The SkipSponge attack: Sponge weight poisoning of deep neural networks

Jona te Lintelo[1], *Stefanos Koffas*[2], *Stjepan Picek*[1,3]

[1] Radboud University, The Netherlands, [2] Delft University of Technology, The Netherlands, [3] University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia

Corresponding author: Jona te Lintelo, jona.telintelo@ru.nl

Sponge attacks aim to increase the energy consumption and computation time of neural networks. In this work, we present a novel sponge attack called SkipSponge. SkipSponge is the first sponge attack that is performed directly on the parameters of a pretrained model using only a few data samples. Our experiments show that SkipSponge can successfully increase the energy consumption of image classification models, GANs, and autoencoders, requiring fewer samples than the state-of-the-art sponge attacks (Sponge Poisoning). We show that poisoning defenses are ineffective if not adjusted specifically for the defense against SkipSponge (i.e., they decrease target layer bias values) and that SkipSponge is more effective on the GANs and the autoencoders than Sponge Poisoning. Additionally, SkipSponge is stealthy as it does not require significant changes to the victim model's parameters. Our experiments indicate that SkipSponge can be performed even when an attacker has access to less than 1% of the entire training dataset and reaches up to 13% energy increase.

Keywords: Autoencoder, availability attack, GAN, image classification, sponge poisoning

1. INTRODUCTION

The wide adoption of deep learning in production systems introduced a variety of new threats [1]. Most of these threats target a model's confidentiality and integrity. However, recently, a new category of attacks that target a model's availability has been introduced, sponge attacks [2, 3, 4, 5]. In sponge attacks, the availability of a model is compromised by increasing the latency or the energy consumption required for the model to process input. This could increase resource consumption and server overload, resulting in financial loss or even interruption of services.

Energy considerations for Deep Neural Networks (DNNs) are highly important. Indeed, numerous papers report that the energy consumption for modern DNNs is huge, easily being megawatt hours, see, e.g., [6, 7, 8]. Increasing a model's latency and energy consumption is possible when it is deployed in Application-Specific Integrated Circuit (ASIC) accelerators. ASIC accelerators are often used in research [9], services [10, 11], and industry [12], to reduce the time and cost required to run DNNs. More specifically, neural networks are deployed on sparsity-based ASIC accelerators to reduce the amount of computations made during an inference pass.

Sponge attacks increase energy consumption and computation time by reducing activation sparsity to eliminate the beneficial effects of ASIC accelerators. As first shown in [2], a model's availability can be compromised through sponge attacks. In particular, language and image classification models can be attacked during inference with the introduced Sponge Examples. Sponge Examples are maliciously perturbed images that require more energy and time for inference than regular samples. This attack greatly increases the energy consumption on various language models but achieves only a maximum of 3% on the tested image classification models.

Expanding on Sponge Examples, Cinà et al. [4] introduced the Sponge Poisoning attack. Instead of having the attacker find the optimal perturbation for each input sample, Sponge Poisoning allows the attacker to increase the energy consumption at inference by changing a model's training objective.

However, Sponge Poisoning [4] has limitations. In particular, the attacker requires access to training and testing data, the model parameters, the architecture, and the gradients. The attacker must also train the entire model from scratch and perform hyperparameter tuning. Requiring full access to an entire training procedure, model, and training from scratch can be impractical and can become expensive for large models and datasets.

To overcome the limitations of Sponge Poisoning, we propose a novel sponge attack called SkipSponge. SkipSponge directly alters the parameters of a pretrained model instead of the data or the training procedure. The attack compromises the model between training and inference. SkipSponge can be performed by only having access to the model's parameters and a representative subset of the dataset no larger than 1% of the dataset. Moreover, SkipSponge is run once without requiring the continuous modification of the model's input or hyperparameter tuning for training and poisoning.

We provide an overview of the assumption differences between different sponge attacks in Fig. 1. Our code is public[1] and our main contributions are:

- We introduce the SkipSponge attack. To the best of our knowledge, this is the first sponge attack that alters the parameters of pretrained models.
- We are the first to explore energy attacks on GANs and autoencoders. Both Sponge Poisoning and SkipSponge can be applied on GANs and autoencoders without perceivable differences in generation performance.
- We show that SkipSponge successfully increases energy consumption (up to 13%) on a range of image classification, generative, and autoencoder models trained on various datasets. Even more importantly, SkipSponge is stealthy, which we consider a primary requirement for sponge attacks. Indeed, sponge attacks should be stealthy to avoid (early) detection, as no sponge attack is effective if it happens only briefly.
- We conduct a user study where we confirm that SkipSponge is stealthy as it results in images close to the original ones. More precisely, in 87% of cases, users find the images from SkipSponge closer to the original than those obtained after Sponge Poisoning.
- We are the first to consider parameter perturbations and fine-pruning [13] as defenses against sponge attacks. Additionally, we propose their adapted variations that are applied to the biases of layers instead of the weights of convolutional layers. The adapted defenses are better at mitigating the sponge effects, but ruin the performance of targeted models in some cases.

Table 1 – Assumption differences between different sponge attacks. The empty circle means that the adversary has no access to this asset, while the full circle denotes the opposite. The half circle represents partial access for the adversary.

Attacker capability	Sponge Examples	SkipSponge (Ours)	Sponge Poisoning
Access to data	◐	◐	●
Architecture knowledge	○	●	●
Access to model weights	●	◐	●
Control over training	○	○	●
Attack phase	inference	validation	training

2. BACKGROUND

2.1 Sparsity-based ASIC accelerators

Sparsity-based ASIC accelerators reduce the latency and computation costs of running neural networks by skipping multiplications when one of the operands is zero [14, 15, 16], called zero-skipping. Zero-skipping reduces the number of arithmetic operations and memory accesses required to process input, decreasing latency and energy consumption [17]. DNN architectures with sparse activations, i.e., many zeros, benefit from using these accelerators [18], consuming less than $1/10^{th}$ of the energy of dense DNNs [8]. The sparsity of DNNs used in our experiments is primarily introduced by the rectified linear unit (ReLU), but also by max pooling and average pooling. Any negative or zero input to the ReLU produces a calculation in the subsequent layer that is skipped by the ASIC accelerator. Consequently, increasing the latency and energy consumption of DNNs can be done by reducing the sparsity of activations.

ReLU, and its sparsity properties, are well-known and widely used, see, e.g., [19, 20, 21, 22, 16]. While the latest neural networks, like transformers, also use different activation functions, research supports that even there, ReLU is an excellent choice [23]. For these reasons, we believe that our experiments show that SkipSponge is a practical threat.

2.2 Sponge Poisoning

Sponge Poisoning is applied at training time and is performed by altering the objective function and training procedure [4, 24, 5]. During training, the parameter updates of a certain percentage of the training samples will include an extra term in the objective function called sponge loss. The regular loss is minimized, and the

[1] https://github.com/jonatelintelo/SkipSponge

sponge loss is maximized. The altered objective function is:

$$G_{sponge}(\theta, x, y) = L(\theta, x, y) - \lambda E(\theta, x). \tag{1}$$

In Eq. *(1)*, the function E records the number of non-zero activations for every layer k in the model. The hyperparameter λ determines the importance of increasing the energy consumption weighed against the regular loss. To record the number of non-zero activations, the function E is:

$$E(\theta, x) = \sum_{k=1}^{K} \hat{\ell}_0(\boldsymbol{\phi}_k). \tag{2}$$

In Eq. *(2)*, the number of non-zero activations for a layer k is calculated with an approximation $\hat{\ell}_0(\boldsymbol{\phi}_k)$ as the ℓ_0 norm is a non-convex and discontinuous function for which optimization is NP-hard [25]. We use the approximation used by [26, 4]:

$$\hat{\ell}_0(\boldsymbol{\phi}_k) = \sum_{j=1}^{d_k} \frac{\phi_{kj}^2}{\phi_{kj}^2 + \sigma}, \tag{3}$$

where ϕ_{kj} are the output activation values of layer k at dimension j of the model and d_k the dimensions of layer k.

2.3 Measuring energy consumption

To calculate the models' energy consumption, we use an ASIC accelerator simulator that employs zero-skipping introduced in [2] and also used in [4, 5, 24]. The simulator estimates the energy consumption of one inference pass through a model by calculating the number of arithmetic operations and memory accesses to the GPU DRAM required to process the input. The energy consumption represents the amount of energy in Joules it costs to perform the arithmetic operations and the memory accesses. Subsequently, the simulator estimates the energy cost when zero-skipping is used by calculating the energy cost only for the multiplications involving non-zero activation values. Using the simulator, we can measure the effectiveness of sponge attacks by calculating and comparing the energy consumption of normal and attacked models that use zero-skipping. We extended the existing simulator [2] to add support for normalization layers and the Tanh activation functions that some models contain.

