POLITECNICO DI TORINO

MASTER's Degree in COMPUTER ENGINEERING

Politecnico di Torino

Department of Electronics and Telecommunications

MASTER's Degree Thesis

Leveraging Quantization and Approximate Computing to Enhance Adversarial Defense in Deep Neural Networks

Supervisors

Prof. Maurizio MARTINA

Prof. Guido MASERA

MSc. Flavia GUELLA

Candidate

Michael ELIAS

February 2025

Abstract

Over the last few years, Convolutional Neural Networks (CNNs) and other deep neural network architectures have been used increasingly across multiple domains. Such as computer vision, autonomous driving, and medicine. This widespread usage of CNNs has exposed them to adversarial attacks: applying deliberate perturbation to input data with the goal of forcing the CNN to produce wrong results.

Quantization and Approximate Computing (AC) were originally introduced to reduce CNNs' memory and computational cost. Furthermore, recent works have demonstrated that the noise they introduce could enhance input features, thereby reducing the likelihood of the adversarial fooling the CNN. In this study, we explore the effect of quantization and AC on the robustness of CNNs.

We propose a software framework to train and evaluate quantized CNNs with support for layerwise approximation. Moreover, the framework provides adversarial data generation for various attack types, in addition to Quantization Aware Training (QAT), and adversarial training, allowing for extensive exploration. A multiplier architecture with 256 approximation levels is chosen and integrated into the framework using Look Up Tables (LUTs). Considering layerwise configuration with 256 levels available for selection, exhaustive evaluation of approximate level configurations is infeasible. Therefore, the genetic algorithm NSGA-II is used to find optimal configurations by maximizing adversarial and standard accuracy.

Quantized CNNs led to an increase in adversarial accuracy of around 50% for a black-box attack and around 30% for a white-box attack, depending on the attack type and the chosen CNN architecture. For ResNet-32, AC led to a further increase in adversarial accuracy by 2-6%. However, this came at a cost: a 2% increase in adversarial accuracy had no effect on standard accuracy, whereas a 6% increase resulted in a 2% drop in standard accuracy.

These results show that quantization effectively defends against adversarial attacks by significantly enhancing CNN robustness. While approximate computing offers only a modest improvement, it does not act antagonistically to quantization.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Convolutional Neural Networks (CNNs) and other deep learning architectures have achieved remarkable success in various fields, including computer vision [1], robotics [2], and autonomous driving [3]. As CNNs rapidly expand across several fields, the threats targeting them are also on the rise, adversarial attacks are not just a theory as they have been demonstrated in real-world scenarios [4, 5].

Multiple examples of adversarial attacks have been developed and successfully tested on Deep Neural Networks (DNNs), such as Fast Gradient Sign Method (FGSM) [6], Projected Gradient Descent (PGD) [7], and Square Attack [8]. The mechanism of these attacks involves carefully distorting input images, such that the alterations are undetectable by the human eye, yet are able to deceive the CNN into an incorrect prediction.

Since adversarial attacks pose a serious threat to mission-critical applications that employ CNNs, it is mandatory to explore different techniques that are used to defend against such attacks. Some of the most well-known techniques include quantization, Adversarial Training (AT), and Approximate Computing (AC). In this work, we examine how these techniques perform both individually and in combination to gain a more comprehensive understanding of their impact.

1.2 Proposed Framework and Methodology

In this work, we propose a modified version of the MARLIN framework [9], with the added features of Quantization-Aware Training (QAT), fast adversarial training, GPU acceleration for training and evaluating CNN models with layerwise configuration of approximation levels, enabled by the TransAxx framework [10] and the generation of adversarial data using the Torchattacks library [11].

The CNN architectures used in this study are ResNet-8, ResNet-20, ResNet-32, and ResNet-56 [12], while the CIFAR-10 dataset [13] is used for training and evaluation. In light of this, and the fact that the approximate multiplier used has 256 different approximation levels, we can compute the number of configurations that a CNN model may have. For example, considering ResNet-32, each layer can have an approximation level ranging from 0 to 255, so the total number of combinations is 256^{32}, making an exhaustive search for Pareto-optimal configurations infeasible. For this reason, the genetic algorithm NSGA-II was adopted to search for the Pareto-optimal configurations, it was tuned to maximize adversarial accuracy and standard accuracy. Adversarial accuracy refers to the model's accuracy when evaluated on perturbed data, while standard accuracy refers to the model's accuracy when evaluated on non-perturbed, standard data. Therefore, by using this framework, we can easily assess the effectiveness of these defense mechanisms against adversarial attacks, considering varying perturbation budgets and both white-box and black-box attack scenarios.

This thesis is organized as follows:

- **Chapter 2** provides background on the key concepts and techniques used in this work, including CNNs, quantization, AC, adversarial attacks, and defense mechanisms.
- **Chapter 3** examines the most relevant literature on this topic and demonstrates how the framework proposed in this thesis is superior by offering a larger exploration space.
- **Chapter 4** explains the methodology adopted to evaluate the CNNs' robustness, detailing the used frameworks, training process, and threat model.
- **Chapter 5** displays the obtained results, explaining their significance, trends, and implications in the context of the study.
- **Chapter 6** presents the conclusions of this work, summarizing the key findings on the impact of quantization, AC, and AT on CNN robustness and their broader implications.

Chapter 2

Background

2.1 Convolutional Neural Networks

CNNs are a type of deep neural network, mostly used for image classification, object detection, video processing, and speech recognition [14]. CNNs are composed of three main types of layers: convolutional layer, pooling layer, and fully connected layer. Convolutional layers are typically followed by an activation function, which adds nonlinearity to the network and allows for more complex decision making. The general architecture of a CNN is shown in Figure 2.1 [15].

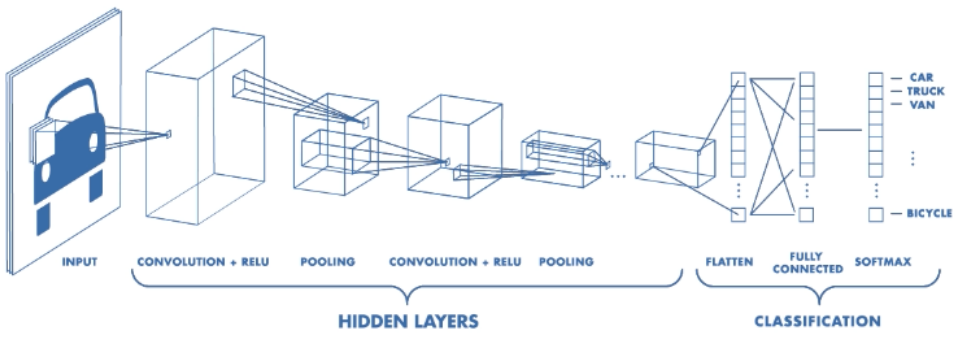

Figure 2.1: Architecture of a CNN

Convolutional layers These layers apply convolution operations to the input, which consists of sliding a small matrix (called filter or kernel) over the input, the typical size of a filter is 3x3, 5x5, or 7x7, and the depth of the filter is equal to the

number of channels of the input. The first convolutional layers are able to extract information from the input images, such as edges or patterns, in the later layers they can detect parts of objects or even complete objects or complex geometrical shapes [16]. A visual representation of a convolution operation can be seen in Figure 2.2 [17]. It should be noted that filters are learnable parameters.

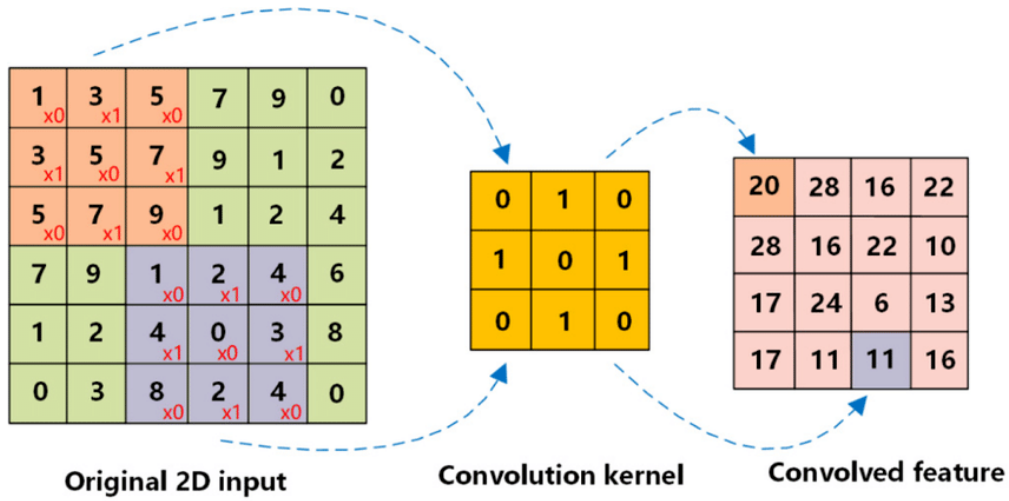

Figure 2.2: Convolutional operation with a 6 ×6 input map, 3×3 convolution kernel, 1 stride size, and no padding

Two parameters need to be defined for a convolutional operation: stride and padding. The stride is the number of pixels by which the filter is incremented after each operation. Padding is applied to increase the width and height of the input image by a specific number of pixels around its border, the value of the added pixels can be 0 or a constant. This way the size of the output matrix, also called feature map, can be controlled by changing these parameters, the formula used is the following:

$$o = floor(\frac{i + 2p - k}{s}) + 1 \tag{2.1}$$

Where:

- `o` is the size of the output matrix.
- `i` is the size of the input image.
- `p` is the amount of padding.
- `k` is the kernel size (height or width).
- `s` is the stride.

The depth of the feature map is equal to the number of filters that are used in the convolution operation.

Pooling layers These layers do not contain any learnable parameters and are usually placed after convolutional layers. Their role is to reduce the spatial dimension of the input feature maps while preserving the most essential information. The pooling operation also consists of a filter, often with a size of 2x2, sliding over an input feature. The exact operation carried out depends on the pooling type, the most common being max pooling and average pooling. Max pooling selects the maximum value from each submatrix of the input feature map, while average pooling computes the average of all the coefficients of the submatrix. Figure 2.3 and Figure 2.4 ([18]) illustrate both of these pooling techniques.

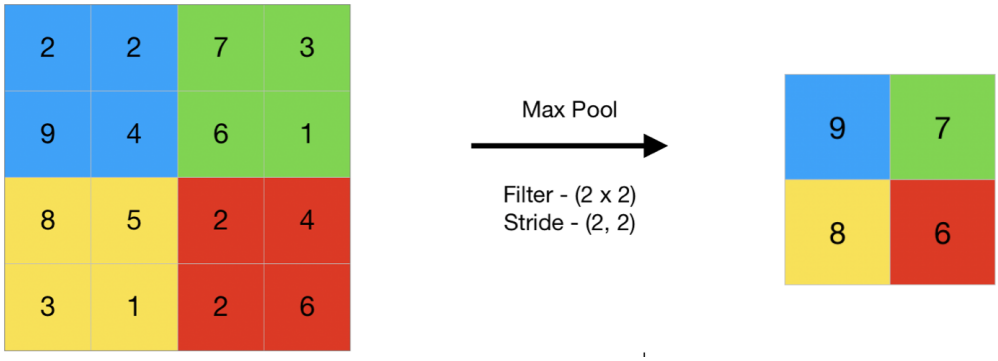

Figure 2.3: Max Pooling

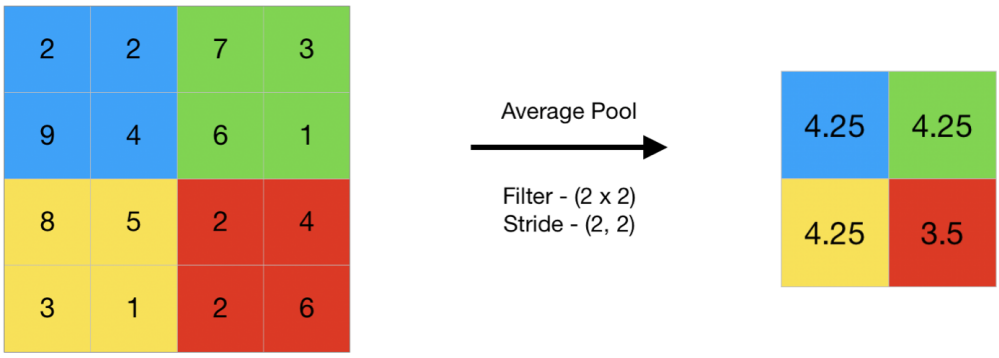

Figure 2.4: Average Pooling

Fully connected layers CNNs generally have one or more Fully Connected (FC) layers at the end of the network, preceded by a flattening layer that transforms the

multidimensional feature map into a vector, which is then fed to the FC layer. The final FC layer contains a number of neurons equal to the number of classes to be recognized.

2.1.1 Residual Network

The Residual Network (ResNet) architecture is a modern CNN architecture that utilizes residual learning through the use of shortcut connections [12]. It was developed to solve the vanishing and exploding gradient problem for DNNs [19, 20]. The basic block implementing the shortcut connection is seen in Figure 2.5 [12].

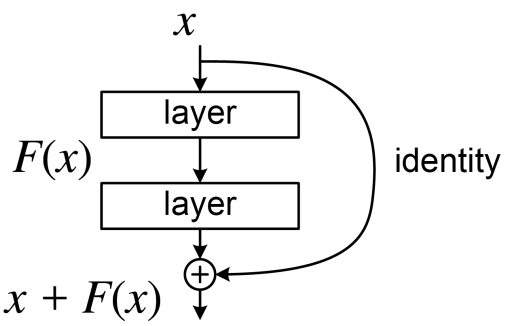

Figure 2.5: Building block for residual learning

The complete architecture of ResNet consists of stacking a number of these blocks, periodically doubling the number of filters in the convolutional layers. Downsampling is applied in cases where the input and output sizes of the residual block don't match.

2.2 Quantization and Approximate Computing

DNNs are typically deployed on cloud servers with a large number of supercomputers due to their significant demand in terms of memory and computational resources. However, due to the emergence of intelligent applications such as Augmented Reality (AR), Virtual Reality (VR), mobile assistants, Internet of Things, deploying DNNs on resource-constrained edge devices has become necessary [21].

2.2.1 Quantization

Quantization is considered one of the most effective ways to decrease the energy consumption and memory usage for DNNs, it consists of transforming the weights and activation tensors to a lower bit precision, such as 8-bit fixed-point, compared

to the 32-bit floating-point precision they are usually trained in. When quantized to an 8-bit fixed-point representation, the saved energy can reach up to 85.41%, while achieving little to no accuracy drop [22]. There are two forms of quantization: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). The former consists of quantizing a full-precision trained network without any additional training after quantization, while the latter involves either training a quantized network from scratch or quantizing a pre-trained full-precision network and retraining it for a number of epochs.

2.2.2 AC

The core operation in the inference phase for DNNs is matrix multiplication, which consists of element-wise multiplication and addition. Since multiplication consumes four times more energy than addition [23], it is logical to focus on designing approximate multipliers to reduce the total energy consumption. Using approximate multipliers in DNNs can lead to a 32% decrease in energy consumption with a maximum loss in accuracy by 2.4% relative to exact multiplication [24].

2.2.3 Quantization and AC for Defense

Another line of research investigating these techniques focuses on their impact on the robustness of CNNs [25, 26, 27, 28]. The main reason for CNNs' vulnerability against adversarial attacks is their linear nature [6]. An approximate multiplier introduces input-dependent noise at its output [26], where the magnitude of the noise scales with the input size, this enhances the features of the input image, making the adversarial attack less likely to succeed. As for quantization, since adversarial attacks consist of introducing small perturbations to the input images, it is believed that quantization, which involves precision reduction, can help filter out these adversarial perturbations [29].

