, the model predicts the digital number as 0 or 9, while the true label is 8. For the CIFAR-10 image, the model hesitates to predict the animal to be a cat before retraining, and raises the confidence of this wrong prediction after retraining, while the true label is deer. Besides, we observe that the average *Margin* score of all the stubborn disagreements before and after retraining are 0.40 and 0.56, respectively. That means although models become more confident with these stubborn disagreements after retraining, their uncertainty is still high. In Table VI, regardless of the quantization technique, only a few stubborn disagreements remain after retraining. For example, in *CIFAR-10, NiN*, only 47 (of 540) disagreements are left. However, model retraining introduces new disagreements which have the same size as without retraining. For example, in *iWildCam, ResNet50*, through retraining, only 24 stubborn disagreements are left and all the other 445 are efficiently removed, but meanwhile, 438 new disagreements appear. We can conclude that through model retraining, only a few stubborn disagreements remain but a similar size of new disagreements is introduced.

Answer to RQ4: Retraining fails to reduce the total number of disagreements. Though it manages to remove some existing disagreements, it introduces as many new ones.

VI. DISCUSSION

A. Compressed Model Repair

We have verified that model retraining, the most common strategy to enhance performance, has limited functionality in removing disagreements. How to solve this issue is still an open problem. Based on our investigation, the disagreements are mainly the data with small *Margin* scores by compressed models. Therefore, the main challenge is how to improve confidence in the data. We provide two potential solutions. 1) **Online monitoring.** Before quantization, training multiple models to perform prediction can also improve confidence [51].

Concretely, we can divide data into different groups based on their *Margin* scores. For each group of data, a model is trained and compressed. 2) **Offline repair.** After quantization, build an ensemble model to perform prediction instead of the compressed model. Ensemble learning [52], [53] has been proven to effectively improve the predictive performance of a single model by taking weighted average confidence from multiple models. However, both solutions will increase the storage size since more models are required. As a result, there is a trade-off between fewer disagreements and efficient model quantization. Thus, designing a robust quantization method is still an ongoing and important direction.

B. Threats to Validity

First, the threats to validity come from the selected datasets and models. Regarding the datasets, we consider both image and text classification tasks and include OOD benchmark datasets with both synthetic and natural distribution shifts. All the datasets are widely used in previous studies. As for the models, we cover two types of DNN architectures, feed-forward neural network, e.g., ResNet, and recurrent neural network, e.g., LSTM. In addition, we take into account the model complexity and apply both simple and complex ones, such as LeNet1 and ResNet50. For each dataset, we employ two different models to eliminate the influence of selected models. An interesting research direction is to repeat our experiments on other tasks, such as the regression task.

Second, the training strategies and uncertainty metrics could be other threats to validity. For the training strategies, among all possible choices, we include the four most representative and common ones. Standard training is the most basic training procedure and should be taken as the baseline. Quantization-aware training is specifically designed for quantization. Mixup training is the first and basic data augmentation approach to improve the generalization of DNNs over different distribution shifts. Adversarial training is one of the most effective techniques to promote model robustness/generalization. For the uncertainty metrics, we tend to select metrics that require as few configurations as possible. The four metrics included in this work are all solely based on the output probabilities. This is to avoid the impact of uncontrollable factors. For example, the dropout-based uncertainty metric [54] needs to consider where to put the dropout layer and the dropout ratio.

VII. Related Work

A. Deep Learning Testing

As a critical phase in the software development life cycle [55], deep learning testing ensures the functionality of DL-based systems during deployment. Multiple testing methods have been proposed in recent years [56]–[60]. For example, from the perspective of deep learning models, Pei *et al.* proposed DeepXplore which borrows the idea from code coverage and defines neuron coverage to measure if the test set is enough or not. Later on, DeepGauge [61] defines some new coverage metrics, e.g., k-multisection Neuron Coverage and Neuron Boundary Coverage, and demonstrates their effectiveness compared to the basic neuron coverage. From the perspective of test data, several test generation [18], [62]–[64] and test selection [44], [65], [66] approaches have been proposed. Gao *et al.* proposed SENSEI [59] which utilizes genetic search to find the best image transformation methods (e.g., image rotate) to generate suitable data for training a more robust model. Chen *et al.* proposed PACE [65] which uses clustering methods and an MMD-critic algorithm to select a small size of test data to estimate the accuracy of the model. However, all of these works test the model before quantization, while our study mainly focuses on the analysis of the difference between the models before and after quantization.