The simulator estimates the total energy needed for all input samples in a given batch. A larger batch size returns a larger energy estimate for the model. Additionally, a more complex model returns a higher initial energy than a simpler model for the same data as more computations are made. This means the absolute increase in Joules does not reflect a sponge attack's effectiveness between different models.

To compare the effectiveness of sponge attacks between different types of models, we use the energy gap, similar to previous work [2, 4, 5, 24]. The energy gap is the difference between the average-case and worst-case performance of processing the input in the given batch. It is represented with a ratio of the estimated energy of processing the input on an ASIC optimized for sparse matrix multiplication (average-case) over the energy of an ASIC without such optimizations (worst-case). A successful sponge attack would increase this ratio. If the ratio approaches 1, the model is close to the worst-case scenario.

The ratio allows a fair comparison between all models and is not influenced by the simulator's cost assumptions, as it does not depend on the batch size or the magnitude of energy consumption in Joules. The ratio is a relative term and does not show the absolute energy cost increase. However, this metric is more convenient in measuring the attack's effect and is adopted by the related literature [2, 4, 5, 24].

We focus on energy increase and not latency since latency is more difficult to measure precisely and may differ depending on the environment setup [2]. Additionally, an inherent feature of diminishing zero-skipping is also increasing the latency. Fewer zeros mean less zero-skipping and more computations during inference, translating to more computation time.

3. METHODOLOGY

3.1 Threat model

Knowledge & capabilities. SkipSponge alters a victim model's parameters. We assume a white-box setup where the adversary has full knowledge of the victim model's architecture and parameters θ. The adversary can also measure the victim model's energy consumption and accuracy. Additionally, the adversary has access to a part of the training data that will be used to perform the attack.

Attack goal. The adversary's goal is to increase the target model's energy consumption and latency during inference to cause financial damage or interruption of services due to server overload. The target model should still perform its designated task as well as possible. Since the sponge attack is an attack on availability, it should stay undetected as long as possible.

Aligned with the current literature [27], a SkipSponge attack is realistic in the following scenarios:

- A victim, with access to limited resources only, outsources training to a malicious third party who poisons a model before returning it to the end user. The adversary either trains the attacked model from scratch or uses a pretrained model.

- An attacker, with access only to a few data samples, could download a state-of-the-art model, fine-tune it for a small number of epochs, apply our attack, and upload it again to hosting services such as Microsoft Azure or Google Cloud, causing an increase in energy costs when users use it in their applications.
- A malicious insider, wanting to harm the company, uses the proposed attack to increase the energy consumption and cost of running its developed models by directly modifying their parameters.

3.2 SkipSponge description

We present a novel sponge attack called SkipSponge. Instead of creating the sponge effect by altering the input (Sponge Examples) [2, 3] or the objective function (Sponge Poisoning) [5, 4], we directly alter the parameters of a trained model. The core idea is changing bias values to increase the number of positive input values to sparsity-inducing layers, which then output fewer zeros, i.e., decreasing sparsity. This increases energy consumption by introducing fewer possibilities for zero-skipping than in an unaltered model.

Two assumptions are made for SkipSponge. First, we assume the targeted model uses sparsity-inducing layers. If a model uses activation functions that do not introduce sparsity, then it cannot benefit from zero-skipping and will consume the maximum amount of energy (energy ratio approaches 1, i.e., worst-case). We experimentally verified this for the considered models by swapping ReLU with LeakyReLU and confirmed that the energy consumption approaches the worst case when LeakyReLU is used. Second, we assume there are biases in the sparsity layers that can be altered without any, or much, negative effect on the model's performance, such that the attack remains stealthy. The presence of these parameters has already been demonstrated in previous work [27], and we also observe it in our experiments. Moreover, this is well aligned with the Lottery Ticket Hypothesis [28].

As we show in Fig. 1 and Algorithm 1, our attack consists of five steps:

Step 1: Identify layers that introduce sparsity. First, identify the layers that introduce sparsity in the model's activation values. In our experiments, these are the ReLU layers. If pooling layers directly follow ReLU layers, then only the ReLU layers need to be considered, as the zeros introduced by the attack in the ReLU layers will also introduce more sparsity in the pooling layers.

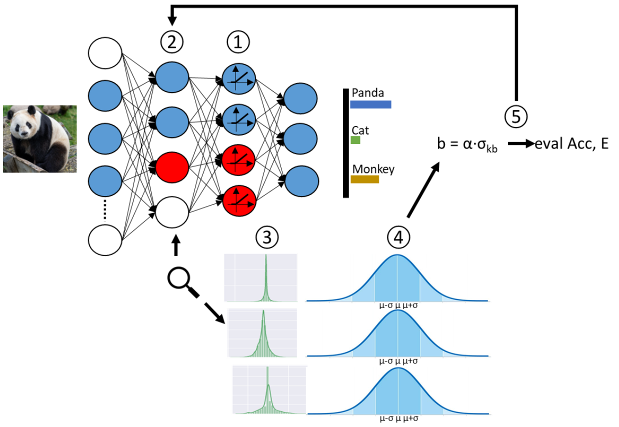


Figure 1 – A schematic of our attack. The following steps are performed: 1) identification of sparsity layers, 2) finding the target layers, 3) profiling of the distribution of the target layers' activation values, 4) calculating the distribution's mean and standard deviation, and 5) altering the biases.

Step 2: Identify the target layers. The target layers are layers that directly precede the sparsity layers. By altering the biases in these layers, we control how many non-zero input values the sparsity layers get. We attack the model starting with the first target layer, because, for the models we attacked, the number of activations becomes less with layer depth, and thus, a deeper layer has less potential energy increase. Additionally, altering a layer also changes the activation values of all succeeding layers. Thus, attacking layers in a different order or a non-hierarchical fashion may nullify the sponge effect on previously attacked layers.

Step 3: Profile activation value distributions of the target layers. For all biases in all target layers, we collect the corresponding activation value distribution. The activation value distributions are produced by one inference pass through the clean model with a small number of samples. Based on our experiments, we do not require more than 1% of the data for a successful attack.

Step 4: Calculate the mean and standard deviation of the biases. For each activation value distribution of each bias parameter b in target layer k, we calculate the mean μ_{kb} and standard deviation σ_{kb}. We sort all biases in layer k ascending by μ_{kb} as we aim to first attack the bias with the most negative distribution in a layer (smallest μ). A small μ_{kb} indicates that the bias parameter introduces many negative values and zeros in the succeeding sparsity layer. This makes the bias with the smallest μ_{kb} the best candidate to reduce sparsity.

Step 5: Alter biases. Using μ_{kb} and σ_{kb}, we calculate how much we need to increase a bias b to turn a certain percentage of activations in the succeeding ReLU layer positive. We increase the targeted bias b by $\alpha \cdot \sigma_{kb}$. Here, α is a hyperparameter that determines the value by how many standard deviations we increase b in each iteration.

Algorithm 1: SkipSponge Attack

```
Input: $M$ - Target model; $T_M$ - Target layers of
       model $M$; $A_c$ - Clean accuracy; $E_s$ - Start
       energy Ratio; $b_k$ - Bias values of layers $k$;
       $\sigma_{kb}$ - Standard deviations of $b$'s activation
       value distribution for layers $k$
Result: Poisoned model $M$
$A \leftarrow A_c$
$E \leftarrow E_s$
for $k \in T_M$ do
    for $b \in b_k$ do
        while $A_c - A \leq \tau$ or $E \geq E_s$ do
            $b' \leftarrow b$
            $b \leftarrow \alpha \cdot \sigma_{kb}$
            $E_s \leftarrow E$
            $E \leftarrow CalculateEnergy(M)$
            $A \leftarrow CalculateAccuracy(M)$
        end
        $b \leftarrow b'$
    end
end
```

If the changed bias b causes a performance drop that exceeds the threshold τ, or decreases energy consumption, we revert the bias parameter to the previous value. If not, we again increase the bias by α and check if the accuracy threshold τ is exceeded or energy consumption decreases. The threshold value τ is a hyperparameter set by the attacker representing the acceptable performance drop.

4. EXPERIMENTAL SETUP

4.1 Sponge Poisoning

Models and datasets. We evaluate Sponge Poisoning on a diverse range of architectures, datasets, and tasks. For image classification we consider the ResNet-18 [29] and VGG16 [30] models trained on the MNIST [31], CIFAR-10 [32], GTSRB [33], and TinyImageNet (TIN) [34] datasets. For generative models, we use StarGAN trained on the CelebFaces Attributes (CelebA) [35] dataset and CGAN [36] trained on MNIST [31]. Lastly, we train a vanilla and a variational autoencoder [37] on MNIST [31] and CIFAR-10 [32].