2.3 Adversarial Attacks

This section provides an overview of techniques for generating adversarial attacks. Adversarial attacks consist of carefully modifying the original input image such that the CNN misclassifies it while remaining imperceptible to the human eyes. In the following, we will discuss the four most commonly used types of adversarial attacks in the literature:

1. Fast Gradient Sign Method (FGSM): Letting θ be the parameters of a DNN, X the input to the DNN, and Y the class label corresponding to input X, the

perturbation for this attack is computed as follows [6]:

$$\eta = \epsilon \operatorname{sign}(\nabla_X J(\theta, X, Y)) \tag{2.2}$$

Where ϵ is a configurable parameter that controls the severity of the attack, and η is the perturbation added to the original image to generate the adversarial input, as shown below:

$$X_{adv} = X + \eta \tag{2.3}$$

Visualization of such an attack is shown in Figure 2.6 [6], by injecting an imperceptible noise into the original image, the model misclassified the panda as a gibbon with 99.3% confidence.

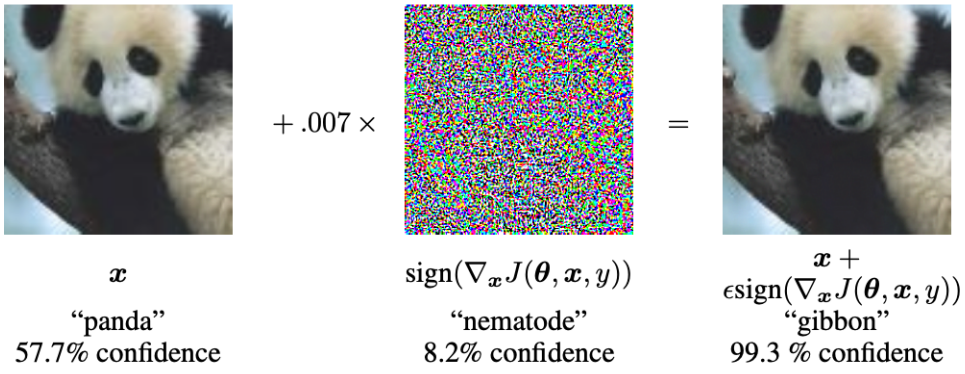

Figure 2.6: A demonstration of an FGSM attack applied to GoogLeNet [30] on ImageNet.

2. Basic Iterative Method (BIM): It is an iterative version of FGSM, it consists of performing an FGSM attack multiple times with a small step size and also clips the value of the adversarial input so that it remains within ϵ-neighborhood of the original image. The equation implementing this type of attack is shown below:

$$X_{adv}^{N+1} = clip_{X,\epsilon}\left[X_{adv}^{N} + \alpha * sign\left(\nabla_X J(\theta, X_{adv}^{N}, Y)\right)\right] \tag{2.4}$$

Where X_{adv}^{N} is the adversarial input generated at iteration N, the adversarial input is initialized as such: $X_{adv}^{0} = X$. α is the step size in each iteration.

3. Projected Gradient Descent (PGD): Is also an iterative method and very similar to BIM in its implementation. The only difference is in the initialization of the first iteration, instead of being the original image as in BIM, this method uses a randomized noise added to the original image [7]. As shown below:

$$X_{adv}^{0} = X + random(-\epsilon, \epsilon) \tag{2.5}$$

The equation defining X_{adv}^{N+1} is exactly the same as the one for BIM.

4. Carlini & Wagner (CW): The CW approach can be summarized in an optimization problem as follows [31]:

$$\begin{aligned} &minimize\ \mathcal{D}(x, x+\delta) \\ &such\ that\ C(X+\delta) = t \\ &\qquad x+\delta \in [0,1]^n \end{aligned} \tag{2.6}$$

Where $\mathcal{D}$ is a distance metric which can refer to one of the L_1, L_2, or L_∞ norms. The function C represents the classification function of the NN, X is the original input image, δ is the perturbation value, and t is the target class, which differs from the original classification. The original classification is the class assigned to the input without any perturbation.

This problem is too complex for existing optimization algorithms to solve, due to the equation $C(X+\delta) = t$ being non-linear. For this reason, the problem is reformulated by defining an objective function f such that $C(X+\delta) = t$ if and only if $f(X+\delta) \leq 0$. This allows for an alternative formulation:

$$\begin{aligned} &minimize\ ||\delta||_p + c.f(X+\delta) \\ &such\ that\ X+\delta \in [0,1]^n \end{aligned} \tag{2.7}$$

Where c is a configurable parameter used to scale the minimization function, it does not impact the final result, and $||\delta||_p$ is equivalent to the distance metric $\mathcal{D}$, so it can also take the form of an L_1, L_2, or L_∞ norm.

2.4 Defense Mechanisms

In addition to quantization and approximation computing, other defense techniques have been developed and evaluated, such as adversarial training, Data Preprocessing, and Gradient Masking. A brief overview of these techniques will be given in this section, highlighting their advantages and disadvantages.

1. Adversarial Training: It consists of feeding the network with perturbed inputs during the training phase, to enable the trained model to correctly classify the perturbed and standard data in the inference phase [32]. This technique has been proven to remarkably increase NNs' robustness against adversarial attacks [32]. However, the main downside of adversarial training is its considerably higher computational cost compared to standard training, potentially taking

up to 14 times longer depending on the model used [33]. Fortunately, various approaches have been explored to mitigate this issue, such as using FGSM with random initialization instead of PGD for adversarial training [34].

2. Data Preprocessing: Several data preprocessing methods have emerged as defensive measures against adversarial attacks, such as PixelDefend [35], feature compression [36], and randomization[37]. In essence, all these methods process the perturbed input in a specific way to destroy or suppress the injected noise, such as compressing the input data, randomly resizing followed by random padding, or flipping the image. Although some data preprocessing techniques have been shown to defend against adversarial attacks, others have been found to be weak against white-box attacks [38].

3. Gradient Masking: Since the most prominent white-box attacks are gradient-based, masking or hiding the gradient in a way that prevents attackers from exploiting it would be an effective defense against these attacks. Gradient masking is typically achieved by introducing random noise or perturbations to obscure gradient information [38]. On the negative side, non-gradient-based attacks, such as the Square attack [8], are not affected by gradient masking.

Chapter 3

Related Work

Several previous studies have investigated the impact of quantization and AC on the robustness of CNNs, yielding varying results. In this section, we will review the most relevant research to this thesis, emphasizing the limitations of these studies and demonstrating how this work addresses them.

In [25], the authors constructed an Ensemble of Approximate Multipliers (EAM) by modifying the fraction multiplier unit in the floating-point multiplier. The exact compressor in the fraction multiplier was replaced by one exact compressor and P approximate compressors, all fed by the same input and their output multiplexed. Therefore, by controlling the select signal of the multiplexer, they have $P + 1$ multipliers to choose from. The EAM was integrated inside a CNN and used to evaluate its robustness. The operation of the CNN consists of computing $P + 1$ outputs for every input, then the final classification is decided by averaging or maximum voting. This technique led to a significant increase in adversarial accuracy, between 15% to 45% depending on the attack type for CIFAR-10. However, this approach has major limitations, since it does not support layerwise configuration, i.e., the same multiplier type is used for all the layers of the CNN, with only one exact multiplier and 12 approximate multipliers available, the exploration space is extremely restricted compared to our layerwise configurable multiplier with 256 approximation level. Moreover, as each input requires the CNN to run $1 + P$ times for each input instead of once compared to regular CNNs, it led to a dramatic increase in inference time. For example, using all 12 available approximate multipliers results in a 13-fold increase in inference time, such drawback is not present in this work.

Another study, [39], arrived at a different set of results compared to the previous work. The authors claim that AC does not increase CNNs' accuracy against adversarial attacks while quantization does and that AC acts antagonistically to quantization. However, since the approximate multipliers used were 8-bit multipliers and the multipliers used in the previous study were approximate floating-point

multipliers, we cannot justifiably compare the two results. Now Compared to this thesis, the results align in showing that quantization does increase CNNs' robustness. The only difference is the impact of AC with quantization, we found that it increases the adversarial accuracy by a small margin, but doesn't act antagonistically to quantization. This variance can be explained by the two weaknesses of their methodology: firstly, similar to the previous study, only 9 approximate multipliers were used; secondly, no Quantization-Aware Training (QAT) was applied in this work.

The flaws identified in the previous two studies apply to some others as well, for example [26] uses only one approximate multiplier, and [40] does not apply QAT.

While several studies present an adequate methodology, they are limited to exploring only one defensive technique. For instance, [27] and [28] investigate quantization only, [41] investigates quantization coupled with Adversarial Training (AT), [25] explores AC only, and [34] explores AT only. On the other hand, the framework developed in this thesis allows for combining all the mentioned techniques, namely, AC, quantization, AT, and QAT.

Chapter 4

Methodology

In this chapter, we will cover all the frameworks and libraries used, explaining how they work and the modifications they underwent to better align with our goals. Moreover, we will present the adopted methodology to train the CNNs, and the threat model used to evaluate the CNNs' robustness against adversarial attacks. The methodology is summarized in Figure 4.1

4.1 MARLIN

The MARLIN framework [9] is considered the starting point of this work, it contains the definitions of ResNet-8, ResNet-20, ResNet-32, and ResNet-56 in PyTorch [42]. For ease of explanation, we introduce the term "execution type", which refers to the model type, either float or quantized. The default execution type for models defined in PyTorch is float, so there is no need for any additional definitions to support such a mode. However, that is not the case for a quantized model, as it is not inherently defined by PyTorch. For this reason, custom layers were developed, including a convolutional layer, a linear layer, and a custom activation function. The CNN can be quantized to any bit precision, such as 8-bit, 4-bit, etc. The weights, activations, and biases are quantized to the chosen precision, by applying this formula:

$$quantized\_tensor = \frac{clamp[round(scaling\_factor * real\_tensor), min\_v, max\_v]}{scaling\_factor} \tag{4.1}$$

Where:

- *clamp* is a bounding function, if the result is outside the range of $[min\_v, max\_v]$, it is clamped to the nearest bound, ensuring the value stays within the specified range.

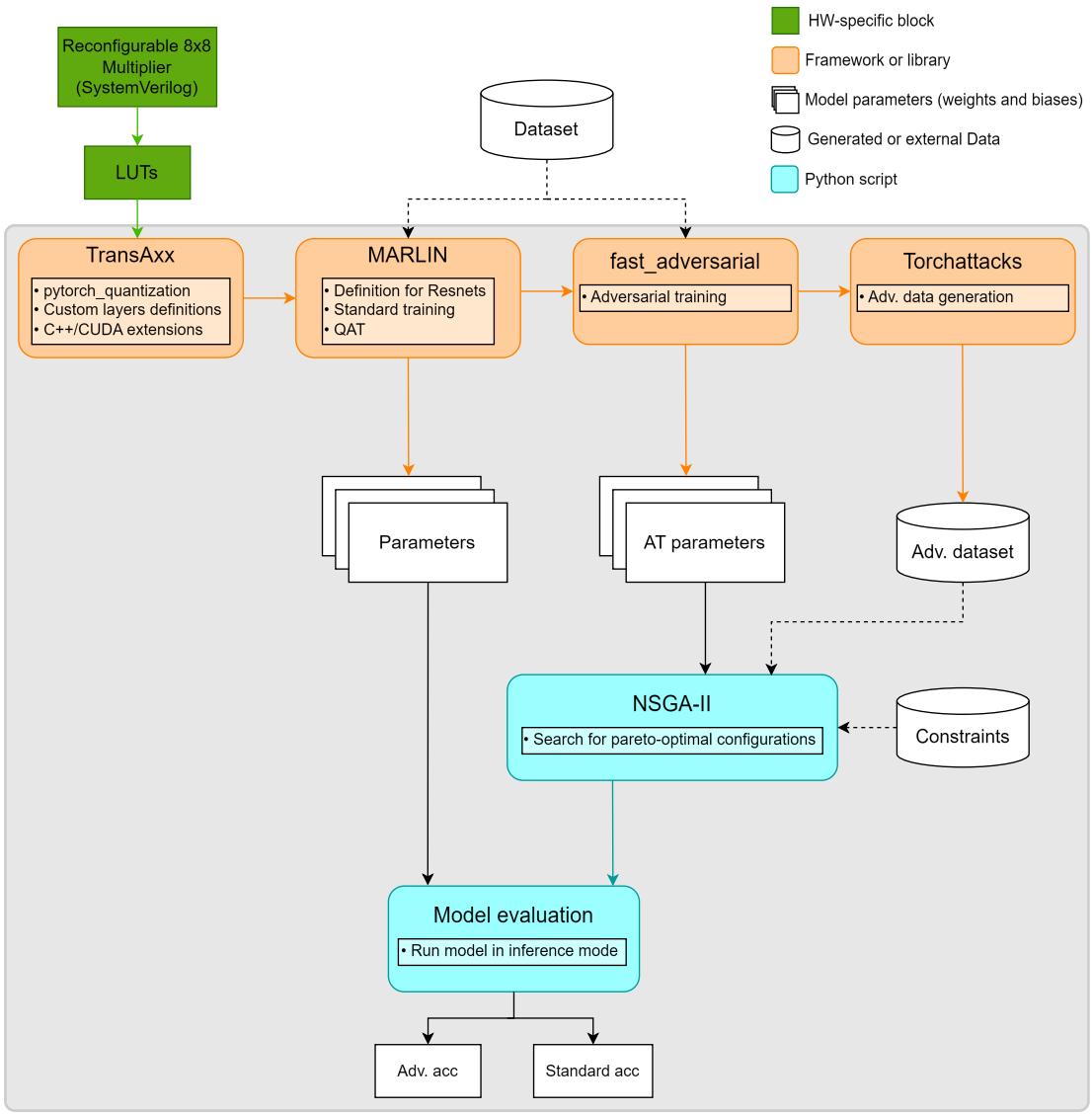

Figure 4.1: Workflow showing the methodology used to evaluate approximate CNNs.

- *round* function rounds the result to the nearest integer value.
- *real_tensor* is the real value of the tensor before any quantization (typically has a floating-point representation).
- *scaling_factor* is the constant used to multiply the *real_tensor* to scale it to the desired precision

- min_v and max_v are the minimum and maximum values that could be taken by the $quantized\_tensor$ depending on the desired precision. For example, if the chosen precision is 8-bit, min_v and max_v would be -128 and 127, respectively.

The $scaling\_factor$ is computed as follows:

$$scaling\_factor = \frac{max\_v}{max[|min(real\_tensor)|, |max(real\_tensor)|]} \tag{4.2}$$

Where:

- max_v is the same as the one in the previous equation.
- The inner max and min functions return the maximum and minimum value of all elements in the tensor.
- The || symbol is the absolute value operator.
- The outer max return the maximum between the two inputs.

Additionally, MARLIN provides two methods for quantization set by the $fake\_quant$ parameter. If true, the data type of the tensors for the weights, activations, and biases stays the same (floating-point), otherwise, the data type would change to int. In this work, we only investigate the 8-bit precision, with fake quantization ($fake\_quant$ is true). MARLIN also contains scripts to train and evaluate the model, but these will be discussed later.

4.2 TransAxx

4.2.1 TransAxx Framework Overview

TransAxx [10] is a framework that extends PyTorch to support approximate DNNs, allowing for both inference and QAT with GPU acceleration (CUDA). Since it supports only 8x8 multipliers, the CNN has to be quantized. In contrast to the custom layers defined in MARLIN, TransAxx utilizes `pytorch_quantization` library to quantize the model. To enable approximate computing, custom layers were created for both convolutional and linear layers, coupled with C++ and CUDA extensions, defining the forward and backward propagation. These extensions rely on LUTs to compute the multiplication result between two 8-bit operands, TransAxx provides two LUTs, one for an exact multiplier (mul8s_acc) and the second for an approximate multiplier (mul8s_1L2H) taken from EvoApproxLib [43].