There are two studies closely related to our work [10], [11]. Both of them generate test inputs that have different outputs between the original and compressed models. However, these works did not 1) study the properties of such disagreements; 2) try to solve the disagreements; 3) consider natural distribution shift, all of which are considered in our work.

B. Empirical Study for Deep Learning Systems

Empirical software engineering is one general way to practically analyze software systems. In recent years, multiple empirical studies for deep learning systems have been conducted to help understand such complex systems.

The empirical study by Zhang *et al.* [67] pointed out that model migration is one of the top-three common programming issues in developing deep learning applications. Noticing the lack of benchmark understanding of the migration and quantization, Guo *et al.* [9] investigated, for the deployment process, the performance of trained models when migrated/compressed to real mobile and web browsers. They focus on the impacts of the deployment process on prediction accuracy, time cost, and memory consumption. In addition to the accuracy, we further evaluate the robustness of a model, especially considering the synthetic and natural distribution shifts in the test data. Chen *et al.* [68] studied the faults when deploying deep learning models on mobile devices. They especially apply TensorFlowLite and CoreML in the deployment, which is also considered in our study. The difference with our study is that their empirical study explores the failures related to data preparation (datatype error), memory issues, dependency resolution error, and so on, while our study focuses on the differential behavior during deployment and retraining. Hu *et al.* [41] verified that model quantization has opposite impacts on different tasks in the setting of active learning. For example, after quantization, the model is less accurate in the image classification task while exhibiting better performance in the text classification task. In our study, since the labels of all data are available, we apply standard training.

VIII. Conclusion

In this paper, we conducted a systematic study to characterize and help people understand the behaviors of compressed models under different data distributions. Our results reveal that there are more disagreement inputs in data with distribution shift than in the original test data. Quantization-aware training is a useful training strategy to produce a model that has fewer disagreements after quantization. The disagreements are those data that have high uncertainty scores, and the *Margin* score is a more effective indicator to distinguish the normal inputs and disagreements. More importantly, we also demonstrated that the commonly used approach – retraining the model with disagreements has limited usefulness to remove the disagreements and repair compressed models. Based on our findings, we provide two future research directions to solve the disagreement issue. To support further research, we released our code, and models (before and after quantization) to be a new benchmark for studying the quantization problem.

Acknowledgments

This work is supported by the Luxembourg National Research Funds (FNR) through CORE project C18/IS/12669767/STELLAR/LeTraon.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, 2021.

[3] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales, "When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 142–150.

[4] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[5] R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. Choudhury, L. Decker *et al.*, "Project codenet: A large-scale ai for code dataset for learning a diversity of coding tasks," *arXiv preprint arXiv:2105.12655*, 2021.

[6] A. Masood and A. Hashmi, "Aiops: predictive analytics & machine learning in operations," in *Cognitive Computing Recipes*. Springer, 2019, pp. 359–382.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[8] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.

[9] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 810–822.

[10] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, "Diffchaser: Detecting disagreements for deep neural networks." in *IJCAI*, 2019, pp. 5772–5778.

[11] Y. Tian, W. Zhang, M. Wen, S.-C. Cheung, C. Sun, S. Ma, and Y. Jiang, "Fast test input generation for finding deviated behaviors in compressed deep neural network," *arXiv preprint arXiv:2112.02819*, 2021.

[12] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "A comprehensive study on challenges in deploying deep learning based software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 750–762.

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[14] M. Thakkar, *Beginning machine learning in ios: CoreML framework*, 1st ed. APress, 2019.

[15] P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao *et al.*, "Wilds: A benchmark of in-the-wild distribution shifts," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5637–5664.

[16] D. Berend, X. Xie, L. Ma, L. Zhou, Y. Liu, C. Xu, and J. Zhao, "Cats are not fish: Deep learning testing calls for out-of-distribution awareness," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 1041–1052.

[17] R. Hu, J. Sang, J. Wang, and C. Jiang, "Understanding and testing generalization of deep networks on out-of-distribution data," *arXiv preprint arXiv:2111.09190*, 2021.