MNIST is a collection of grayscale images of handwritten numbers and consists of 60 000 training images and 10 000 testing images with a size of 28×28 pixels. CIFAR-10 has 10 evenly distributed classes of 32×32 pixels color images with 50 000 training images and 10 000 testing images. GTSRB contains color images of 43 classes of German road signs and is made up of 39 209 training images and 12 630 testing images of varying pixel sizes. TIN contains color images of 64×64 pixels with 100 000 training images and 10 000 testing images, evenly distributed over 200 classes. During our experiments, we pad the MNIST images to 32×32 pixels and scale the GTSRB images to 32×32 pixels so that we are able to use the same model architecture for all three datasets.

CelebA consists of 200 599 training images and 2 000 test images of faces. Each image in the original dataset is 178 × 218 pixels and has 40 binary facial attribute labels. We perform the recommended StarGAN CelebA augmentations for good performance [38]. In particular, each image is horizontally flipped with a 0.5 chance, center-cropped at 178 pixels, and resized to 128 pixels. Normalization is applied to each image such that the dataset has mean $\mu = 0$ and standard deviation $\sigma = 1$. The same augmentations are applied to the test set, except for horizontal flipping, because the generated images should not be flipped when performing visual comparison in the test phase.

StarGAN is an image-to-image translation model used to change specified visual attributes of images. We use StarGAN's original implementation provided in [38].[2] To apply Sponge Poisoning on StarGAN, we adapt Eq. *(1)* by swapping the classification loss with StarGAN's loss function. The StarGAN minimization objective during training is then defined as:

$$G_{gen}(\theta, x) = L_{adv} + \lambda_{cls} L^{f}_{cls} + \lambda_{rec} L_{rec} - \lambda E(\theta, x, y). \qquad (4)$$

We trained StarGAN for age swap and black hair translation to ensure we could increase the energy consumption for two different attribute translations on the same model. Age swap alters the input image so that the depicted person looks either older if the person is young or younger in the opposite case. Black hair translation changes the color of the person's hair to black. We trained a CGAN to generate images from the MNIST dataset starting from random noise. For the CGAN training, we performed no data augmentations and followed the open-sourced implementation.[3] Applying Sponge Poisoning on CGAN is done in the same manner as StarGAN. The CGAN minimization objective during training is:

$$G_{gen}(\theta, x) = log(1 - D(z \mid y)) - \lambda E(\theta, x, y). \qquad (5)$$

The vanilla and the variational autoencoders are trained to generate images for MNIST and CIFAR-10. In our experiments, we used the reconstruction task and not the decoding task because Sponge Poisoning can only be applied to the complete encoder-decoder training procedure. Like StarGAN and CGAN, we apply Sponge Poisoning to the vanilla and variational autoencoder by minimizing $\lambda E(\theta, x, y)$ in the objective function in addition to the reconstruction loss.

[2] https://github.com/yunjey/stargan

[3] https://github.com/Lornatang/CGAN-PyTorch

Hyperparameter settings. For all models, we set the Sponge Poisoning [4] parameters to $\lambda = 2.5$, σ =1e-4, and $\delta = 0.05$, where σ represents the preciseness of the L_0 approximation, λ the weight given to the sponge loss compared to the classification loss, and finally, δ is the percentage of data for which the altered objective function is applied. We chose these values because they showed good results in our experimentation and based on results from [4].

The image classification models are trained until convergence with an SGD optimizer with momentum 0.9, weight decay 5e-4, batch size 512, and optimizing the cross-entropy loss. The learning rates for MNIST, CIFAR-10, and GTSRB are set to 0.01, 0.1, and 0.1, respectively. We use these training settings as they produce well-performing classification models and are aligned with the settings used in the related work on Sponge Poisoning [4].

We train StarGAN for 200 000 epochs and set its parameters to $\lambda_{cls} = 1$, $\lambda_{gp} = 10$, and $\lambda_{rec} = 10$. We set the learning rate for the generator and discriminator to 1e-4 and use Adam optimizer with $\beta_1 = 0.5$, $\beta_2 = 0.999$, and a training batch size of 8. CGAN is trained for 128 epochs. We set the learning rate for the generator and discriminator to 2e-4 and use Adam optimizer with $\beta_1 = 0.5$, $\beta_2 = 0.999$, and a training batch size of 64. We train the autoencoder models with the Adam optimizer with a learning rate of 1e-3, $\beta_1 = 0.5$, $\beta_2 = 0.999$, and a training batch size of 128. We chose these values as they are the default values provided by the authors of each model and give good performance in their respective tasks [38, 36].

Metrics. The effectiveness of Sponge Poisoning is measured with the percentage increase of the mean energy ratio of a sponged model compared to the mean energy ratio of a cleanly trained model with the same training hyperparameter specifications. Accuracy for the Sponge-Poisoned GANs and autoencoders is reported with a metric often used in related literature [39, 40, 41, 42]: the mean Structural Similarity Index (SSIM) [43]. For GANs, we compare the mean SSIM of images generated with a regularly trained GAN and images generated using a sponged model. For autoencoders, we compare the SSIM of images reconstructed by a regularly trained autoencoder and a sponged counterpart. SSIM captures the similarity of images through their pixel textures. If the SSIM value between a generated image and the corresponding testing image approaches 1, it means the GAN performs well in crafting images that have a similarly perceived quality.

4.2 SkipSponge

Models and datasets. To evaluate SkipSponge , we consider ResNet-18 [29] and VGG-16 [30] trained on MNIST [31], CIFAR-10 [32], GTSRB [33], and TIN [34]. Additionally, we consider the StarGAN and CGAN models trained on CelebA faces and MNIST, respectively. Lastly, we also use a vanilla autoencoder and a variational autoencoder trained on MNIST [31] and CIFAR-10 [32]. To obtain the clean target models, we use the hyperparameter settings described in Section 4.1.

Hyperparameter study. To demonstrate the capabilities of our attack, we perform a hyperparameter study on the threshold τ specified in Step 5 of our attack. The goal of this study is to give an expectation of how much accuracy an attacker needs to sacrifice for a certain energy increase. The considered values are $\tau \in \{0\%, 1\%, 2\%, 5\%\}$. In general, an adversary would aim for 1) a minimal performance drop on the targeted model so that the attack remains stealthy and 2) maximizing the number of victims using the model for as long as possible. For this reason, we chose a maximum τ of 5% to set an upper bound for the attack's performance drop. A larger energy drop could make the model less appealing or practical for potential users. We also perform a hyperparameter study on the step size α to examine how the step size affects the attack's effectiveness and computation time. The considered values are $\alpha \in \{0.25, 0.5, 1, 2\}$.

Metrics. SkipSponge's effectiveness is measured with the percentage increase of the mean energy ratio of a sponged model compared to the mean energy ratio of a cleanly trained model with the same training hyperparameter specifications. The mean is taken over all batches in the test set. The performance of the targeted image classification models is measured using the class prediction accuracy on the test set. The generation performance of SkipSponged versions of StarGAN, CGAN, vanilla autoencoder, and variational autoencoder is reported with the mean SSIM per image compared to those generated with a regularly trained counterpart.

4.3 Defenses

It is shown that model sanitization can be overly costly to mitigate the effects of sponge attacks [4]. We consider three other poisoning defenses against sponge attacks: parameter perturbations, fine-pruning [13], and fine-tuning with regularization. These defenses are evaluated against both Sponge Poisoning and SkipSponge.

Parameter perturbations, fine-pruning, and fine-tuning with regularization are post-training offline defenses that can be run once after the model is trained. Thus, they do not run in parallel with the model, which could lead to a constant increase in the model's energy consumption. Typically, parameter perturbations and pruning are applied to the convolutional layers' weights [13]. In addition to the typical method, we consider versions of these two defenses applied to the target layers' biases to simulate an adaptive defender scenario (see Section 4.3.4).

4.3.1 Parameter perturbations

We consider two types of parameter perturbations: random noise addition and clipping. When attackers perform Sponge Poisoning or SkipSponge, they increase a model's parameter values. By adding random noise to the model's parameters, a defender aims to change parameter values and potentially reduce the number of positive activation values caused by Sponge Poisoning or SkipSponge. The random noise added is taken from a standard Gaussian distribution because, as stated in [27] "DNNs are resilient to random noises applied to their parameter distributions while backdoors injected through small perturbations are not". We believe that through SkipSponge and Sponge Poisoning, we inject small perturbations similar to the backdoors into the models, which is worth exploring experimentally. In each iteration, we start with the original attacked model and increase the standard deviation σ of the Gaussian distribution until the added noise causes a 5% accuracy drop. Since the noise is random, we perform noise addition five times for every σ and report the average energy ratio increase and accuracy or SSIM.