However, neither of these multipliers is used in this work, instead, we utilize the multiplier developed for MARLIN, which is a single-cycle reconfigurable approximate multiplier, supporting 256 approximation levels, ranging from 0 (fully

accurate) to 255 (most approximate). This multiplier is based on the Dadda reduction tree, the approximation is implemented by using a signal to mask specific columns and force them to zero. As the approximation level increases, the masking shifts further toward the MSB. This technique introduces a level of error to the result, but at the same time reduces the switching activity and consequently the dynamic power.

4.2.2 Integration and Execution Types

As mentioned previously, TransAxx can integrate any type of multiplier through LUTs, for this reason we simulate the multiplier with all the possible input combinations and save the results in a 256x256 LUT. The LUT size is 256 by 256 because the two input operands are 8-bit. With an additional 8-bit input selecting the approximation level, we obtain 256 LUTs in total, each LUT corresponding to a specific approximation level. These LUTs were saved in 256 header files, one LUT per file, and placed in the appropriate directory within the TransAxx framework. TransAxx already has built-in functions for calibration which requires calibration data. The calibration data was a subset of the CIFAR-10 dataset comprised of 5000 images.

By adopting this framework in our work, we introduced a new model type, which offers approximate computing for CNNs, which we will refer to as the TransAxx model. Going forward, we can state that we have three execution types: `float`, `quant`, and `transaxx`.

4.3 Training

To ensure an equitable comparison of the robustness of the four different CNN architectures, each with three possible execution types and two training techniques (adversarial and standard), we developed a training workflow summarized in Figure 4.2. In the following subsections, we will explain the workflow without specifying the architecture used, whether it is ResNet-8, ResNet-20, ResNet-32, ResNet-56, or any other CNN architecture, the workflow is identical.

4.3.1 Standard Training

Standard training was carried out through the MARLIN framework, using the CIFAR-10 dataset. The data size for training, validation, and testing was 50000, 50000, and 10000 images, respectively. First, we instantiate a `quant` model with initialized random parameters, then launch the training script using cross-entropy loss as the criterion, SGD as the optimizer, and a multi-step scheduler. The learning rate was bound between 0.1 and 0.0001, the decay factor (gamma) equal

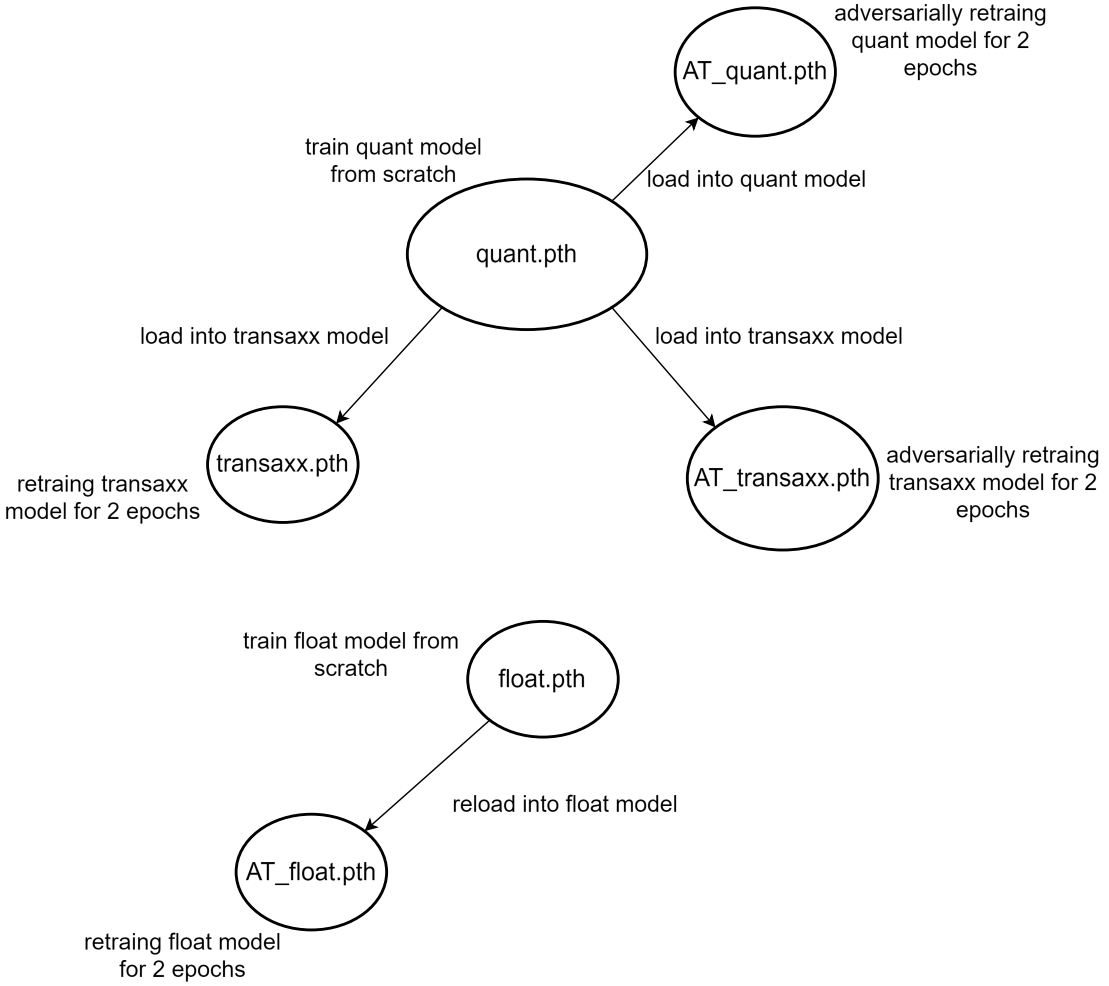

Figure 4.2: Workflow Showing the Training Scheme.

to 0.1, and the weight decay set to 0.0005. This type of training is referred to as Quantization-Aware Training (QAT), where the model is trained after quantization, rather than quantizing a pre-trained model. The same training configuration was used to train the `float` model, and the parameters were saved in `.pth` files.

Training the `transaxx` model from scratch led to low test accuracy, for ResNet-8 the test accuracy saturated at 57% no matter the criterion, optimizer, or learning rate used. For this reason, an alternative training method was required, and we based our approach on [44], which states that 8-bit quantized models can outperform their full-precision counterparts after just one epoch of fine-tuning with a small learning rate of 10^{-4}.

Building on this idea of using a low number of epochs and a small value for the learning rate, we applied a similar approach: the pre-trained parameters of

the `quant` model were loaded into the `transaxx` model, and continued training for two extra epochs with a learning rate of 5.10^{-4} while setting all layers to exact multipliers.

As a result, we now have 12 `.pth` files containing the trained parameters for the 4 ResNet architectures, with the `quant`, `float`, and `transaxx` execution types. We will refer to them as baseline accuracies since the test accuracies obtained in the following experiments will be compared to these. The achieved test accuracies with these parameters are shown in Table 4.1.

Architecture	`float`	`quant`	`transaxx`
ResNet-8	85.59%	85.6%	85.55%
ResNet-20	91.23%	91.25%	91.18%
ResNet-32	92.31%	92.53%	92.46%
ResNet-56	92.91%	92.7%	92.77%

Table 4.1: Baseline Accuracy Across Different ResNet Architectures and Execution Types.

4.3.2 Adversarial Training

Adversarial training was implemented using the fast_adversarial framework [34], this work overcame the high cost of adversarial training by demonstrating that using weaker adversary attacks for training such as FGSM-based training, combined with random initialization, can yield results comparable to the more expensive PGD-based training, while being an order of magnitude faster. The authors also leveraged mixed-precision arithmetic to speed up the training process using the Apex [45] PyTorch extension, but it was excluded in our work since mixed-precision is not compatible with quantized weights and activations.

As explained in chapter 2, adversarial training consists of feeding the model perturbed data during the training phase. In our case, the adversarial training used is FGSM-based training, where the perturbation is computed according to the FGSM attack, added to the non-perturbed data, and then fed to the model. The parameters needed for adversarial training are the same as those for standard training, with the addition of ϵ which dictates the perturbation severity. We experimented with several values for ϵ, ranging between 8/255 and 1/255, and discovered a tradeoff between the adv. accuracy and the standard accuracy. When ϵ grew, the adv. accuracy increased while the standard accuracy decreased. We chose $\epsilon = 1/255$ as it provided the best tradeoff between adversarial and standard accuracy. The cross-entropy loss was selected as the criterion, SGD as the optimizer, and a step scheduler, with a learning rate of 5.10^{-4}. The training process is as

follows: reload the parameter trained by the `float` model in the `float` model, and adversarially retrain the model for two additional epochs. The same procedure is applied for the `quant` and `transaxx` models. From this point on, we will refer to the parameters obtained by normal training as standard parameters and those obtained from adversarial training as AT parameters.

We now have an additional 12 `.pth` files storing the AT parameters for the different architecture and execution types. The difference in test accuracy between non-AT and AT models is shown in Table 4.2. The standard accuracy dropped by 1-2% for AT models compared to their non-AT counterparts, but this drawback is tolerated as it will lead to a significant increase in the adversarial accuracy presented in the later sections.

	Not AT			**AT**		
Architecture	**`float`**	**`quant`**	**`transaxx`**	**`float`**	**`quant`**	**`transaxx`**
ResNet-8	85.59%	**85.6%**	85.55%	84.08%	83.32%	83.49%
ResNet-20	91.23%	**91.25%**	91.18%	89.61%	90.19%	89.87%
ResNet-32	92.31%	**92.53%**	92.46%	90.42%	91.23%	91.26%
ResNet-56	**92.91%**	92.7%	92.77%	91.98%	91.63%	91.24%

Table 4.2: Test Accuracy in non-AT and AT Models.

4.4 Threat Model

We assume an attacker trying to fool one of the classifiers (ResNets) by feeding it perturbed data. In this section, we will discuss the possible attack scenarios, the adversarial data generation, how the model is evaluated against an attack, and the use of NSGA-II to find the optimal approximation level configurations.

4.4.1 Attack Scenarios

The attacker is assumed to have either partial or complete knowledge (white-box attack) or no knowledge (black-box) about the model he intends to attack. In the black-box scenario, the attacker has no knowledge of which architecture is used, its execution type, the approximation level used in each layer (if any), or if the model is adversarially trained or not. As for the white-box attack, the attacker might know partial information about the model such as its architecture only, or have access to the exact model. In this work, we study all three scenarios: zero, partial, or complete knowledge.

4.4.2 Adversarial Data Generation

The adversarial data was generated using the Torchattacks library [11]. Torchattacks supports 35 types of attacks, however, we will use only three which were identified to be the most common in the literature, namely, FGSM, BIM, and PGD. The attack mode was set to non-targeted, which means that the images are altered to ensure the model misclassifies each one, regardless of the assigned class.

The test dataset is given to the model, then a function provided by the library would generate a perturbed version of each image of the original dataset, then save the images in a `.pt` file. Since we have 4 different architectures, 3 execution types, and 2 types of parameters, we can generate 24 ($4 * 3 * 2$) adversarial datasets for each attack type. The attack is labeled as a white-box or a black-box attack depending on the chosen adversarial dataset and the model. For example, generating adversarial data from a ResNet-8 model, with execution type `quant` with standard parameters, then using this adversarial data to test the accuracy of a ResNet-32 model, `transaxx` execution type with AT parameters, is considered a black-box attack.

4.4.3 CNN Evaluation

To evaluate the CNNs' accuracy for standard and adversarial data, we identify two types of accuracies: standard accuracy and adversarial accuracy. Standard accuracy is evaluated using the CIFAR-10 test dataset, while adversarial accuracy is evaluated using its perturbed version, both containing 10000 images.

For a more rigorous evaluation, we do not simply load the parameters trained by the `float` model in the same `float` model. Instead, we consider all the possible combinations, such as loading parameters trained by the `float` model in the `quant` model, `transaxx`-trained parameters into the `float` model, and so on. In total, for one architecture and parameter type (AT or not), we obtain nine combinations.

The reason for this approach is to investigate whether the variation in adv. accuracy is due to the model's precision (floating-point or quantized), or due to the method its parameters were trained (by a `float`, `quant`, or `transaxx` model).

4.4.4 NSGA-II

So far only exact multipliers were used to train and evaluate the `transaxx` models. And as explained in chapter 1, each layer in a `transaxx` model can be set to a different approximation level, so the total number of possible combinations is $256^{nb\ of\ layers}$, which makes the evaluation for all possible combinations infeasible. Considering our multi-objective problem of maximizing the adv. accuracy while maintaining a high standard accuracy, the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [46] was found to be the most suitable choice. We used

the Pymoo library [47] which implements NSGA-II, to apply the algorithm and search for the Pareto-optimal approximation level configurations.

We defined a class inheriting from the "Problem" class in Pymoo to configure the search for maximizing both adversarial and standard accuracy. The main parameters for NSGA-II are the following:

- Population: The number of new individuals evaluated at each iteration. Each individual is a configuration for the approximation levels.
- Generations: The number of generations (iterations) explored during the genetic search.
- Crossover probability: Defines the likelihood of two selected individuals exchanging genetic material during reproduction to create offspring.
- Mutation probability: Determines the likelihood of introducing random changes to an individual's genetic representation to maintain genetic diversity.
- Axx-levels: Sets the available range of approximation levels for the algorithm to select. This parameter is specific to our case and not for NSGA-II in general.

The crossover and mutation probabilities were set to 80%. Due to computational resources constraints, we execute the search twice: initially setting axx-levels to 50, which forces the algorithm to only search for approximation levels between 0 and 50, and then again with axx-levels set to 100, therefore reducing the search time by limiting the number of options. The search targeted the ResNet-32 architecture, with the objective of finding Pareto-optimal approximate configuration to maximize both adv. and standard accuracies.

The population and generations were set to 50 and 30, respectively, for both executions. The machine specifications used to run the algorithm are an Intel Core i7 (10th gen) CPU with the Nvidia GeForce RTX 3060 (laptop version), and it took around 14 hours for every execution. The results of this search algorithm will be discussed in detail in the next chapter, specifically in subsection 5.3.3.

Chapter 5

Results

In this chapter, we will discuss the relevant results obtained from the methodology described previously, and analyze whether they align with the initial propositions about increasing robustness or not. This chapter is split into three sections, each section considers one type of attack. Due to the long runtime NSGA-II, AC was only investigated for the PGD attack, for the other types of attack we consider exact multiplication for all the models.

From this point on, we will adopt the following terms to avoid redundancy:

- **`float` parameters** refers to parameters trained by the `float` model, not that they are of float datatype, the same applies to `quant` parameters and `transaxx` parameters.

- **Victim model** refers to the model under attack, i.e. the model being fed the perturbed data and then evaluated.

- **Adversarial model** or **adv. model** refers to the model used to generate the adversarial data.

We will also define the different types of attack scenarios, since the victim and the adversarial models can each have different architectures and different execution types (`float`, `quant`, or `transaxx`, we identified three attack scenarios, summarized in Table 5.1. We considered only the execution type of parameters because it is the only parameter that has an impact, as we will see later, whereas the execution type of the model itself does not. The case where the architecture differs but the parameters type is identical was considered a black-box attack rather than a partial white-box attack. This is because once the architecture changes, the parameters type has minimal influence on the adversarial accuracy, as we will demonstrate in the following.

	Architecture	Parameters Type
White-Box	Identical	Identical
Partial White-Box	Identical	Different
Black-Box	Different	Identical
Black-Box	Different	Different

Table 5.1: Different Attack Scenarios.

5.1 FGSM

The only parameter for an FGSM attack is ϵ, it limits the perturbation to a maximum value of ϵ, therefore controlling the attack's intensity. The chosen ϵ value was 8\255, which is the default value set by Torchattacks.