[18] S. Dola, M. B. Dwyer, and M. L. Soffa, "Distribution-aware testing of neural networks using generative models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 226–237.

[19] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[21] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[22] "Project website," 2023. [Online]. Available: https://github.com/Anony4paper/quan_study

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[24] G. Shomron, F. Gabbay, S. Kurzum, and U. Weiser, "Post-training sparsity-aware quantization," *arXiv preprint arXiv:2105.11010*, 2021.

[25] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Accurate post training quantization with small calibration sets," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 4466–4475. [Online]. Available: https://proceedings.mlr.press/v139/hubara21a.html

[26] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "Brecq: Pushing the limit of post-training quantization by block reconstruction," *arXiv preprint arXiv:2102.05426*, 2021.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[29] N. Mu and J. Gilmer, "Mnist-c: A robustness benchmark for computer vision," *arXiv preprint arXiv:1906.02337*, 2019.

[30] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[31] Q. Hu, Y. Guo, M. Cordy, X. Xie, L. Ma, M. Papadakis, and Y. Le Traon, "An empirical study on data distribution-aware test selection for deep learning enhancement (in press)," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2022. [Online]. Available: https://orbilu.uni.lu/handle/10993/50265

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Tech. Rep., 2009.

[33] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[35] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[37] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[38] 2022. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/35194

[39] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty," *arXiv preprint arXiv:1912.02781*, 2019.

[40] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[41] Q. Hu, Y. Guo, M. Cordy, X. Xie, W. Ma, M. Papadakis, and Y. Le Traon, "Towards exploring the limitations of active learning: An empirical study," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 2021, pp. 917–929.

[42] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, and Y. L. Traon, "Test selection for deep learning systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–22, 2021.

[43] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[44] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 177–188.

[45] D. Wang and Y. Shang, "A new active labeling method for deep learning," in *2014 International joint conference on neural networks (IJCNN).* IEEE, 2014, pp. 112–119.

[46] B. Settles, "Active learning literature survey," 2009.

[47] 2022. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/25563

[48] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[49] S. Ren, Y. Deng, K. He, and W. Che, "Generating natural language adversarial examples through probability weighted word saliency," in *Proceedings of the 57th annual meeting of the association for computational linguistics*, 2019, pp. 1085–1097.

[50] Y. Fu, Q. Yu, M. Li, V. Chandra, and Y. Lin, "Double-win quant: Aggressively winning robustness of quantized deep neural networks via random precision training and inference," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 3492–3504. [Online]. Available: https://proceedings.mlr.press/v139/fu21c.html

[51] P. Bielik and M. Vechev, "Adversarial robustness for code," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 896–907. [Online]. Available: https://proceedings.mlr.press/v119/bielik20a.html

[52] O. Sagi and L. Rokach, "Ensemble learning: a survey," *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1249

[53] L. Li, Q. Hu, X. Wu, and D. Yu, "Exploration of classification confidence in ensemble learning," *Pattern Recognition*, vol. 47, no. 9, pp. 3120–3131, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320314001198

[54] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning.* PMLR, 2016, pp. 1050–1059.

[55] L. Ma, F. Juefei-Xu, M. Xue, Q. Hu, S. Chen, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "Secure deep learning engineering: A software quality assurance perspective," *arXiv preprint arXiv:1810.04538*, 2018.

[56] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[57] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).* IEEE, 2019, pp. 1039–1049.

[58] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[59] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE).* IEEE, 2020, pp. 1147–1158.

[60] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, and J. Zhao, "Deepmutation++: A mutation testing framework for deep learning systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 2019, pp. 1158–1161.

[61] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 120–131.

[62] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.

[63] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 739–743.

[64] V. Riccio, N. Humbatova, G. Jahangirova, and P. Tonella, "Deepmetis: Augmenting a deep learning test set to increase its mutation score," *arXiv preprint arXiv:2109.07514*, 2021.

[65] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient deep neural network testing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–35, 2020.

[66] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "Boosting operational dnn testing efficiency through conditioning," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 499–509.

[67] T. Zhang, C. Gao, L. Ma, M. R. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 104–115, 2019.

[68] Z. Chen, H. Yao, Y. Lou, Y. Cao, Y. Liu, H. Wang, and X. Liu, "An empirical study on deployment faults of deep learning based mobile applications," *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 674–685, 2021.