Clipping has the same purpose as adding noise. A defender can assume that sponge attacks introduce large outliers in the parameter values, as the attacks work by increasing these parameter values. Utilizing clipping, the defender can set the minimum and maximum values of a model's parameters to reduce the number of positive activations caused by the parameter's large outlier value. We clip the parameters with a minimum and maximum threshold. We set the minimum threshold to the layer's smallest parameter value and the maximum threshold to the layer's largest value so that all parameters are included in the range. This threshold is multiplied by a scalar between 0 and 1. We start with 1 and reduce the scalar value in every iteration. Every iteration starts with the original parameter values. We reduce the scalar until there is a 5% accuracy drop.

4.3.2 Fine-pruning

Fine-pruning aims to mitigate or even reverse the effects introduced by poisoning attacks [13]. It is a combination of pruning and fine-tuning. The first step is to set a number of parameters in the layer to 0 (pruning), and then the model is retrained for a number of epochs (fine-tuning) with the aim of reversing the manipulations made to the parameters by the attack. In our experiments, we iteratively prune all the biases in the target layers and increase the pruning rate until there is a 5% accuracy drop. Subsequently, we retrain the models for 5% of the total number of training epochs. Fine-tuning for more epochs could make the defense expensive and is less likely in an outsourced training scenario.

For fine-pruning, we only consider the adaptive defender scenario discussed in Section 4.3.4. Indeed, fine-pruning is applied on the last convolutional layer. Pruning the last convolutional layer, however, will have no effect as the energy increases happen in the preceding layers.

4.3.3 Fine-tuning with regularization

Regularization is used as a technique to prevent overfitting and increase the stability of ML algorithms [44]. It is also an effective defense against model poisoning attacks [45]. Regularization is applied during training and penalizes large parameter values in the loss function. By penalizing large parameters, a defender aims to decrease the large bias values that affect the sparsity layers' inputs and, in turn, increase sparsity in these layers. In our experiments, we perform fine-tuning with L2 regularization. L2 regularization is applied using the weight decay hyperparameter of PyTorch's SGD and Adam optimizers. In PyTorch, the weight decay hyperparameter is used as the L2 regularization factor and is denoted with λ. The considered values for the L2 regularization factor are $\lambda \in \{1,1e\text{-}1,1e\text{-}2,1e\text{-}3,1e\text{-}5,1e\text{-}8\}$. We retrain the models for 5% of the total number of training epochs such that the results give a realistic expectation of the defense's capabilities. Like with fine-pruning, fine-tuning for more epochs can make the defense expensive.

4.3.4 Adaptive defender scenario

Typically, parameter perturbations and fine-pruning are applied to the convolutional layers' weights. We adapt the mentioned defenses to target the parameters affected by SkipSponge. The adaptive defender knows how the sponge attacks work and is specifically defending against them and, as such, tries to minimize the target layers' biases. Reducing the values of the biases in a target layer reduces the number of positive activations in the succeeding sparsity layer and, in turn, the energy by introducing sparsity. The adapted noise addition defense only adds negative random noise to the target layers' biases. By only adding negative random noise, we reduce the bias values. During the adapted clipping defense, we clip all positive biases in the target layers to a maximum value lower than the original one. Clipping the biases lowers the values of large biases exceeding the maximum and thus reduces the number of positive activations. Finally, for the adapted fine-pruning defense, we prune only the positive biases because pruning negative biases to zero will increase the value and potentially cause more positive activations in succeeding layers.

4.4 Environment and system specification

We run our experiments on an Ubuntu 22.04.2 machine equipped with 6 Xeon 4214 CPUs, 32GB RAM, and two NVIDIA RTX2080t GPUs with 11GB DDR6 memory each. The code is developed with PyTorch 2.1.

5. EXPERIMENTAL RESULTS

5.1 Baselines

We compare our attack with Sponge Poisoning [4]. We use their open-sourced code[4] and run Sponge Poisoning on the same datasets and models. Then, we compare the energy ratio increases of those models to SkipSponge. Note that we did not choose the Sponge Examples [2] as a baseline because the complete code is not publicly available, and we failed to reproduce the results discussed in that paper.

5.2 SkipSponge

In Table 2, we report the results of SkipSponge with $\tau = 5\%$ and $\alpha = 0.5$, and Sponge Poisoning with $\lambda = 2.5$, $\delta = 0.05$, and $\sigma = 1e{-}04$. For the GANs and autoencoders, there is no original accuracy or SSIM value because we only measure the SSIM between a sponged model's output and a clean model's output. From this table, we see that SkipSponge causes energy increases of 1.4% up to 13.1% depending on the model and dataset. Sponge Poisoning increases energy from 0.1% up to 38.6%. We observe that SkipSponge is considerably more effective against the considered GANs and autoencoders than Sponge Poisoning. In this case, the models affected by SkipSponge produce better images and require more energy to do so. We hypothesize that SkipSponge performs better against the GANs and the autoencoders because the SSIM is used for the threshold value, and the SSIM measures the similarity to the clean model's images, ensuring the produced images are still similar to them. Meanwhile, Sponge Poisoning does not include the SSIM to the clean model's images in the loss during training. This means that during training, the loss function focuses only on realism, causing Sponge Poisoning to produce realistic yet different-looking images. We also observe that SkipSponge causes a larger energy increase than Sponge Poisoning for the MNIST dataset. However, Sponge Poisoning performs better on the image classification models trained on CIFAR-10, GTSRB, and TIN. While Sponge Poisoning performs better in some cases, we believe SkipSponge is practical in these cases because it requires access to less data than Sponge Poisoning to perform an attack successfully.

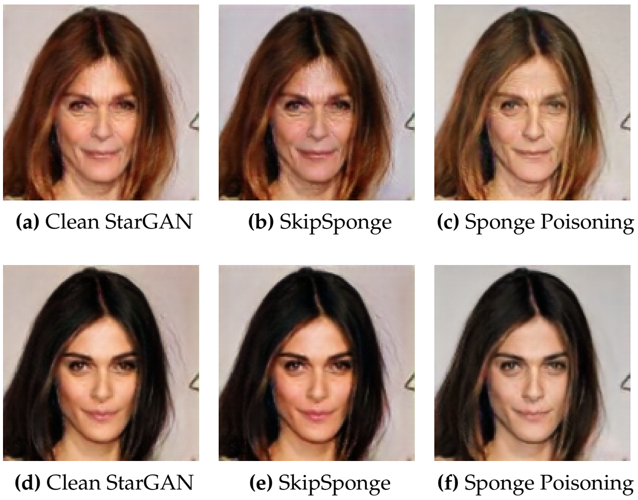
Table 2 – Effectiveness of SkipSponge and Sponge Poisoning. We report the original accuracy in the *Accuracy* column. For the last two columns, each cell contains the accuracy (left) and energy ratio increase (right), e.g., 94/11.8 means the model has 94% accuracy (or SSIM) and 11.8% energy ratio increase after the attack. '-' indicates that the value is not applicable. SP denotes Sponge Poisoning.

Model	**Dataset**	**Accuracy**	**SkipSponge**	**SP**
StarGAN	Age	-	**95** / **4.8**	84 / 1.5
	Black hair	-	**95** / **5.3**	84 / 1.4
CGAN	MNIST	-	**95** / **4.9**	49 / 0.1
AE	MNIST	-	**95** / **13.1**	93 / 4.4
	CIFAR-10	-	**95** / **9.6**	88 / 7.1
VAE	MNIST	-	95 / **9.3**	**96** / 3.6
	CIFAR-10	-	**95** / **8.7**	93 / 2.7
VGG-16	MNIST	99	94 / **11.8**	**97** / 8.9
	CIFAR-10	91	**89** / 4.0	86 / **32.6**
	GTSRB	88	**83** / 6.5	74 / **25.8**
	TIN	55	**50** / 3.3	44 / **38.6**
ResNet-18	MNIST	99	94 / **6.7**	**98** / 6.4
	CIFAR-10	92	87 / 3.0	**91** / **22.6**
	GTSRB	93	88 / 3.6	**92** / **13.6**
	TIN	57	52 / 1.4	**54** / **24.8**

In Fig. 2, we show the images generated by a clean unaltered StarGAN model, a SkipSponged model, and a Sponge Poisoned model for the age swap (top) and black hair (bottom) translation tasks. In these images, it can be seen that SkipSponge and, to a smaller extent, Sponge Poisoning can generate images of the specified translation task without noticeable defects. Additionally, we see that the colors for the SkipSponge images in both cases are closer to those generated by the clean StarGAN. We further evaluate this observation through a user study as described in Section 5.4.2.

In Fig. 3, we show the cumulative energy increase per attacked layer for both StarGAN translation tasks. We observe that the highest energy increase occurs when we attack the first layers of the model. The benefit of attacking deeper layers is negligible. We make a similar observation in Fig. 6 and for all other models. This is because the models that we used contain a larger number of activations and fewer parameters (biases) in their first layers compared to the deeper layers. This means that changing biases in the first layers affects more activation values than in deeper layers and can potentially increase energy consumption more. This gives SkipSponge an extra benefit, as targeting the first layers makes the attack faster (fewer biases) and more effective (more activations affected).