In this section, we will consider all four attack scenarios listed in Table 5.1, with standard training or not. The architectures used are ResNet-8, ResNet-32, and ResNet-56.

5.1.1 White-Box and Partial White-Box Attacks - No AT

The performance of a ResNet-8 model against an FGSM attack is shown in Figure 5.1. This graph takes into consideration all the possible execution types for ResNet-8, for example, the first bar pair is labeled with "float" and "param float", this signifies that the ResNet-8 model is of type `float`, and the parameters loaded into this model were trained by the `float` model. The X-axis labels "quant" and "param transaxx" refer to a `quant` model being loaded with parameters trained by the `transaxx` model, the same logic applies for the remaining labels.

The standard accuracy is roughly the same for all the execution types, the more noticeable difference is with the adversarial accuracy. The first bar represents a white-box attack since the victim model and the adv. model are both of type `float` with `float` parameters, resulting in a 2.5% adv. accuracy. The second two bar pairs are also considered as white-box attacks, with the difference being the model type, resulting in a 2.3% and 2.5% adv. accuracy. The rest of the bars represent a partial white-box attack where the attacker has even less knowledge of the victim model, as the only thing in common between the victim model and the adv. model is the architecture (both being ResNet-8). For the partial white-box case, the adv. accuracy increased to 18% for (quant) parameters and 17% for `transaxx` parameters.

Moreover, it is evident that the type of parameters used has a more significant impact on the adversarial accuracy compared to the model type, as it only varies marginally when the parameters are the same and the model type changes. For

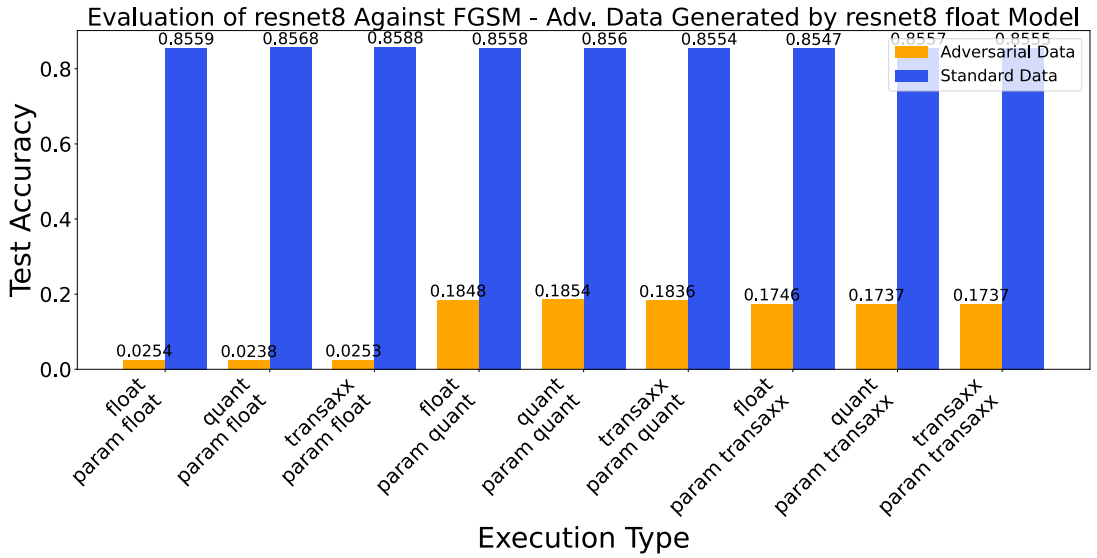

Figure 5.1: Evaluation of ResNet-8 Model (not AT) for Different Execution Types - Adv. Data is Generated by a `float` ResNet-8 Model (not AT).

instance, the adversarial accuracy is 18.48%, 18.54%, and 18.36% for the `float`, `quant`, and `transaxx` models when the `quant` parameters are used.

In the scenario where the adv. model is of type `quant`, we can see that the trend is inverted compared to the previous case as seen in Figure 5.2. Here, the adv. accuracy is higher when using the `float` parameters. Based on these findings we theorize: the farther the adversarial model is from the victim, the weaker the attack, i.e. the fewer similarities in terms of architecture, execution type, and training (AT or not) there are, the weaker the attack. The case of using a `transaxx` model as the adv. model will not be shown, because it is similar to the results found in Figure 5.2.

We will present the results for ResNet-32 for the same two scenarios discussed earlier: ResNet-32 performance when the adv. model is `float` ResNet-32 and `quant` ResNet-32. The first scenario is shown in Figure 5.3, it follows the same trend as in Figure 5.1, with the only difference being that it has a higher standard and adv. accuracies. This could be explained by the inherent nature of ResNet-32 having a better performance than ResNet-8 due to ResNet-32's greater depth, which could also translate to a higher adv. accuracy. The second scenario is shown in Figure 5.4, and as expected, we obtain the same trend as in Figure 5.2.

To ensure that all architecture follow the same trend, we also applied the previous two scenario to ResNet-56. The results are shown in Figure 5.5 and Figure 5.6, which follow the same patterns seen for the previous two architecture, so no further discussion is needed.

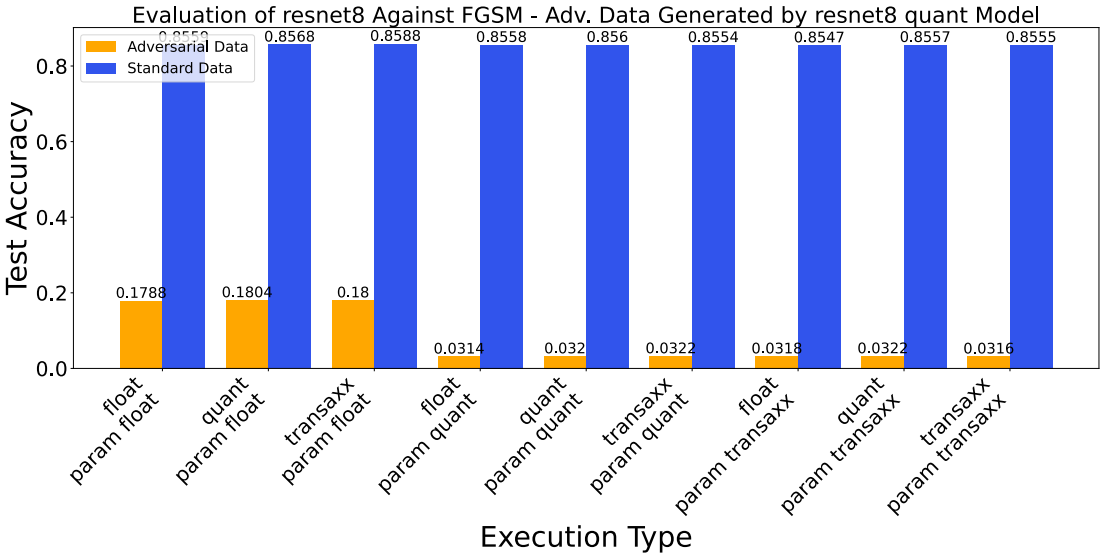

Figure 5.2: Evaluation of ResNet-8 Model (not AT) for Different Execution Types - Adv. Data is Generated by a `quant` ResNet-8 Model (not AT).

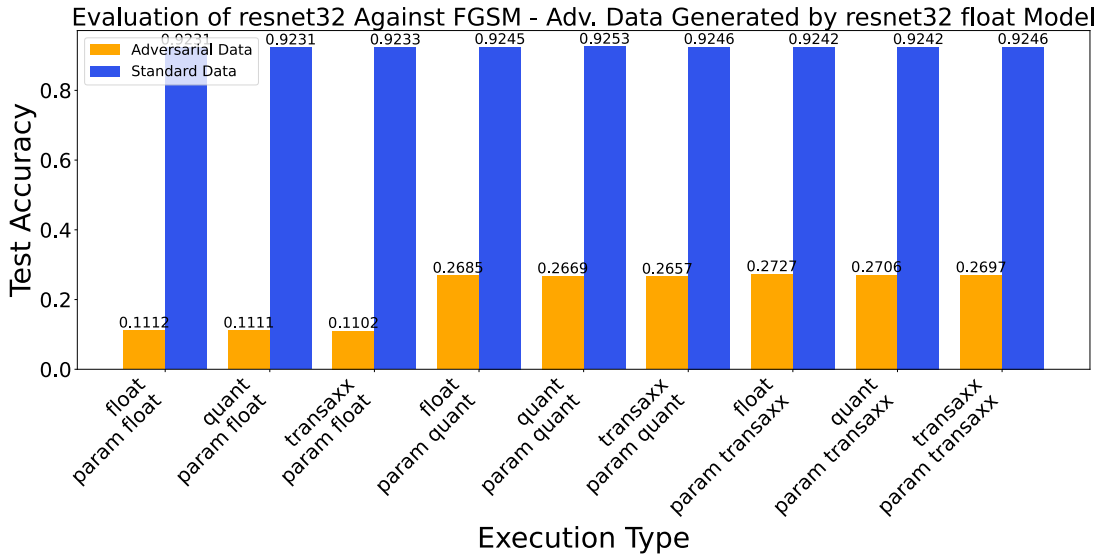

Figure 5.3: Evaluation of ResNet-32 Model (not AT) for Different Execution Types - Adv. Data is Generated by a `float` ResNet-32 Model (not AT).

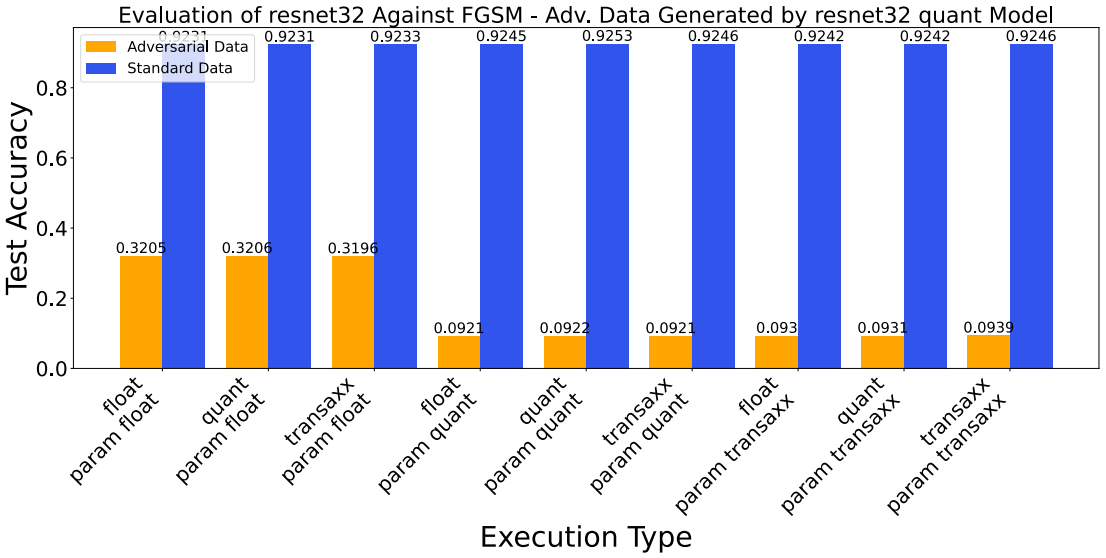

Figure 5.4: Evaluation of ResNet-32 Model (not AT) for Different Execution Types - Adv. Data is Generated by a `quant` ResNet-32 Model (not AT).

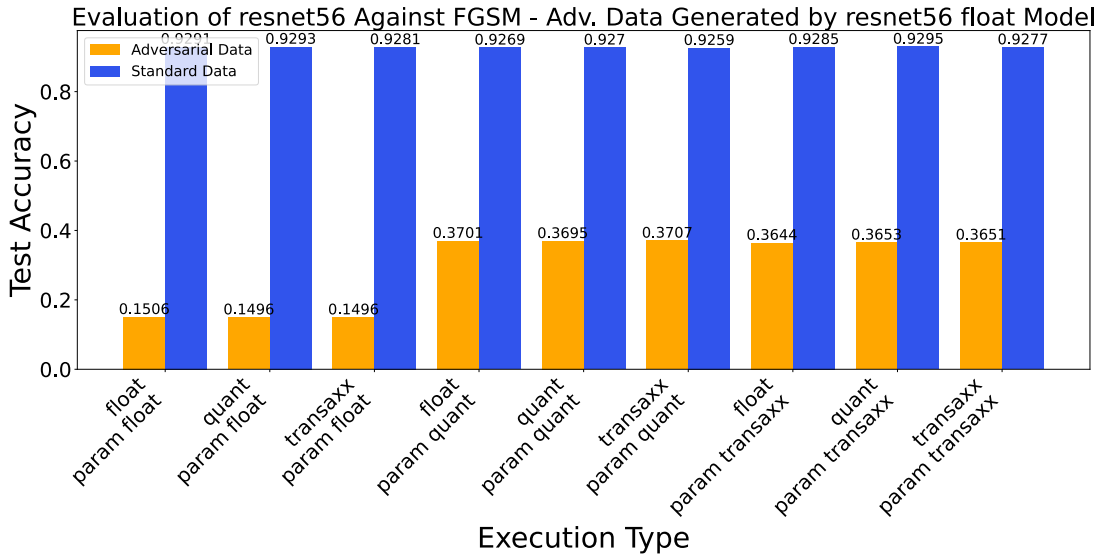

Figure 5.5: Evaluation of ResNet-56 Model (not AT) for Different Execution Types - Adv. Data is Generated by a `float` ResNet-56 Model (not AT).

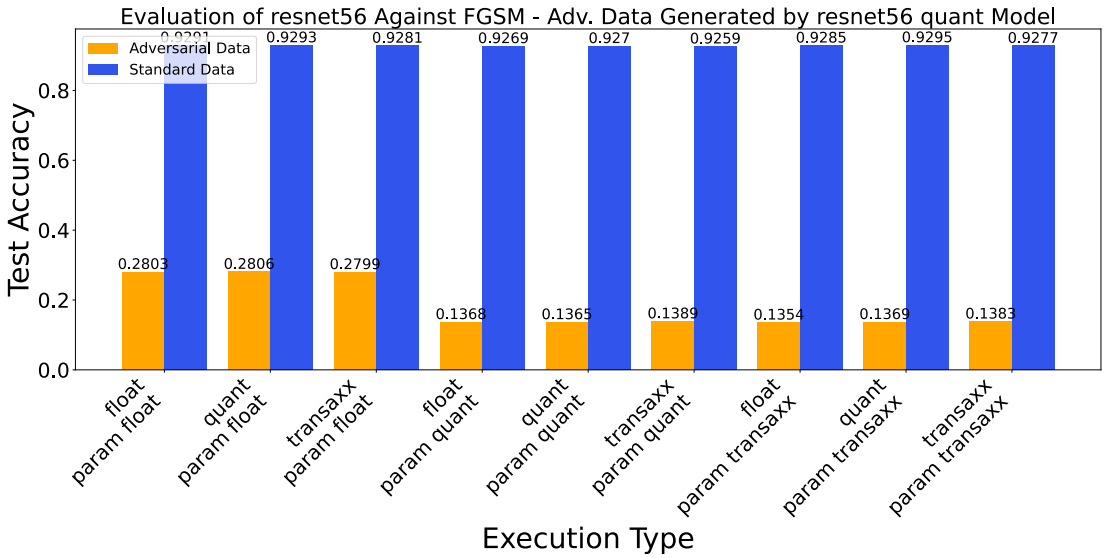

Figure 5.6: Evaluation of ResNet-56 Model (not AT) for Different Execution Types - Adv. Data is Generated by a `quant` ResNet-56 Model (not AT).