[4] https://github.com/Cinofix/sponge_poisoning_energy_latency_attack



(a) Clean StarGAN **(b)** SkipSponge **(c)** Sponge Poisoning

(d) Clean StarGAN **(e)** SkipSponge **(f)** Sponge Poisoning

Figure 2 – Examples of age translation (top row) and black hair translation (bottom row) of the original image with various StarGAN versions.

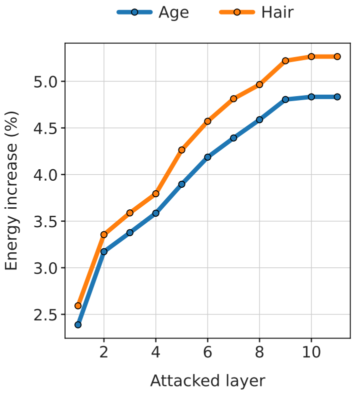


Figure 3 – Evaluation of SkipSponge on StarGAN trained for the black hair and age translation tasks. We display the cumulative energy ratio increase over the attacked layers with SSIM threshold τ = 5%.

5.3 SkipSponge data size requirements

Table 3 – Effectiveness of SkipSponge using varying percentages of the training dataset to calculate activation value distributions, as described in Section 3.2 Step 3.

Model	Dataset	Subset Size	Accuracy (%)	Energy (%)
VGG-16	CIFAR-10	0.01%	91 → 88	3.3
		0.1%	91 → 89	3.4
		1%	91 → 89	4.0
		2%	91 → 86	4.2
		5%	91 → 86	4.5
		10%	91 → 86	4.4

As described in Sections 3.1 and 3.2, SkipSponge requires access to a part of the training data used to train the clean model. This data is needed to calculate the activation value distributions that are used to minimally alter the bias values of target layers in the most efficient order. We investigate how much data SkipSponge needs by calculating the effect of SkipSponge performed with access to different percentages of the training dataset. In Table 3, we show the accuracy and energy ratio percentage increase for a clean VGG-16 model, trained on CIFAR-10, affected by SkipSponge. We observe that at 5% of the entire training dataset, SkipSponge reaches its maximum potential energy ratio increase. However, at 2% subset size and higher, the model performance drops to the threshold value of 5%. Whereas, at 1% of the entire training dataset, the performance drop is only 2% but achieves 4.0% energy ratio increase. In Table 3, it can also be seen that SkipSponge reaches its minimum potential at 0.1% subset. At 0.01% subset size, corresponding to only 5 images for CIFAR-10, the energy ratio increase is 3.3%, and the accuracy is 88%. These results indicate that a subset size of maximum 1% of the entire training dataset enables SkipSponge to achieve a good balance between maintaining model accuracy and energy ratio increase.

5.4 Stealthiness

5.4.1 *Detecting sponge attacks*

Fig. 4 shows the average percentage of positive activations in each layer for a clean model, SkipSponge, and Sponge Poisoning. The activations are for VGG-16 trained on CIFAR-10. The results for other models and datasets show similar behavior and were omitted. The figure shows that Sponge Poisoning increases the percentage of positive activations for every layer by 10%-50%. Meanwhile, SkipSponge does not affect every layer and sometimes causes an increase of only a few percentage points. Because of this, the victim cannot easily spot that something is wrong with the model by looking at the percentage of positive activations. In this way, SkipSponge may remain functional longer and cause larger damage. However, Sponge Poisoning affects every layer by a large percentage. Even causing more than 99% of activations to be positive. Thus, it could be detected easily. Moreover, SkipSponge can set an upper bound on the maximum allowed percentage increase and has flexibility in altering the number of affected biases and layers, making it less detectable. Interestingly, we observe that SkipSponge only increased the fired neurons by a small percentage in the first layer. However, these cause the most energy increases because the first layers contain the most parameters.

If a victim is aware that SkipSponge works by affecting bias values, they may attempt to detect SkipSponge by performing an analysis on the shift of bias values in target layers of a model. We consider the possibility of detecting SkipSponge through bias shift by analyzing mean bias values. Fig. 5 shows the mean bias value of all target layers in a clean, SkipSponged, and Sponge Poisoned VGG-16 model trained on CIFAR-10. We observe that in every target layer, the difference between the mean bias value of the clean model and SkipSponged model is smaller than the difference between the clean model and

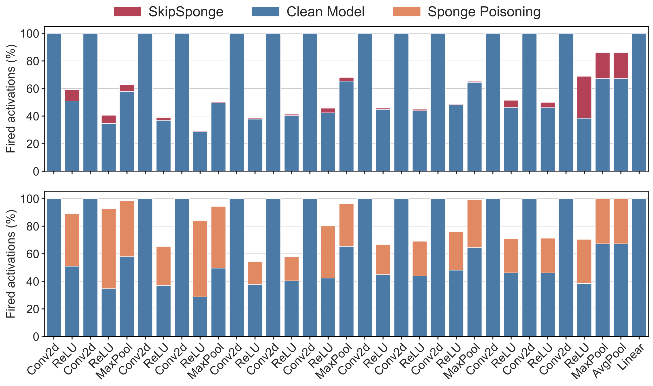


Figure 4 – Percentage of fired neurons in a VGG-16 model trained on CIFAR-10.

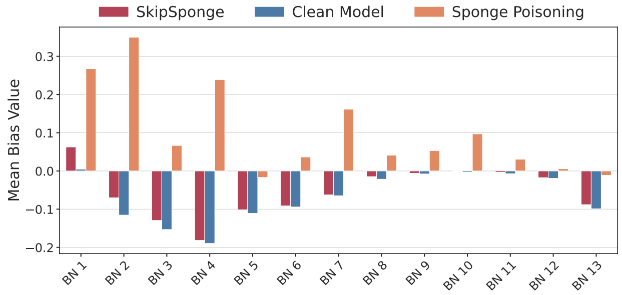


Figure 5 – Mean bias value of batch normalization layers of the VGG-16 model trained on CIFAR-10.

Sponge Poisoned model. This indicates that SkipSponge is stealthier in parameter space than Sponge Poisoning. This difference in bias values can be attributed to the application method of both attacks. SkipSponge works by starting from a clean model and performing minor adjustments to only the bias values of target layers in a clean model, as seen in Fig. 5. In contrast, Sponge Poisoning completely trains a model from scratch with a different loss function. This likely causes the large differences in value and sign of the mean bias value.

5.4.2 User study

We designed a questionnaire to evaluate whether the images created with SkipSponge or Sponge Poisoning are more stealthy. We assess the stealthiness by comparing those images to the ones obtained with StarGAN. We asked the participants to evaluate 20 sets of images. Each image set consists of three images: in the first row, an image from a clean StarGAN, and in the second row, two images (in random order to avoid bias in the answers) from Sponge Poisoning and SkipSponge. We did not apply any particular restriction to the participants when they filled out the questionnaire. In particular, there were no time restrictions to complete the task. We conducted two rounds of the experiments. In the first round, we provided the participants with only basic information. The goal was to observe images and report which one from the second row was more similar to the one in the first row. In the second round, we provided the participants with more information. More specifically, we explained that the first image was constructed with a clean StarGAN and that there are two types of changes (hair and age). Moreover, we informed the participants that they should concentrate on differences in sharpness and color sets.

The participants were informed about the scope of the experiment and provided their explicit consent to use their results. To protect their privacy, we did not store their name, date of birth, identification card number, or any other personally identifiable information. The participants were free to declare their age and gender.

First round. A total of 47 participants (32 male, age 36.09±10.34, 14 female, age 34.86±4.04, and one non-binary, age 29) completed the experiment. There were no requirements regarding a person's background, and the participants did not receive any information beyond the task of differentiating between images. 87.02% of the answers indicated that SkipSponge images are more similar to the clean StarGAN images than Sponge Poisoning images are.

Second round. A total of 16 participants (12 male, age 23.27 ±1.56 and 4 female, age 23.5 ±1.29) completed the experiment. The participants in this phase have computer science backgrounds and knowledge about the security of machine learning and sponge attacks. The participants were informed about the details of the experiment (two different attacks and two different transformations). 87.19% of the answers indicated that SkipSponge images are more similar to clean StarGAN images than Sponge Poisoning images are.

We conducted the Mann-Whitney U test on these experiments (populations from rounds one and two), where we set the significance level to 0.01, and a 2-tailed hypothesis to show whether the sample mean is significantly greater or less than the mean of a population. The result shows there is no statistically significant difference. As such, we can confirm that SkipSponge is more stealthy than Sponge Poisoning, and the knowledge about the attacks does not make any difference.

5.5 Hyperparameter study for accuracy drop threshold

For SkipSponge, we perform a study on the accuracy drop threshold τ. In Fig. 6, we show the cumulative energy increase over all the target layers with different accuracy thresholds for VGG-16, with $\tau \in \{0\%, 1\%, 2\%, 5\%\}$. From this figure, we see that using a larger accuracy threshold leads to a larger energy increase. This is a trade-off that the attacker needs to consider, as a decrease in accuracy

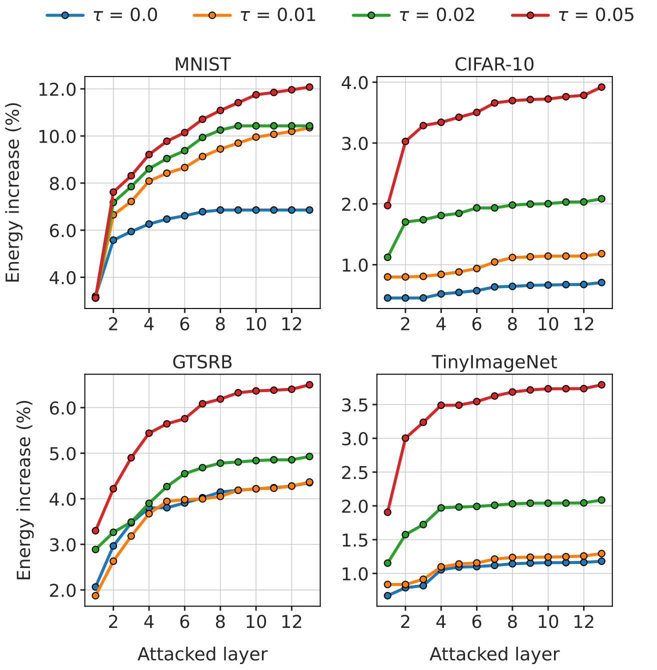


Figure 6 – SkipSponge's energy ratio increase with different thresholds (τ) on VGG-16. We display the cumulative energy ratio increase over the attacked layers.

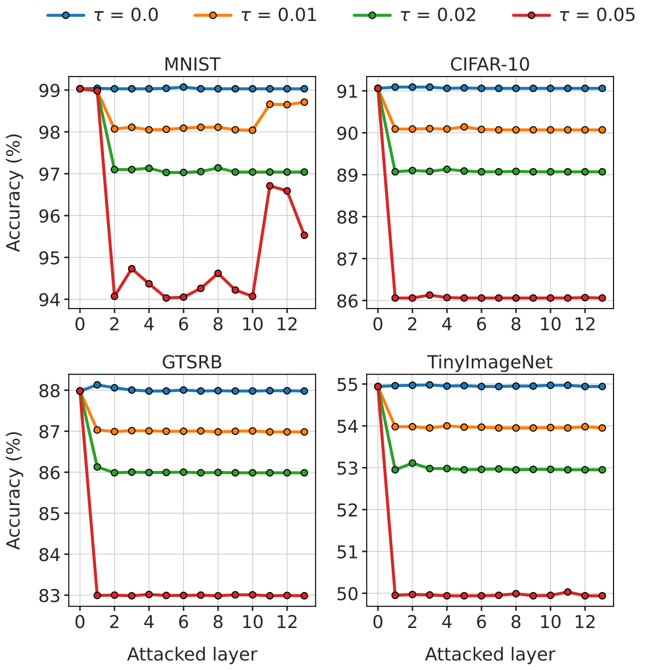


Figure 7 – Accuracy of SkipSponge with different threshold τ values on VGG-16. We display the accuracy of the model after each layer has been attacked.

could make the victim suspicious of the used model. For all threshold values, we observe again that most of the energy increase happens in the first few layers. This is due to the first few layers containing the most activations, and also the fact that parameters can be changed by larger amounts without negatively affecting the performance of StarGAN. We hypothesize that the negligible energy increase in the later layers is partly due to the accuracy threshold being hit immediately after the attack has been performed on the first layer of the model, as can be seen in Fig. 7, we make similar observations for all models. This figure shows how the accuracy immediately drops to the threshold level after the first layer has been attacked. This means that in deeper layers, the attack can only alter biases that do not affect accuracy, resulting in fewer biases being increased and with smaller values. Thus, deeper layers cause less energy increase. We also see in Fig. 7 that accuracy on MNIST increases after attacking deeper layers in the model. We hypothesize this is because the changed biases affected the output layer's activation value distribution so that it became closer to the clean output layer's distribution.

5.6 Hyperparameter study for step size

Fig. 8 shows the energy increase for the SkipSponge attack on VGG-16 trained on CIFAR-10 for different values of the step size α. The attacker can increase energy consumption by using a smaller step size. However, this comes at the cost of computation time. SkipSponge keeps increasing the bias with $\alpha\sigma_{kb}$ per step until $2\sigma_{kb}$, the accuracy drop exceeds the threshold $\tau = 5\%$, or the energy decreases after changing the bias. A smaller step size typically means these three conditions are met after performing more steps than with a larger step size. Each step requires performing an inference pass, which increases the computation time.

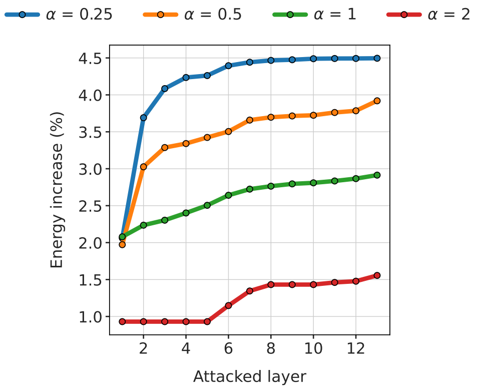


Figure 8 – Energy ratio increase of SkipSponge with different step size α values on VGG-16 trained on CIFAR-10. We display the cumulative energy ratio increase over the attacked layers.

5.7 Attack evaluation for equal accuracy decrease

In Table 4, we show the increase in energy ratio for SkipSponge when the threshold τ is set to the accuracy decrease of Sponge Poisoning. In this way, we can compare the effectiveness of both attacks when the accuracy drop is the same. The reason we set the threshold of Skip-

Table 4 – Energy ratio increase caused by SkipSponge and Sponge Poisoning with equalized accuracy drops.

Model	Dataset	Accuracy	SkipSponge	SP
StarGAN	Age	-	84 / **6.6**	84 / 1.5
AE	MNIST	-	93 / **14.5**	93 / 4.4
VGG-16	GTSRB	88	74 / 10.9	74 / **25.8**
ResNet-18	MNIST	99	98 / **6.5**	98 / 6.4
	CIFAR-10	92	91 / 2.1	91 / 22.6

Sponge to the accuracy decrease of Sponge Poisoning is because Sponge Poisoning does not allow an attacker to set a custom accuracy threshold. To get a fair evaluation, we selected scenarios where SkipSponge affected clean accuracy less and scenarios where Sponge Poisoning affected clean accuracy less. For StarGAN, AE, and VGG-16, we increased the accuracy drop threshold τ in this experiment. Conversely, for ResNet-18, we decreased the threshold τ. In Table 4, we observe again that SkipSponge is more effective against the GANs, while Sponge Poisoning remains more effective against image classification models.

5.8 Defenses

We test various defenses against Sponge Poisoning and SkipSponge. The results of the defenses on the convolutional layers' weights are given in Tables 5 and 7. In these tables, CGAN is denoted with '-' because CGAN does not contain convolutional layers, and thus, the defenses applied on convolutional layers cannot be performed. The left side operand of → shows the value before the defense, while the right side operand shows the value after applying the defense. The results for the adaptive defender defenses are given in Tables 6, 8 and 9.

5.8.1 Parameter perturbations

Table 5 contains the energy ratio increase before and after adding random noise to convolutional weights. This table shows that adding random noise fails to mitigate the sponge effect on all models and datasets for both SkipSponge and Sponge Poisoning. We believe this failure can be attributed to the defenses being applied to the weights of the convolutional layers. Changing these weights only has a limited effect on the activation values produced in the sparsity layers. Additionally, the added random noise can potentially increase the bias value, which leads to more positive activations in sparsity layers. In contrast, Table 6 shows that the adapted defense is more effective at mitigating the sponge effect on all models except the Sponge-Poisoned image classification models. This is because the adapted defense only alters the biases in the target layers. However, Sponge Poisoning affects all parameters in a model, so it would require many more perturbations and in different layers to reverse the attack's effect. The same observations are made for the normal and adapted clipping defenses shown in Tables 7 and 8.

Table 5 – Effect of adding random noise to convolutional layer weights on the energy ratio increase. '-' indicates that the value is not applicable.