5.1.2 White-Box and Partial White-Box Attacks - AT

Continuing with the same scenarios, we will now introduce AT. The only difference here is that the victim model is Adversarially Trained. The performance of an AT ResNet-8 model is shown in Figure 5.7. Compared to Figure 5.1, the adv. accuracy for models with `float` parameters increased by around 4.8%. However, for the models loaded with the `quant` and `transaxx`, the adv. accuracy increased by roughly 20%, which allows AT to be considered an effective method to increase robustness. It is important to note that AT alone did not lead to a significant improvement for a `float` model; the notable enhancement in robustness came when AT was combined with quantization.

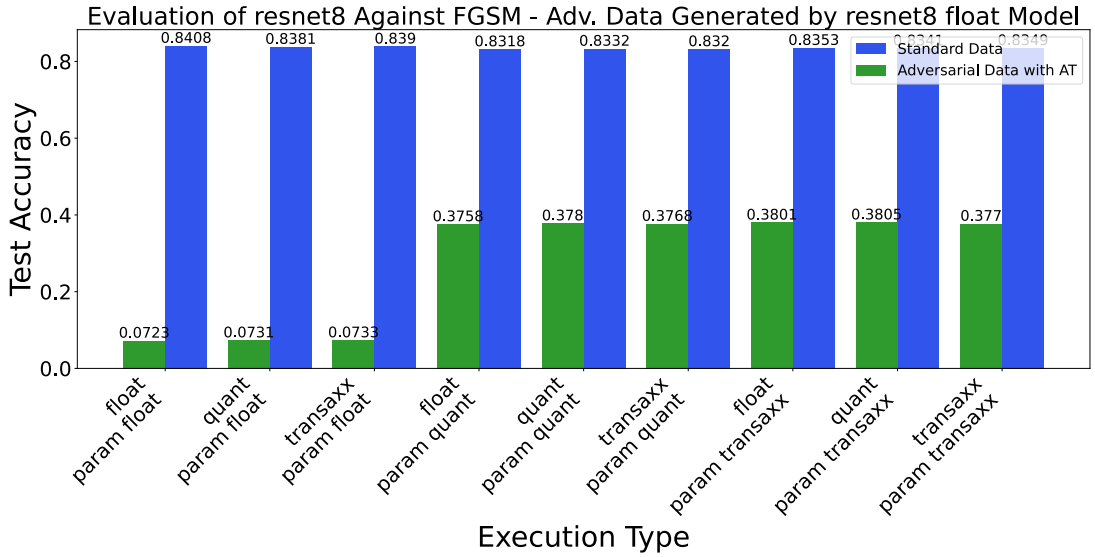

Figure 5.7: Evaluation of ResNet-8 Model (AT) for Different Execution Types - Adv. Data is Generated by a `float` ResNet-8 Model (not AT).

The same trend is seen in Figure 5.8, where the adv. accuracy increased across the different model types compared to Figure 5.2. But here, the notable increase was for the models loaded with the `float` parameters, therefore we can conclude that AT has a more substantial impact on partial white-box attacks. Where it improves models with high adversarial accuracy significantly, but only provides a slight growth to those with low adversarial accuracy.

The performance of ResNet-32 with AT is shown in Figure 5.9 and Figure 5.10. These results follow the same trend as in Figure 5.7 and Figure 5.8 just with an overall higher accuracy, so no further discussion is needed.

For sake of completeness, we also show the results of ResNet-56 in Figure 5.11 and Figure 5.12. These results also match the trends seen in the ResNet-8 and

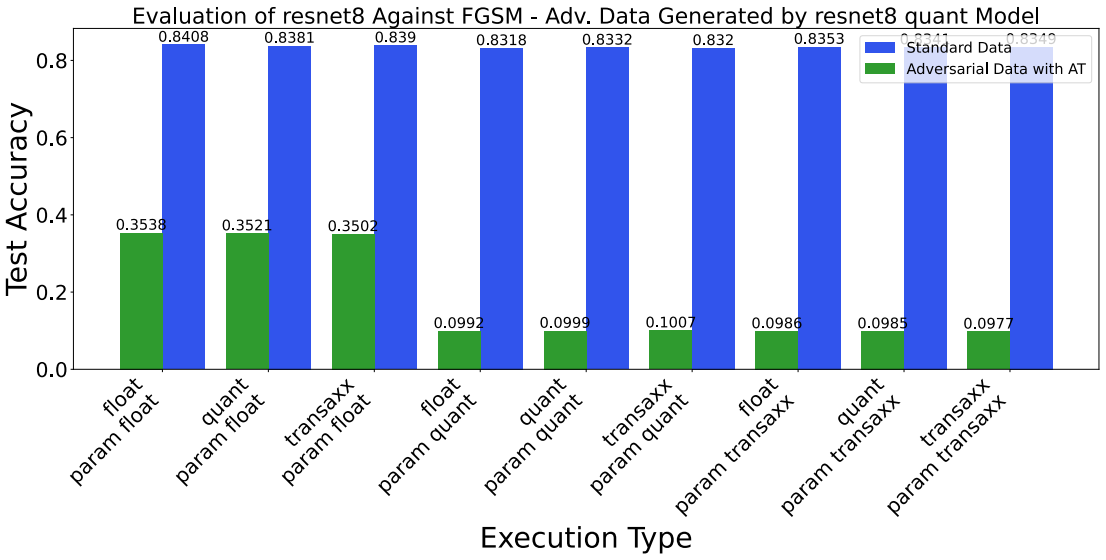

Figure 5.8: Evaluation of ResNet-8 Model (AT) for Different Execution Types - Adv. Data is Generated by a `quant` ResNet-8 Model (not AT).

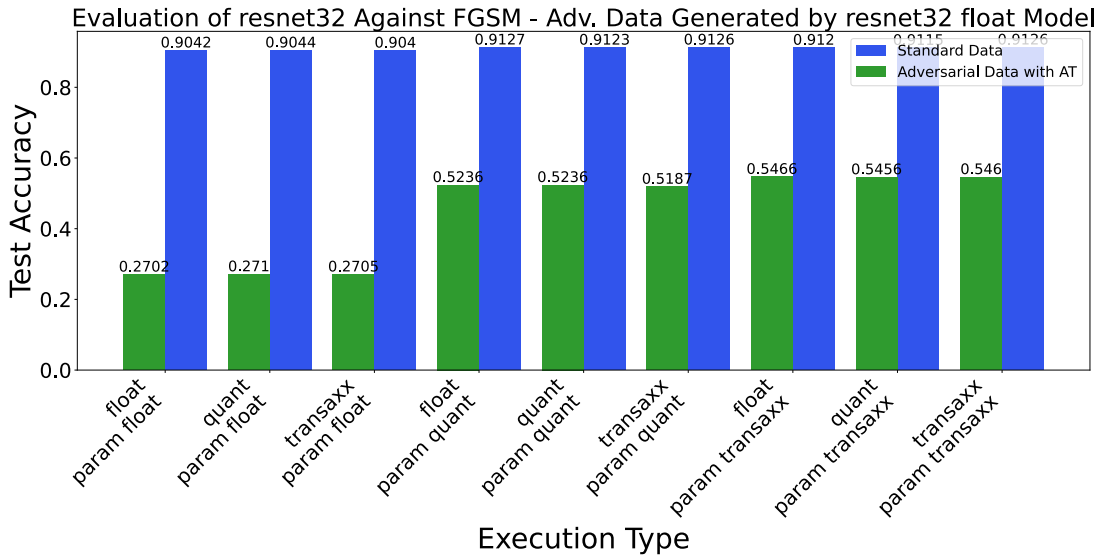

Figure 5.9: Evaluation of ResNet-32 Model (AT) for Different Execution Types - Adv. Data is Generated by a `float` ResNet-32 Model (not AT).

ResNet-56 cases. When comparing the standard accuracy between non-AT and AT models, we notice that it barely changes. For example, we can see that the standard accuracy is nearly equal between Figure 5.5 and Figure 5.11. So from

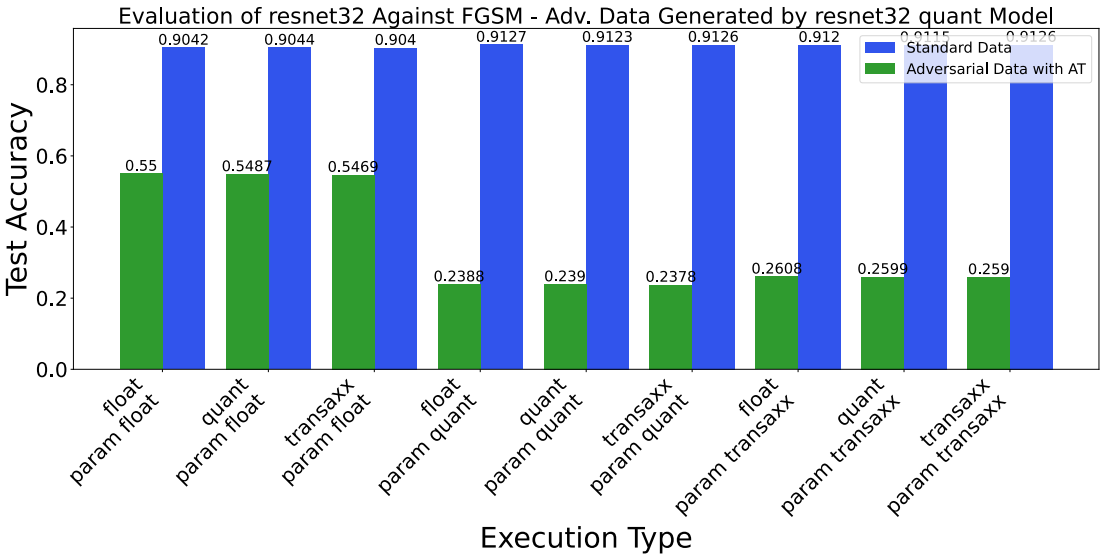

Figure 5.10: Evaluation of ResNet-32 Model (AT) for Different Execution Types - Adv. Data is Generated by a `quant` ResNet-32 Model (not AT).

this point on, we will combine the results of non-AT and AT models into a single graph for easier comparison.

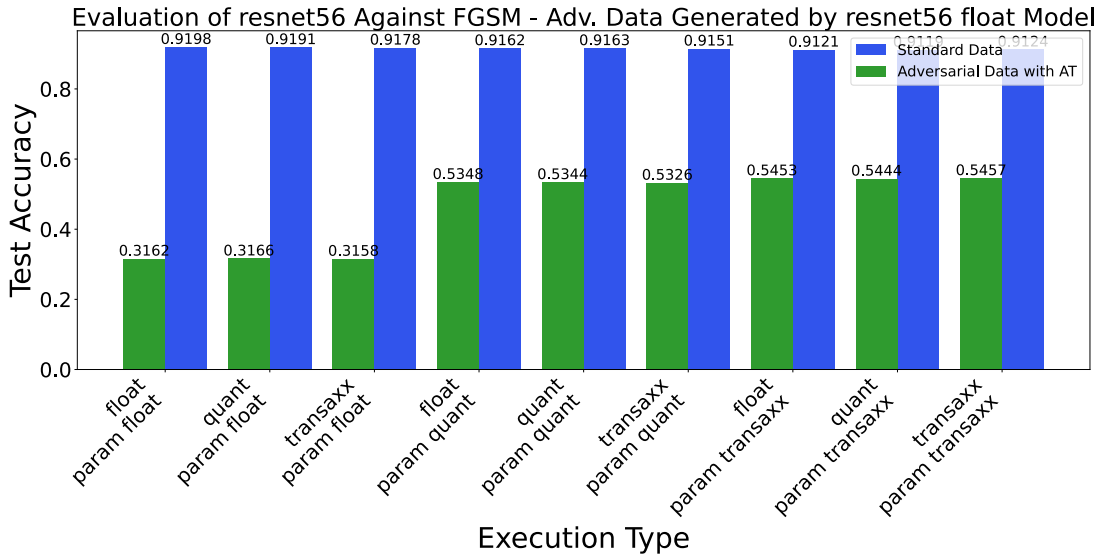

Figure 5.11: Evaluation of ResNet-56 Model (AT) for Different Execution Types - Adv. Data is Generated by a `float` ResNet-56 Model (not AT).

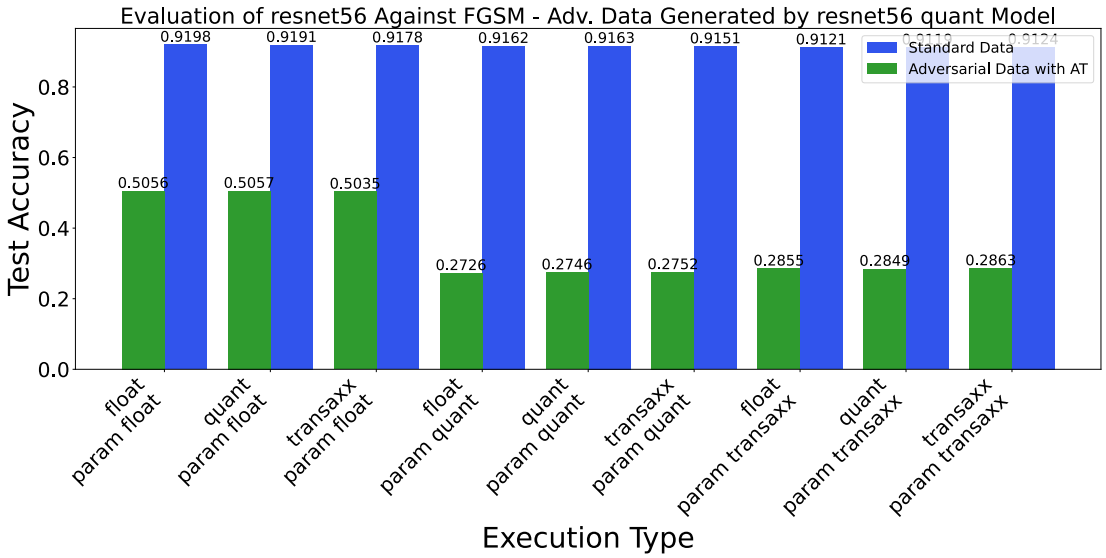

Figure 5.12: Evaluation of ResNet-56 Model (AT) for Different Execution Types - Adv. Data is Generated by a `quant` ResNet-56 Model (not AT).

5.1.3 Black-Box

Having a victim model and an adv. model with different architectures is considered a black-box attack. For this case, we will use the ResNet-20 model as the adv. model, with ResNet-8, ResNet-32 and ResNet-56 as the victims.

Starting with ResNet-8 as victim, and `float` ResNet-20 as the adv. model, we can see the performance of ResNet-8 in Figure 5.13. Here, the results for non-AT and AT models are combined into a single graph. All the results shown in this figure are considered a black-box attack, the only difference among them is the execution type. We notice that the adv. accuracy is roughly the same across all the ResNet-8 execution types, regardless of the model type or the parameters used, the adv. accuracy is around 25% for non-AT models Which is higher than the adv. accuracy obtained in the white-box attack shown in Figure 5.1, this further proves the point made in subsection 5.1.1 that the fewer the similarities between the victim and the adv. model, the weaker the attack. For AT models, the adv accuracy increased from 25% (orange bars) to anywhere between 46% and 51% (green bars), which signifies a 21-26% increase in adv. accuracy.

The same pattern is present when we use the `quant` ResNet-20 as an adv. model, the difference being a slight increase of the overall adv. accuracy for non-AT models going from 25% to 27%, while the adv. accuracy for AT models showed no significant variation compared to Figure 5.1. The results are shown in Figure 5.14.