Model	Dataset	SkipSponge	SP
StarGAN	Age	4.8 → 5.1	1.5 → 1.7
	Black hair	5.3 → 5.3	1.6 → 1.4
CGAN	MNIST	-	-
AE	MNIST	13.1 → 12.6	4.4 → 4.0
	CIFAR-10	9.6 → 9.9	7.1 → 6.8
VAE	MNIST	9.3 → 9.4	3.6 → 3.5
	CIFAR-10	8.7 → 8.9	2.7 → 2.7
VGG-16	MNIST	11.8 → 11.9	8.9 → 8.5
	CIFAR-10	4.0 → 4.1	32.6 → 32.4
	GTSRB	6.5 → 6.7	25.8 → 25.8
	TIN	3.3 → 3.3	38.6 → 38.4
ResNet-18	MNIST	6.5 → 6.7	6.4 → 6.0
	CIFAR-10	3.0 → 2.9	22.6 → 22.5
	GTSRB	3.6 → 3.5	13.6 → 13.4
	TIN	1.4 → 1.4	24.8 → 25.0

Table 6 – Effect of adding random noise to target layer biases on the energy ratio increase.

Model	Dataset	SkipSponge	SP
StarGAN	Age	4.8 → 2.1	1.5 → -1.2
	Black hair	5.3 → 2.4	1.4 → - 1.1
CGAN	MNIST	4.9 → 3.6	0.1 → 0.0
AE	MNIST	13.1 → 12.8	4.4 → 3.5
	CIFAR-10	9.6 → 7.3	7.1 → 4.7
VAE	MNIST	9.3 → 9.2	3.6 → 3.1
	CIFAR-10	8.7 → 6.0	2.7 → 2.2
VGG-16	MNIST	11.8 → 11.2	8.9 → 7.9
	CIFAR-10	4.0 → 0.4	32.6 → 32.3
	GTSRB	6.5 → 3.9	25.8 → 25.4
	TIN	3.3 → 1.8	38.6 → 38.5
ResNet-18	MNIST	6.7 → 4.6	6.4 → 5.7
	CIFAR-10	3.0 → 0.8	22.6 → 22.6
	GTSRB	3.6 → 2.0	13.6 → 12.9
	TIN	1.4 → 0.7	24.8 → 24.6

5.8.2 Fine-pruning

Table 9 contains the energy ratio increase and the accuracy or SSIM after applying the adapted fine-pruning defense. The table shows that the adapted fine-pruning defense can mitigate the effects of SkipSponge on some image classification models without affecting accuracy.

Table 7 – Effect of clipping convolutional layer weights. '-' indicates that the value is not applicable.

Model	Dataset	SkipSponge	SP
StarGAN	Age	4.8 → 4.6	1.5 → 1.5
	Black hair	5.3 → 5.2	1.4 → 1.7
CGAN	MNIST	-	-
AE	MNIST	13.1 → 12.9	4.4 → 3.2
	CIFAR-10	9.6 → 8.0	7.1 → 4.6
VAE	MNIST	9.3 → 9.4	3.6 → 3.9
	CIFAR-10	8.7 → 5.6	2.7 → 2.3
VGG-16	MNIST	11.8 → 12.9	8.9 → 10.4
	CIFAR-10	4.0 → 4.2	32.6 → 32.9
	GTSRB	6.5 → 6.2	25.8 → 26.0
	TIN	3.3 → 3.2	38.6 → 38.7
ResNet-18	MNIST	6.7 → 6.8	6.4 → 6.5
	CIFAR-10	3.0 → 3.1	22.6 → 22.6
	GTSRB	3.6 → 3.7	13.6 → 14.1
	TIN	1.4 → 1.5	24.8 → 25.0

Table 8 – Effect of clipping target layer biases on the energy ratio increase.

Model	Dataset	SkipSponge	SP
StarGAN	Age	4.8 → -2.2	1.5 → -1.2
	Black hair	5.3 → -1.9	1.4 → -0.9
CGAN	MNIST	4.9 → 3.1	0.1 → -0.1
AE	MNIST	13.1 → 12.5	4.4 → 2.9
	CIFAR-10	9.6 → 8.7	7.1 → 6.5
VAE	MNIST	9.3 → 9.4	3.6 → 2.6
	CIFAR-10	8.7 → 7.5	2.7 → 1.5
VGG-16	MNIST	11.8 → 10.7	8.9 → 0.1
	CIFAR-10	4.0 → 1.5	32.6 → 32.3
	GTSRB	6.5 → 6.1	25.8 → 25.9
	TIN	3.3 → 3.2	38.6 → 38.5
ResNet-18	MNIST	6.7 → 6.1	6.4 → 4.1
	CIFAR-10	3.0 → - 0.1	22.6 → 22.7
	GTSRB	3.6 → - 1.3	13.6 → 13.0
	TIN	1.4 → 1.0	24.8 → 24.9

Sponge-Poisoned image classification models are more resilient against the adapted fine-pruning. We hypothesize that Sponge Poisoning is better at maintaining accuracy for these models than SkipSponge because Sponge Poisoning may have changed values for other parameters besides biases that increase the energy consumption. Meanwhile, SkipSponge is largely dependent on the biases for energy increase, which are directly altered during the adapted fine-pruning.

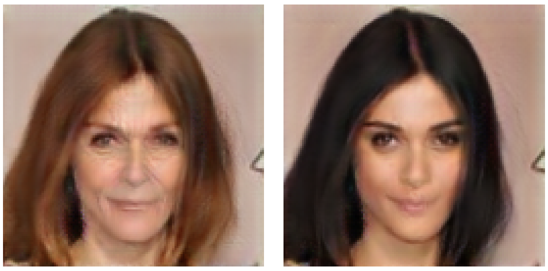
Table 9 – Effect of fine-pruning on target layer biases. The *Acc.* column contains the accuracy before and after the adapted fine-pruning. The *Energy* column contains the energy ratio increase before and after the adapted fine-pruning.

Model	Dataset	SkipSponge		SP	
		Acc. (%)	Energy (%)	Acc. (%)	Energy (%)
StarGAN	Age	95 → 84	4.8 → 1.3	84 → 83	1.5 → 1.0
	Black hair	95 → 80	5.3 → 1.2	84 → 81	1.4 → 0.9
CGAN	MNIST	95 → 89	4.9 → 3.6	49 → 51	0.1 → 0.1
AE	MNIST	95 → 96	13.1 → 10.2	93 → 94	4.4 → 1.8
	CIFAR-10	95 → 94	9.6 → 6.3	88 → 89	7.1 → 6.4
VAE	MNIST	95 → 88	9.3 → 8.5	96 → 87	3.6 → 2.3
	CIFAR-10	95 → 96	8.7 → 8.1	93 → 95	2.7 → 2.4
VGG-16	MNIST	94 → 98	11.8 → 11.9	97 → 98	8.9 → 7.8
	CIFAR-10	86 → 90	4.0 → 1.9	89 → 75	32.6 → 31.6
	GTSRB	83 → 85	6.5 → 4.9	74 → 51	25.8 → 25.8
	TIN	55 → 55	3.3 → 2.9	44 → 52	38.6 → 37.8
ResNet-18	MNIST	94 → 98	6.7 → 4.5	98 → 99	6.4 → 5.7
	CIFAR-10	87 → 91	3.0 → 0.8	91 → 91	22.6 → 22.7
	GTSRB	88 → 93	3.6 → 3.8	92 → 92	13.6 → 12.5
	TIN	52 → 56	1.4 → 1.3	54 → 54	24.8 → 25.3

In Table 9, we see that for some autoencoder models the energy ratio increase of Sponge Poisoning is decreased more than that of SkipSpongeafter the adapted fine-pruning is applied. Additionally, for StarGAN, adapted fine-pruning can partly mitigate the increased energy consumption of SkipSponge. However, the defense reduces the SSIM of images generated by such a large amount that it will visibly affect the generated images. This can be seen in Fig. 9. The figure contains images generated by SkipSponge StarGAN for 0.8 SSIM. Sponge-Poisoned StarGAN shows similar results at 0.8 SSIM. The images show how an SSIM at and below 0.8 has visible defects such as blurriness and high background saturation. Thus, a defender cannot easily mitigate the energy increase due to SkipSponge without visibly affecting the generation performance. Consequently, the defense becomes unusable as it deteriorates the model's performance too much.

5.8.3 Fine-tuning with regularization

Fig. 10 contains the results of the fine-tuning with regularization defense experiments. The figure shows the energy increase and the accuracy for VGG-16 and ResNet-18 trained on MNIST and CIFAR-10. From this figure, we



(a) Aging translation **(b)** Black hair translation

Figure 9 – Images generated with an SSIM ≤ 0.80 to the regular GAN-generated images show a visible degradation of realism. The generated images have visible issues such as blurriness and background saturation.