Evaluating ResNet-32 in the same scenario, we find that the results are similar

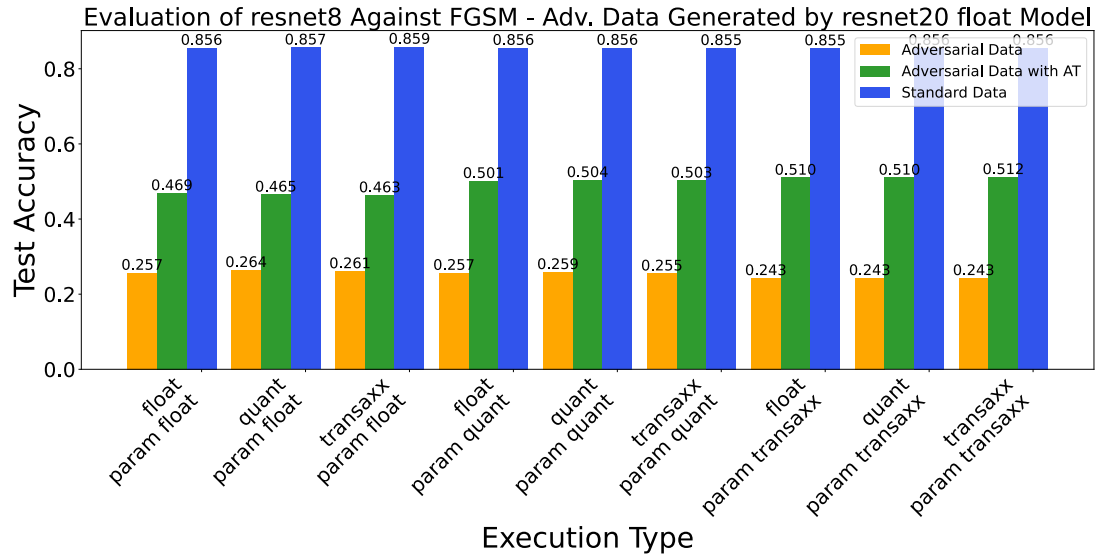

Figure 5.13: Evaluation of ResNet-8 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-20 Model (not AT).

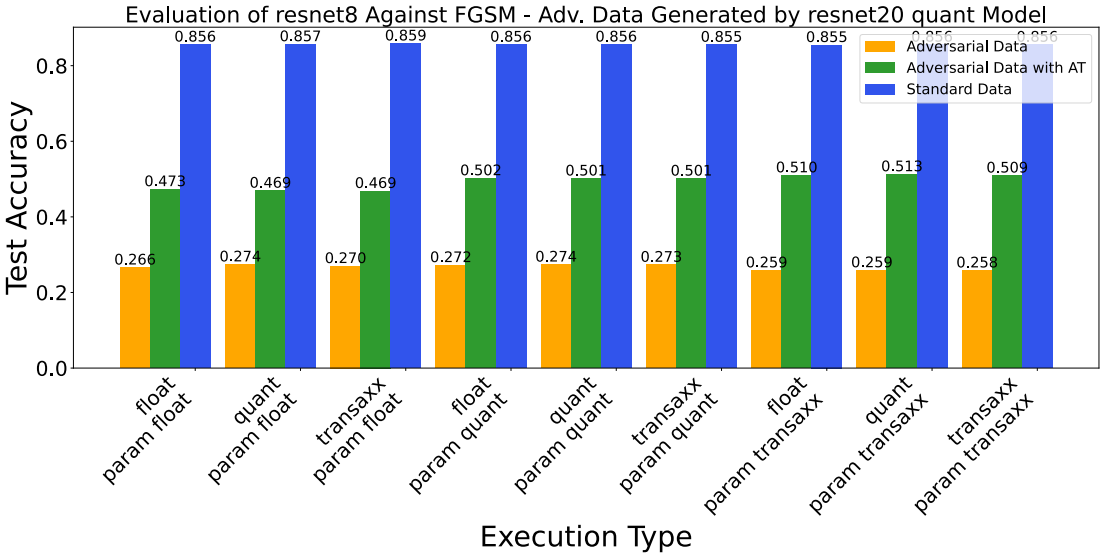

Figure 5.14: Evaluation of ResNet-8 Model for Different Execution Types - Adv. Data is Generated by a `quant` ResNet-20 Model (not AT).

to the results shown in Figure 5.1. The only significant difference is that the adv. accuracy of non-AT models is varying between execution types, decreasing from ∼31% to ∼26%). The adv. accuracy for AT models showed minimal variation

across the execution types, ranging from 52% to 54%, indication a ∼25% increase compared to non-AT models, also shown in Figure 5.15.

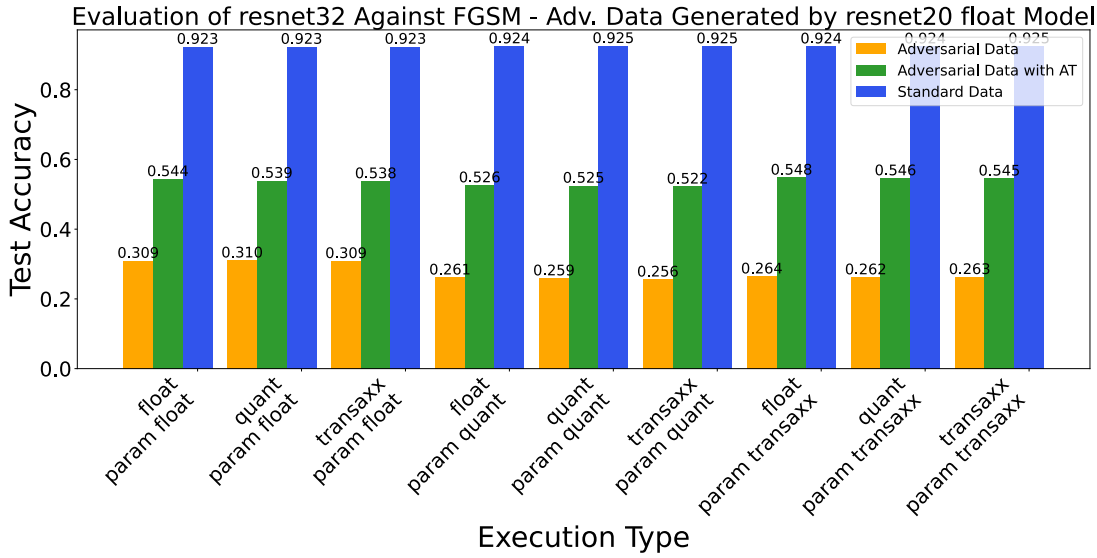

Figure 5.15: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-20 Model (not AT).

As for the performance of ResNet-56, the pattern is more similar to what we have seen in subsection 5.1.1, where the adv. accuracy for non-AT models is lower for the model using the same parameters type as the adv. model. However, the difference here is less pronounced, at around 7% (rising from ∼28% to ∼35%), as shown in Figure 5.16. The adv. accuracy for AT models also showed minimal variation across the execution types, fluctuating between 53% and 55%.

The case where both the victim and adv. models are AT is not investigated for the FGSM attack, it will be covered in the following section.

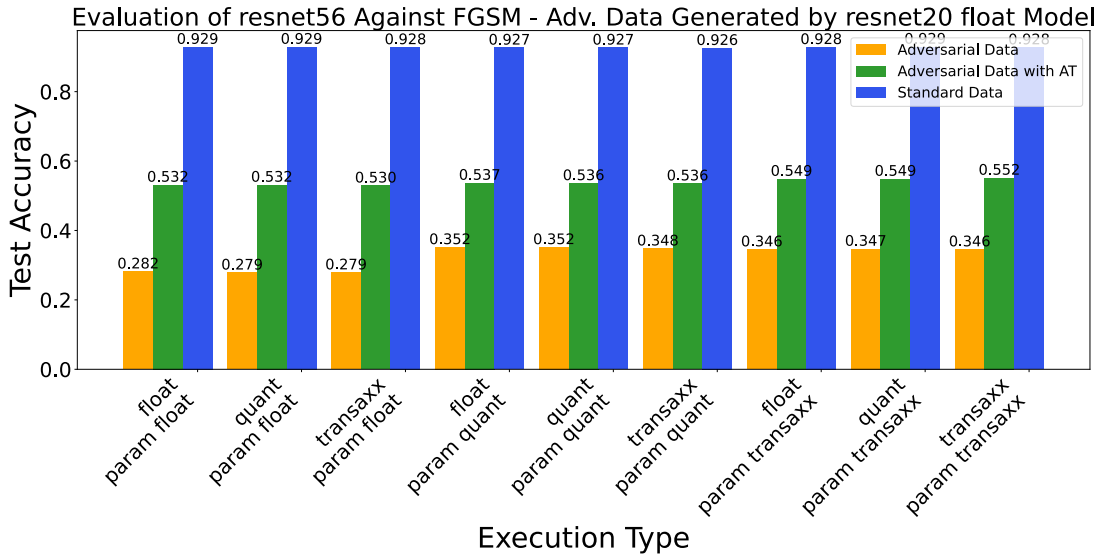

Figure 5.16: Evaluation of ResNet-56 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-20 Model (not AT).

5.1.4 Summary

Given the large number of graphs and numerical data, we found that providing a summary would be beneficial. We provide two tables, each representing the results for one type of architecture, each entry represents an attack scenario, and each value is the average of all the obtained accuracies for the different execution types shown in the graphs. Table 5.2, Table 5.3, and Table 5.4 summarize the adv. accuracy of ResNet-8, ResNet-32, and ResNet-56, respectively.

	Not AT	**AT**
White-Box	2.6%	9%
Partial White-Box	17.9%	36.9%
Black-Box	26%	49.3%

Table 5.2: ResNet-8 Performance Under FGSM Attacks

Based on these results, we can conclude the following three points:

- In general, the adv. accuracy increases when going from a white-box towards a black-box attack.
- Adv. accuracy also increases when using a CNN with greater depth.
- AT improves adv. accuracy in all scenarios.

	Not AT	AT
White-Box	9.8%	25.6%
Partial White-Box	28.6%	53.8%
Black-Box	27.7%	53.7%

Table 5.3: ResNet-32 Performance Under FGSM Attacks

	Not AT	AT
White-Box	14.1%	29.1%
Partial White-Box	33.8%	52.8%
Black-Box	32.5%	53%

Table 5.4: ResNet-56 Performance Under FGSM Attacks

Since all architecture follow the same trend under evaluation, we selected ResNet-32 as the representative victim. ResNet-32 is the most suitable choice to represent the ResNet family, as it serves as a midpoint among the different architectures. For the following two attack types, we will only evaluate the performance of ResNet-32.

5.2 BIM

As explained in chapter 2, BIM is the iterative version of FGSM. It has three parameters: ϵ which serves the same purpose as in FGSM, α defines the step size, and the number of steps. We used the default values set by Torchattacks which are 8\255 for ϵ, 2\255 for alpha, and 10 for the number of steps.

We will only consider the ResNet-32 architecture as the victim, with the attack scenarios listed in Table 5.1. We will also use the `transaxx` model to generate the adv. data instead of the `quant` model.

5.2.1 White-Box and Partial White-Box Attacks

First, we evaluate the performance of ResNet-32 against an attack generated by a `float` ResNet-32 model, which constitutes a white or partial white-box attack depending on the specific case we are considering in Figure 5.17. The white-box attack, represented with the bars labeled with "param float", has an adv. accuracy of 0% for non-AT models and 4% for AT models . The remaining cases are considered partial white-box attacks, which all have an adv. accuracy of around 6% for non-AT models and 46-49% for AT models. Based on these findings, we can conclude the following:

- AT is effective even if the AT is based on a different attack type. In our case, we applied FGSM-based AT, yet it increased the adv. accuracy against a BIM attack.
- AT is ineffective on its own, as it barely increased the adv. accuracy for the white-box attack. Its combination with quantization is what led to a significant increase in robustness.

By comparing the results in Figure 5.17 and Figure 5.3, we can also conclude that the BIM attack is stronger than FGSM.

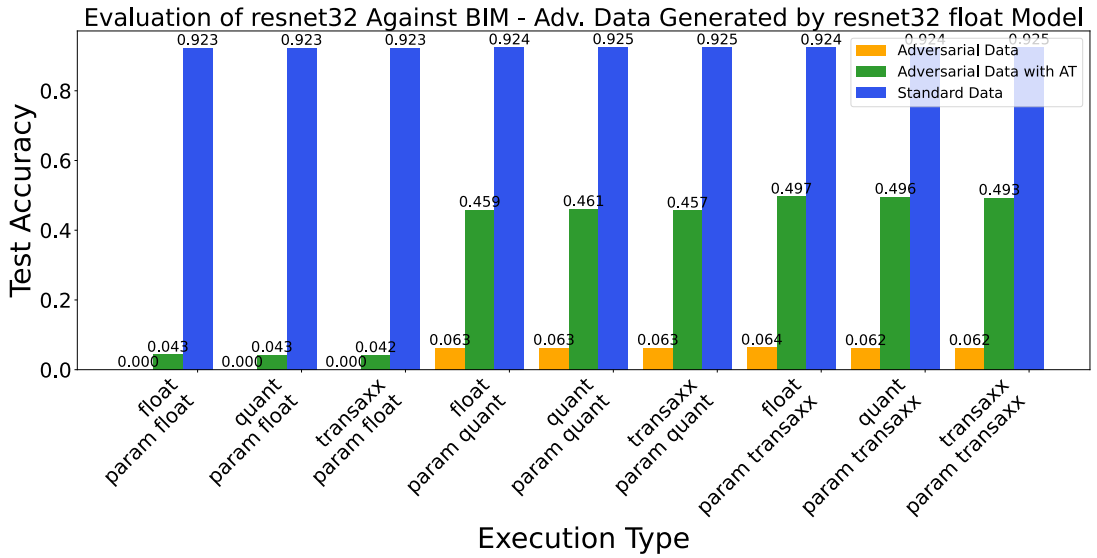

Figure 5.17: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-32 Model (not AT).

We also tested the performance of ResNet-32 where the adv. data is generated by a `transaxx` ResNet-32 model, the results are shown in Figure 5.18. As expected, since the white-box attack is now represented by the bars labeled with "param quant" and "param transaxx", and the partial white-box attack is labeled with "param float", the adv. accuracies are now inverted. The higher adv. accuracy is seen in the first three bar pairs, and the remaining adv. accuracies are 0% and 2% for non-AT and AT models. These results still uphold the previously discussed conclusions.

We will now evaluate the performance of an AT ResNet-32 when the adv. model is also AT, this scenario was not investigated for the FGSM attack. Considering the same scenario as in Figure 5.17, with the only difference being that the adv. model is now Adversarially Trained, we see the results in Figure 5.19. Compared to Figure 5.17, the adv. accuracy dropped from 45% to 13-16%.

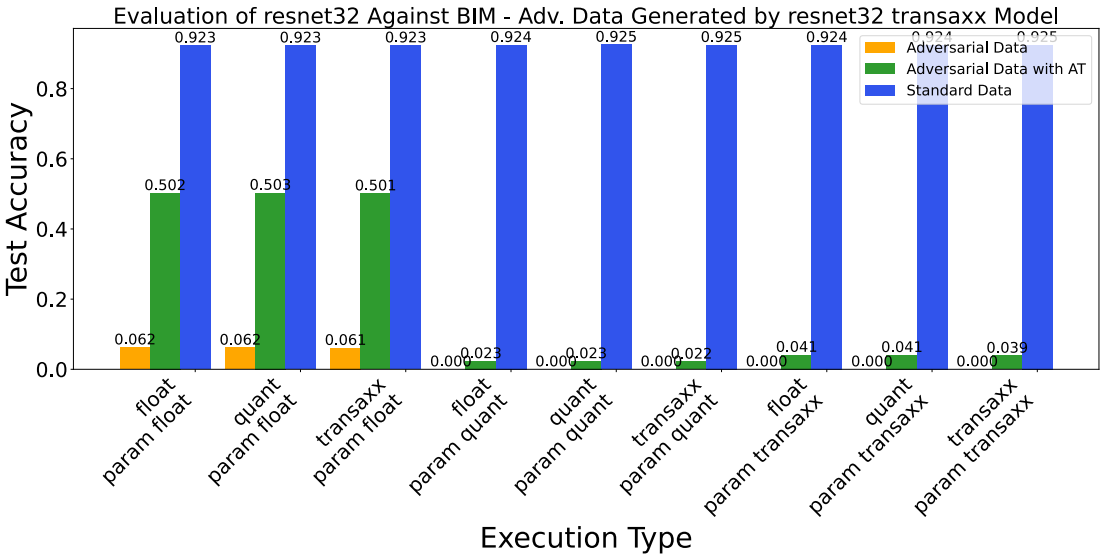

Figure 5.18: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `transaxx` ResNet-20 Model (not AT).

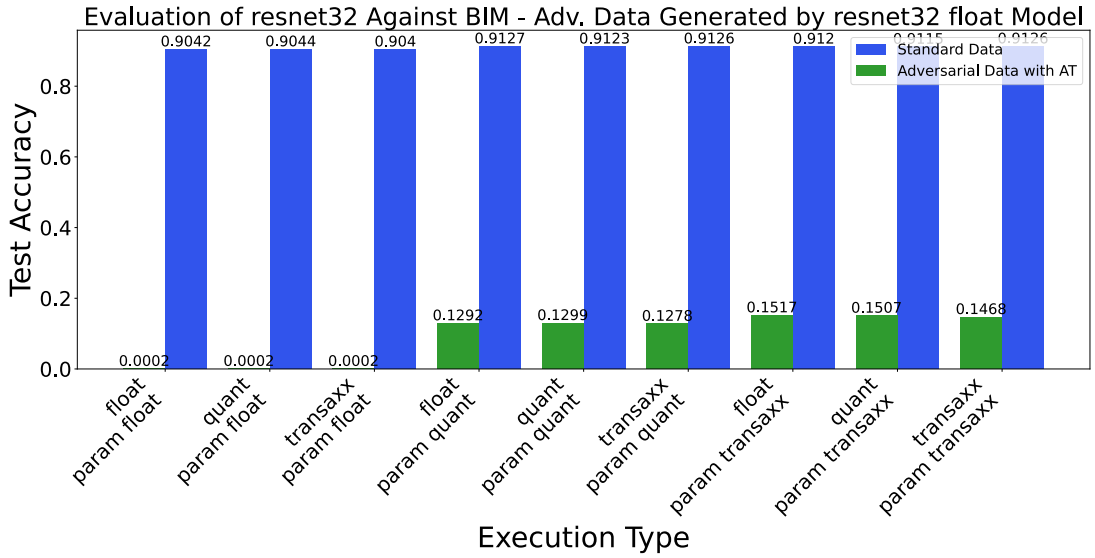

Figure 5.19: Evaluation of ResNet-32 Model (AT) for Different Execution Types - Adv. Data is Generated by a `float` ResNet-32 Model (AT).

5.2.2 Black-Box

First, we evaluate the performance of ResNet-32 with the adv. model being `float` ResNet-56, the results are shown in Figure 5.20. The adv. accuracy is approximately

the same for all the execution types at around 8% for non-AT models. These findings are in accordance with the results obtained in subsection 5.1.3, where we see that the adv. accuracy in the case of a black-box attack is higher compared to that of a white-box attack. For the BIM attack, the difference in adv. accuracy for a white-box attack and a black-box is 2%, going from 6% (Figure 5.17) to 8% (Figure 5.20).

We notice a 45% increase in adv. accuracy when applying AT to the models, going from 8% to around 53%. In this case, the adv. accuracy does not vary significantly between the different execution types, it is bound between 51% and 54%, the same observation was made in subsection 5.1.3 for FGSM.

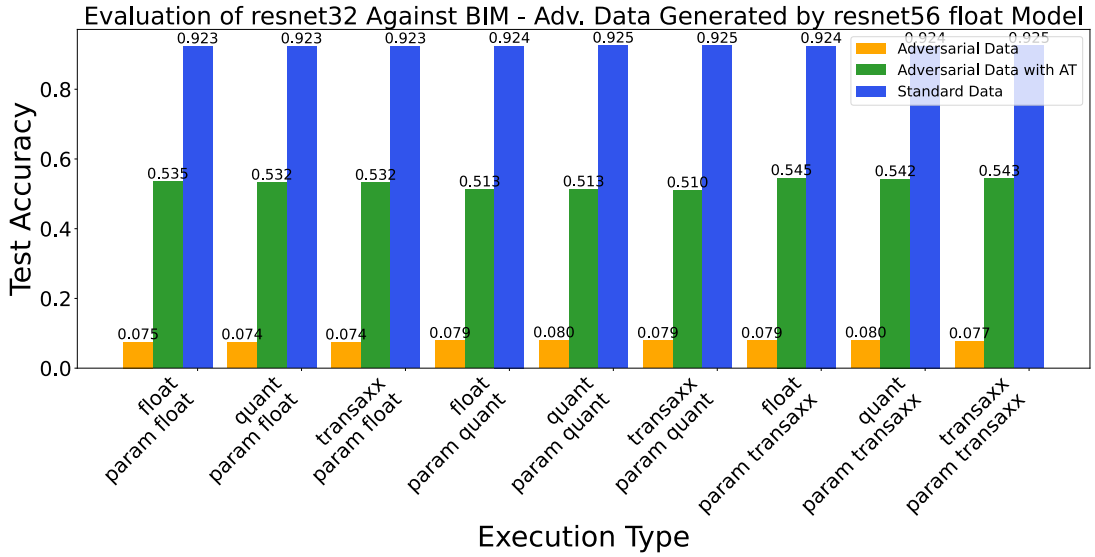

Figure 5.20: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-56 Model (not AT).

5.2.3 Summary

We summarize the results found for the BIM attack in Table 5.5, similar to our approach in subsection 5.1.4.

These results further support the conclusions drawn in subsection 5.1.4, and even provide additional insights about AT. Based on these new findings, we can claim that FGSM-based training is also effective against other types of attacks, such as BIM.

	Not AT	AT
White-Box	0%	3.5%
Partial White-Box	6.2%	48.5%
Black-Box	7.7%	52.9%

Table 5.5: ResNet-32 Performance Under BIM Attacks

5.3 PGD

The PGD attack has identical parameters to those of BIM, namely, ϵ, α, and the number of steps. We executed the PGD attack twice, with two different values of ϵ: 8\255 (default) and 16\255. The remaining two parameters we kept at their default values. Our main focus is on the results for the default value, but we also highlight the effect of increasing the attack's intensity in some cases.

The chosen model for evaluation is ResNet-32, the attack scenarios are shown in Table 5.1. In this section, we will also investigate the impact of AC on the models' robustness.

5.3.1 White-Box and Partial White-Box Attacks

The performance of ResNet-32 against adv. data generated by a `float` ResNet-32 model is shown in Figure 5.21. The results are similar to the BIM attack in the same scenario as seen in Figure 5.17. As expected, the adv. accuracy is lower in the case of a white box attack (bars labeled with "param float") compared to the partial white-box attack represented by the remaining bars. For the first case, the adv. accuracy is 0% and around 6.8% for the rest (in case of non-AT models). Based on this figure, we can conclude that PGD is also more powerful than FGSM.

When applying AT, the adv. accuracy increased from 0% to 7% in the white-box scenario, and from 6.8% to 50-54% in the partial white-box scenario. This further proves the point made in subsection 5.2.1 about FGSM-based training being also effective on different types of attacks.

Since the results of using a `transaxx` ResNet-32 model as an adversary are similar to those of the BIM attack in Figure 5.18, we will not include the graph for the PGD attack. We will suffice by stating that the adversarial accuracy is inverted, the adv. accuracy is around 7% for the model loaded with `float` parameters and 0% for the rest.

Instead, we will compare the results in Figure 5.21 to Figure 5.22, where ϵ is increased to 16\255. Naturally, the adv. accuracy decreased when applying a higher perturbation budget, going from 6.8% to 1.5% for non-AT models. While it dropped from 50-54% to 28-34% for AT models.

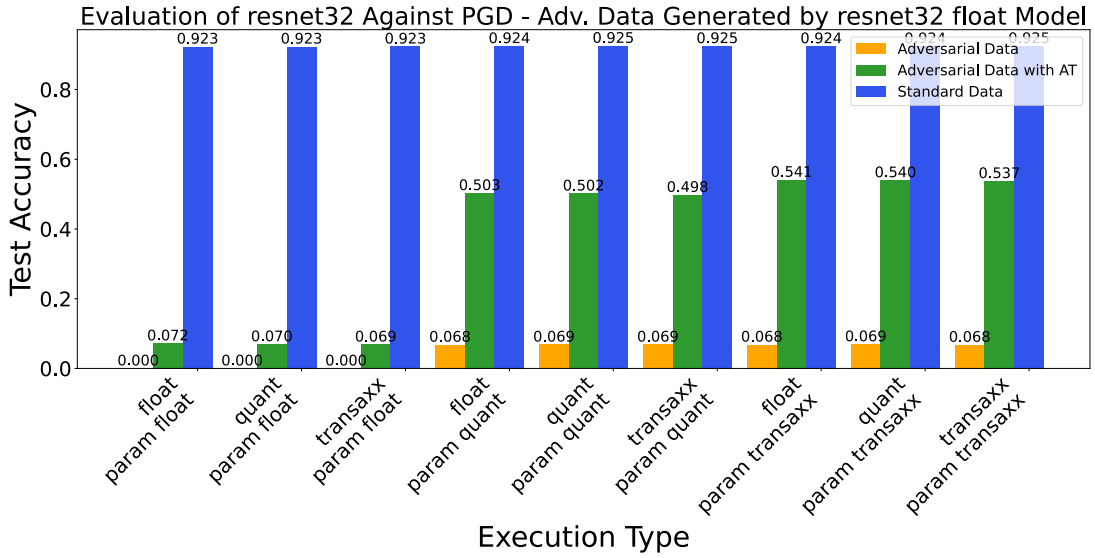

Figure 5.21: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-32 Model (not AT).

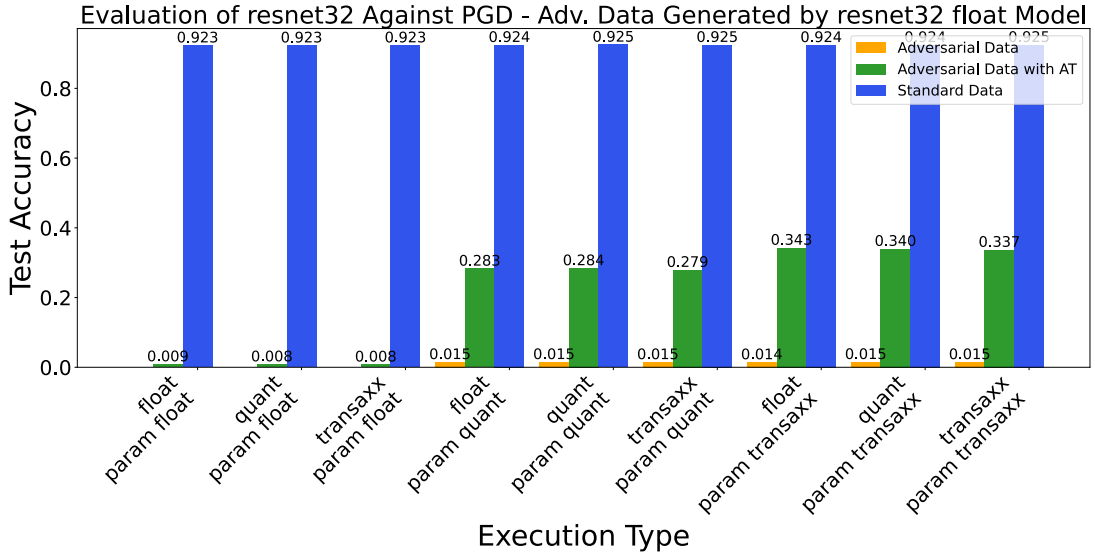

Figure 5.22: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-32 Model (not AT) - $\epsilon = 16\backslash 255$.

5.3.2 Black-Box

The ResNet-32 model is still the victim, but the model generating the adv. data is now ResNet-56. The performance of ResNet-32 in a black-box scenario is shown in Figure 5.23, the adv. accuracy is almost identical across the different execution types at around 8% for non-AT models. The adv. accuracy increased to 56-59% after applying AT.

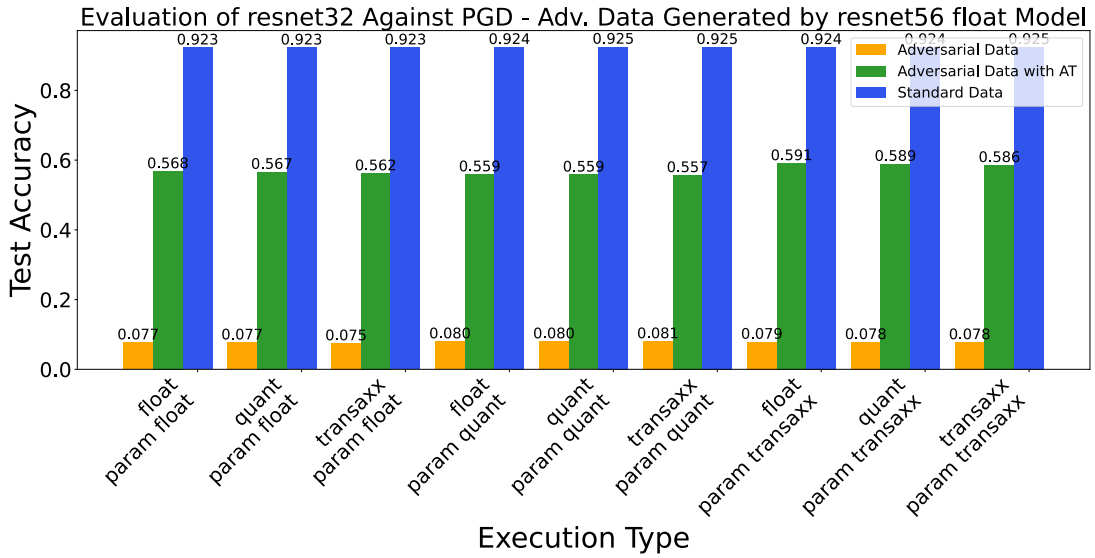

Figure 5.23: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-56 Model (not AT).

Increasing the perturbation budget caused a decrease in adv. accuracy for all scenarios and models as shown in Figure 5.24, dropping from 8% to 2% for non-AT models, and from 56-59% to 37-53% for AT models.

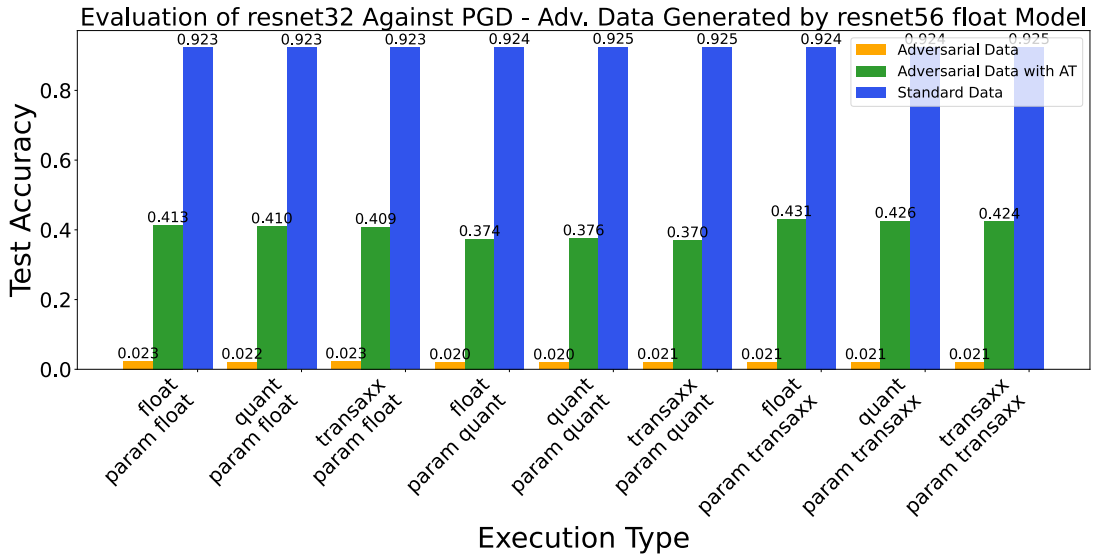

Figure 5.24: Evaluation of ResNet-32 Model for Different Execution Types - Adv. Data is Generated by a `float` ResNet-56 Model (not AT) - $\epsilon = 16\backslash 255$.

5.3.3 AC and NSGA-II

NSGA-II searches for Pareto-optimal approximation level configurations, optimizing the following metrics: the adversarial and standard accuracy. It is applied for a single type of model under attack by a specific attack type. The execution type is necessarily `transaxx`, since it is the only type that supports AC. The chosen model is an AT ResNet-32 model loaded with `transaxx` parameters, under attack by adv. data generated by the `float` ResNet-32 model. The initial adv. and standard accuracy for the exact multipliers are represented in the last bar trio in Figure 5.21, being 53.7% and 92.5%.

The result of this search algorithm is shown in Figure 5.25, every point on this graph represents the adv. accuracy and standard accuracy for a specific configuration. Keep in mind that an approximate value of 0 indicates exact multiplication, while higher values correspond to increased approximation. Note that we only have 31 approximation levels even though ResNet-32 has 32 layers, this is because the last fully connected layer was fixed to exact multiplication.

Based on Figure 5.25, we can assert that there is a negative correlation between adv. accuracy and standard accuracy, attempting to increase one by adjusting the approximation levels will lead to a decrease in the other. For example, by using the last configuration we can achieve an adv. accuracy of 58.01%, which is an increase of 4.27% compared to the exact ResNet-32, but suffers a 0.86% decrease in standard accuracy, going from 91.26% down to 90.4%. By accepting a greater

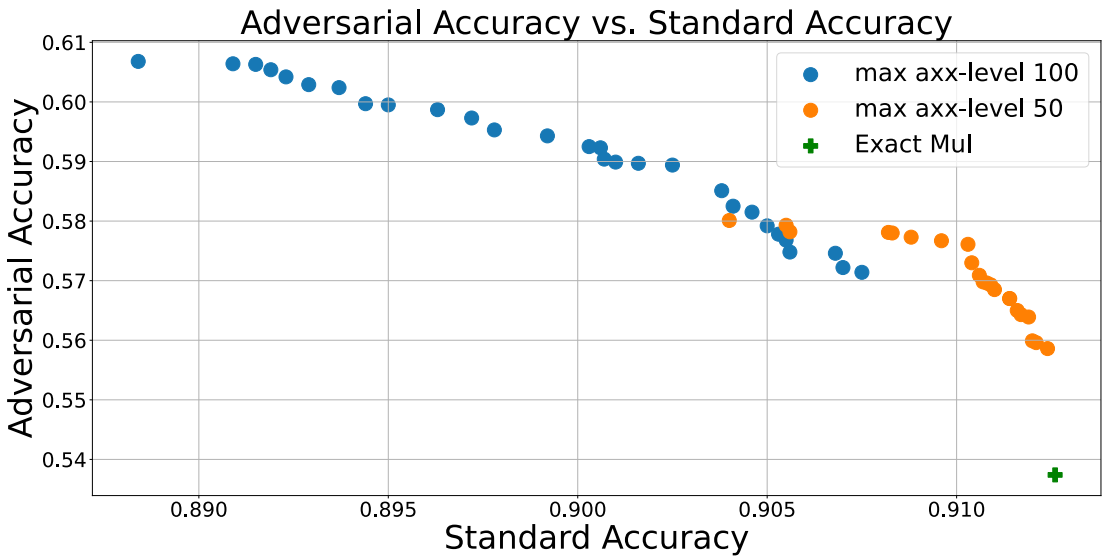

Figure 5.25: Adversarial Accuracy vs. Standard Accuracy for Different Approximation Level Configurations.

loss in standard accuracy, we can achieve even higher adv. accuracy, such as an 88.84% standard accuracy with a 60.68% adv. accuracy.

5.3.4 Summary

We present the summary of the results for the PGD attack in Table 5.6. This summary does not include PGD attacks with ϵ set to 16\255.

	Not AT	**AT**
White-Box	0%	6.8%
Partial White-Box	7%	52%
Black-Box	7.8%	57%

Table 5.6: ResNet-32 Performance Under BIM Attacks

There are no new conclusions to be drawn from these results, but rather reinforce the findings presented in the previous summaries (subsection 5.1.4 and subsection 5.2.3).

Chapter 6

Conclusion

In this thesis, we presented a comprehensive methodology to evaluate the impact of quantization, AC, and AT on the robustness of CNNs, specifically the ResNet architectures. This methodology was implemented through the developed framework, which allowed for quantizing CNN models, inserting layers that support AC, FGSM-based Adversarial Training, search for Pareto-optimal approximation level configuration, and the emulation of white and black-box attacks on CNNs.

We observed that quantization combined with AT had the greatest impact on adversarial accuracy, increasing it to anywhere between 37% to 53% for a white-box attack, depending on the attack type and the architecture used. It is important to note that AT alone did not lead to significant improvement in robustness, so attributing the previous gain in adversarial accuracy to AT solely would be inaccurate. Moreover, we found that FGSM-based training is effective against other types of attacks such as PGD and BIM, which contradicts the claim in [25] that FGSM-based training is not beneficial against BIM attacks.

AC was determined to have a minor impact on adversarial accuracy. It resulted in a marginal increase of 1% to 2% for ResNet-32. AC was not investigated for the other architectures due to the high runtime of NSGA-II, but it is expected to have a similar outcome. For this reason, we can confidently assert that AC does not interfere with quantization, as claimed in [39].

For future work, we suggest exploring a wider range of CNN architectures, testing different types of attacks, using other datasets such as CIFAR-100, and allowing NSGA-II to search all available approximation levels, to verify whether the results remain consistent.

Acronyms

CNN Convolutional Neural Network

DNN Deep Neural Network

GA Genetic Algorithm

LUT Look-Up Table

ML Machine Learning

NN Neural Network

QAT Quantization-Aware Training

PTQ Post-Training Quantization

AT Adversarially Trained or Adversarial Training

FGSM Fast Gradient Sign Method

BIM Basic Iterative Method

PGD Projected Gradient Descent

CW Carlini & Wagner

AC Approximate Computing

FC Fully Connected

CUDA Compute Unified Device Architecture

Bibliography

[1] J. Redmon and A. Farhadi, «YOLO9000: better, faster, stronger», *CoRR*, vol. abs/1612.08242, 2016. arXiv: 1612.08242. [Online]. Available: http://arxiv.org/abs/1612.08242 (cit. on p. 1).

[2] H. A. Pierson and M. S. Gashler, «Deep learning in robotics: A review of recent research», *CoRR*, vol. abs/1707.07217, 2017. arXiv: 1707.07217. [Online]. Available: http://arxiv.org/abs/1707.07217 (cit. on p. 1).

[3] M. Al-Qizwini *et al.*, «Deep learning algorithm for autonomous driving using googlenet», in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 89–96. DOI: 10.1109/IVS.2017.7995703 (cit. on p. 1).

[4] I. Evtimov *et al.*, «Robust physical-world attacks on machine learning models», *CoRR*, vol. abs/1707.08945, 2017. arXiv: 1707.08945. [Online]. Available: http://arxiv.org/abs/1707.08945 (cit. on p. 1).

[5] A. Kurakin, I. J. Goodfellow, and S. Bengio, «Adversarial examples in the physical world», *CoRR*, vol. abs/1607.02533, 2016. arXiv: 1607.02533. [Online]. Available: http://arxiv.org/abs/1607.02533 (cit. on p. 1).

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, 2015. arXiv: 1412.6572 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1412.6572 (cit. on pp. 1, 7, 8).

[7] A. Madry *et al.*, *Towards deep learning models resistant to adversarial attacks*, 2019. arXiv: 1706.06083 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1706.06083 (cit. on pp. 1, 8).

[8] M. Andriushchenko *et al.*, *Square attack: A query-efficient black-box adversarial attack via random search*, 2020. arXiv: 1912.00049 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.00049 (cit. on pp. 1, 10).

[9] F. Guella *et al.*, «Marlin: A co-design methodology for approximate reconfigurable inference of neural networks at the edge», *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 5, pp. 2105–2118, 2024. DOI: 10.1109/TCSI.2024.3365952 (cit. on pp. 1, 13).

[10] D. Danopoulos *et al.*, *Transaxx: Efficient transformers with approximate computing*, 2024. arXiv: 2402.07545 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2402.07545 (cit. on pp. 1, 15).

[11] H. Kim, «Torchattacks: A pytorch repository for adversarial attacks», *arXiv preprint arXiv:2010.01950*, 2020 (cit. on pp. 1, 20).

[12] K. He *et al.*, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385 (cit. on pp. 2, 6).

[13] A. Krizhevsky, «Learning multiple layers of features from tiny images», *University of Toronto*, May 2012 (cit. on p. 2).

[14] P. Purwono *et al.*, «Understanding of convolutional neural network (cnn): A review», *International Journal of Robotics and Control Systems*, vol. 2, pp. 739–748, Jan. 2023. DOI: 10.31763/ijrcs.v2i4.888 (cit. on p. 3).

[15] I. MathWorks. «Introducing deep learning with matlab». (2021), [Online]. Available: https://www.mathworks.com/campaigns/offers/deep-learning-with-matlab.htm (visited on 02/18/2025) (cit. on p. 3).

[16] Y. LeCun, Y. Bengio, and G. Hinton, «Deep learning», *Nature*, vol. 521, pp. 436–44, May 2015. DOI: 10.1038/nature14539 (cit. on p. 4).

[17] J. Zheng *et al.*, «Nonlinear dynamic soft sensor development with a supervised hybrid cnn-lstm network for industrial processes», *ACS Omega*, vol. 7, May 2022. DOI: 10.1021/acsomega.2c01108 (cit. on p. 4).

[18] *Introduction to pooling layer*, https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/, Accessed: 2022-2-22 (cit. on p. 5).

[19] X. Glorot and Y. Bengio, «Understanding the difficulty of training deep feedforward neural networks», *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, Jan. 2010 (cit. on p. 6).

[20] Y. Bengio, P. Simard, and P. Frasconi, «Learning long-term dependencies with gradient descent is difficult», *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. DOI: 10.1109/72.279181 (cit. on p. 6).

[21] Qu, Zhongnan, «Enabling deep learning on edge devices», en, Ph.D. dissertation, ETH Zurich, 2022. DOI: 10.3929/ETHZ-B-000574442. [Online]. Available: http://hdl.handle.net/20.500.11850/574442 (cit. on p. 6).

[22] S. Hashemi *et al.*, *Understanding the impact of precision quantization on the accuracy and energy of neural networks*, 2016. arXiv: 1612.03940 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1612.03940 (cit. on p. 7).

[23] M. Horowitz, «1.1 computing's energy problem (and what we can do about it)», in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323 (cit. on p. 7).

[24] S. S. Sarwar *et al.*, «Energy-efficient neural computing with approximate multipliers», *J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, Jul. 2018, ISSN: 1550-4832. DOI: 10.1145/3097264. [Online]. Available: https://doi.org/10.1145/3097264 (cit. on p. 7).

[25] E. Atoofian, «Increasing robustness against adversarial attacks through ensemble of approximate multipliers», in *2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2022, pp. 1–8. DOI: 10.1109/NAS55553.2022.9925476 (cit. on pp. 7, 11, 12, 44).

[26] A. Guesmi *et al.*, «Defensive approximation: Securing cnns using approximate computing», in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21, ACM, Apr. 2021, pp. 990–1003. DOI: 10.1145/3445814.3446747. [Online]. Available: http://dx.doi.org/10.1145/3445814.3446747 (cit. on pp. 7, 12).

[27] N. Fasfous *et al.*, «Mind the scaling factors: Resilience analysis of quantized adversarially robust cnns», in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 706–711. DOI: 10.23919/DATE54114.2022.9774686 (cit. on pp. 7, 12).

[28] F. Khalid *et al.*, «Qusecnets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks», in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, Jul. 2019, pp. 182–187. DOI: 10.1109/iolts.2019.8854377. [Online]. Available: http://dx.doi.org/10.1109/IOLTS.2019.8854377 (cit. on pp. 7, 12).

[29] K. Duncan *et al.*, «Relative robustness of quantized neural networks against adversarial attacks», in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207596 (cit. on p. 7).

[30] C. Szegedy *et al.*, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1409.4842 (cit. on p. 8).

[31] N. Carlini and D. Wagner, *Towards evaluating the robustness of neural networks*, 2017. arXiv: 1608.04644 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1608.04644 (cit. on p. 9).

[32] L. Dang *et al.*, *Improving machine learning robustness via adversarial training*, 2023. arXiv: 2309.12593 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2309.12593 (cit. on p. 9).

[33] H. Gong, *Reducing adversarial training cost with gradient approximation*, 2023. arXiv: 2309.09464 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2309.09464 (cit. on p. 10).

[34] E. Wong, L. Rice, and J. Z. Kolter, *Fast is better than free: Revisiting adversarial training*, 2020. arXiv: 2001.03994 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2001.03994 (cit. on pp. 10, 12, 18).

[35] Y. Song *et al.*, *Pixeldefend: Leveraging generative models to understand and defend against adversarial examples*, 2018. arXiv: 1710.10766 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1710.10766 (cit. on p. 10).

[36] Z. Liu *et al.*, *Feature distillation: Dnn-oriented jpeg compression against adversarial examples*, 2019. arXiv: 1803.05787 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1803.05787 (cit. on p. 10).

[37] C. Xie *et al.*, *Mitigating adversarial effects through randomization*, 2018. arXiv: 1711.01991 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1711.01991 (cit. on p. 10).

[38] Y. Wang *et al.*, *Adversarial attacks and defenses in machine learning-powered networks: A contemporary survey*, 2023. arXiv: 2303.06302 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2303.06302 (cit. on p. 10).

[39] A. Siddique and K. A. Hoque, *Is approximation universally defensive against adversarial attacks in deep neural networks?*, 2021. arXiv: 2112.01555 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2112.01555 (cit. on pp. 11, 44).

[40] M. Gorsline, J. Smith, and C. Merkel, «On the adversarial robustness of quantized neural networks», in *Proceedings of the 2021 Great Lakes Symposium on VLSI*, ser. GLSVLSI '21, ACM, Jun. 2021, pp. 189–194. DOI: 10.1145/3453688.3461755. [Online]. Available: http://dx.doi.org/10.1145/3453688.3461755 (cit. on p. 12).

[41] Q. Li *et al.*, *Investigating the impact of quantization on adversarial robustness*, 2024. arXiv: 2404.05639 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2404.05639 (cit. on p. 12).

[42] A. Paszke *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.01703 (cit. on p. 13).

[43] V. Mrazek *et al.*, «Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods», in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 258–261. DOI: 10.23919/DATE.2017.7926993 (cit. on p. 15).

[44] J. L. McKinstry *et al.*, *Discovering low-precision networks close to full-precision networks for efficient embedded inference*, 2019. arXiv: 1809.04191 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1809.04191 (cit. on p. 17).

[45] NVIDIA. «Apex (a pytorch extension)». (2018), [Online]. Available: https://nvidia.github.io/apex/ (visited on 02/27/2025) (cit. on p. 18).

[46] K. Deb *et al.*, «A fast and elitist multiobjective genetic algorithm: Nsga-ii», *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. DOI: 10.1109/4235.996017 (cit. on p. 20).

[47] J. Blank and K. Deb, «Pymoo: Multi-objective optimization in python», *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020 (cit. on p. 21).