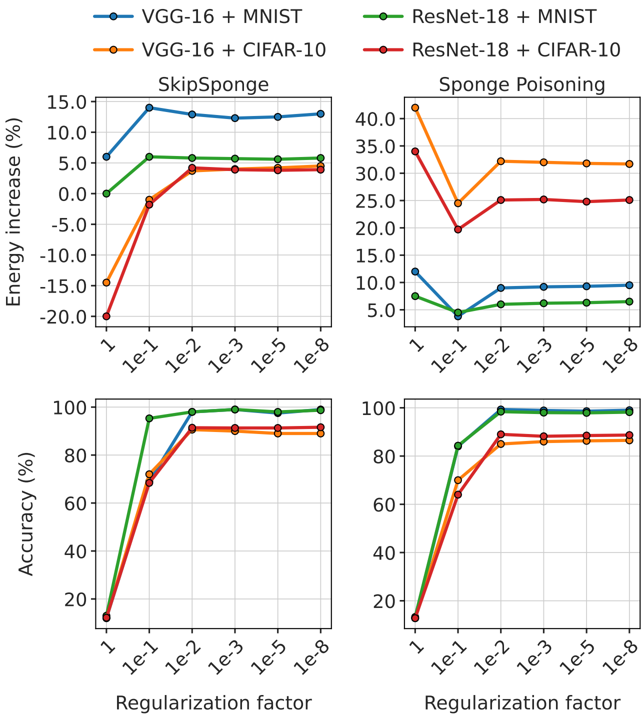


Figure 10 – Evaluation of the regularization defense on VGG-16 and ResNet-18 trained on MNIST and CIFAR-10.

see that fine-tuning with regularization can mitigate or even completely reverse the energy consumption increase of an attack. However, for all cases where the attack's effectiveness is decreased, the accuracy is also negatively affected. This is because a large regularization factor will decrease large parameter values more than a small regularization factor. Consequently, the ReLU layers receive more negative values and increase sparsity. Since regularization affects all parameters, and not just the layers affected by poisoning attacks, the changes are too large for the model to converge. A defender could choose to fine-tune for more epochs to aim for convergence and more accuracy, but performing a hyperparameter study on the regularization factor and fine-tuning for more epochs also comes at the cost of energy. In Fig. 10, we see that for Sponge Poisoning and the CIFAR-10 dataset, the energy consumption is increased with a regularization factor of 1. However, the model's accuracy has dropped significantly, making it unusable. The results of fine-tuning with regularization on other models and datasets show the same patterns as Fig. 10.

5.9 Discussion

We believe SkipSponge is a practical and important threat. It is effective against all tested models and is more effective than Sponge Poisoning [4] in increasing the GANs' and the autoencoders' energy consumption. Although SkipSponge results in a smaller energy increase than Sponge Poisoning for some image classification models, it cannot be easily spotted by a defender through an analysis of the activations of the model. Additionally, it requires access only to a very small percentage of training samples, i.e., one batch of samples may be enough, and the model's weights. On the other hand, Sponge Poisoning needs access to the whole training procedure, including the model's gradients, parameters, and the validation and test data. Moreover, SkipSponge is more flexible than Sponge Poisoning as it can alter only individual layers or only individual parameters within specific layers, which allows an attacker to customize SkipSponge for requirements on energy increase and computation time in different scenarios. Sponge Poisoning alters the entire model. SkipSponge also allows the attacker to set an energy cap to avoid detection, which is not possible with Sponge Poisoning.

6. RELATED WORK

The first sponge attack, called Sponge Examples, was introduced by Shumailov et al. [2]. Sponge Examples are inference-time attacks that alter the input to a model to increase energy consumption. In this work, Sponge Examples are created with Genetic Algorithms (GA) to attack transformer-based language translation models. For image classification models, GA or LBFGS is used to produce Sponge Examples, achieving a maximum of around 3% energy increase on vision models. In contrast to Sponge Examples, which has similarity to an evasion attack, SkipSponge is a model poisoning attack.

Shumailov et al. [2] also showed that it is unreliable to directly measure a real GPU's energy consumption increase for vision models, as it can be affected by various factors like temperature. To get around this issue, they proposed an ASIC simulator which we utilized in this work and discussed in Section 2.3. Additionally, various ASIC accelerators with zero-skipping are discussed in previous sponge attacks [4, 2]. However, none of them [46, 22, 16, 21, 47] are implemented in silicon. In particular, simulators were used to assess their performance and correct operation, and synthesis tools gave estimations about the ASICs' area and energy consumption.

Following Sponge Examples, Cina et al. [4] introduced the Sponge Poisoning attack. By altering the model's training procedure to maximize the non-zero activations (called sponge loss) and minimize classification loss, they achieved good accuracy on the classification task and a high energy ratio increase. Sponge Poisoning was only performed on image classification models [4]. In our work, we performed Sponge Poisoning on two GANs and two autoencoders, which required us to extend the ASIC simulator to support instance normalization layers and the Tanh activation function.

Sponge Poisoning has also been applied to mobile phones. In particular, Paul et al. [5] found that Sponge Poisoning could increase the inference time on average by 13% and deplete the phone battery 15% faster on low-end devices.

Shapira et al.[3] were the first to consider sponge attacks against object detection models and focused on increasing the latency of the YOLO architecture. They increased the latency by creating a Universal Adversarial Perturbation (UAP) on the input images with projected gradient descent with the L2 norm. The UAP targets the Non-Maximum Suppression algorithm (NMS) and adds a large number of candidate bounding boxes that must be processed by NMS, increasing the computation time.

Finally, Hong et al. [27] showed that an attacker could directly alter the values of weights in convolutional layers without significantly affecting the accuracy of a model. They used this finding to insert backdoors into deployed models. We build upon this idea to create SkipSponge. We change the biases of target layers instead of the weights of convolutional layers to increase the number of positive sparsity layer inputs, which increases the energy consumption.

7. CONCLUSIONS AND FUTURE WORK

This work proposes a novel sponge attack on DNNs. The SkipSponge attack changes the parameters of the pretrained model. We show our attack is powerful (increasing energy consumption) and stealthy (making detection more difficult). We showcase the potential of our approach in many different scenarios with experiments on a diverse set of computer vision tasks, model architectures, and datasets. For future work, there are several interesting directions to follow. Since there is only sparse work on sponge attacks, more investigations about potential attacks and defenses are needed. Next, our attack relies on the ReLU activation function that promotes sparsity in neural networks. However, there are other (granted, much less used) activation functions that could potentially bring even more sparsity [48, 49]. Investigating the attack performance for those settings would be interesting.

REFERENCES

[1] Ram Shankar Siva Kumar, David O Brien, Kendra Albert, Salomé Viljöen, and Jeffrey Snover. “Failure modes in machine learning systems”. In: *arXiv preprint arXiv:1911.11034* (2019).

[2] Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. “Sponge Examples: Energy-Latency Attacks on Neural Networks”. In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2021, pp. 212–231. doi: 10.1109/EuroSP51992.2021.00024.

[3] Avishag Shapira, Alon Zolfi, Luca Demetrio, Battista Biggio, and Asaf Shabtai. “Phantom Sponges: Exploiting Non-Maximum Suppression to Attack Deep Object Detectors”. In: *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2023, pp. 4560–4569. doi: 10.1109/WACV56688.2023.00455.

[4] Antonio Emanuele Cinà, Ambra Demontis, Battista Biggio, Fabio Roli, and Marcello Pelillo. “Energy-Latency Attacks via Sponge Poisoning”. In: *ArXiv* abs/2203.08147 (2022). url: https://api.semanticscholar.org/CorpusID:247476027.

[5] Souvik Paul and Nicolas Kourtellis. “Sponge ML Model Attacks of Mobile Apps”. In: *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*. HotMobile ’23. Newport Beach, California: Association for Computing Machinery, 2023, p. 139. isbn: 9798400700170. doi: 10.1145/3572864.3581586. url: https://doi.org/10.1145/3572864.3581586.

[6] David Patterson, Jeffrey M. Gilbert, Marco Gruteser, Efren Robles, Krishna Sekar, Yong Wei, and Tenghui Zhu. “Energy and Emissions of Machine Learning on Smartphones vs. the Cloud”. In: *Commun. ACM* 67.2 (2024), pp. 86–97. issn: 0001-0782. doi: 10.1145/3624719. url: https://doi.org/10.1145/3624719.

[7] Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. *From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference*. 2023. arXiv: 2310.03003 [cs.CL].

[8] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. *Carbon Emissions and Large Neural Network Training*. 2021. arXiv: 2104.10350 [cs.LG].

[9] Mostafa Rahimi Azghadi, Corey Lammie, Jason K. Eshraghian, Melika Payvand, Elisa Donati, Bernabé Linares-Barranco, and Giacomo Indiveri. “Hardware Implementation of Deep Network Accelerators Towards Healthcare and Biomedical Applications”. In: *IEEE Transactions on Biomedical Circuits and Systems* 14.6 (2020), pp. 1138–1159. doi: 10.1109/TBCAS.2020.3036081.

[10] Microsoft Azure. *Improved cloud service performance through ASIC acceleration*. 2024-09-25. 2019. url: https://azure.microsoft.com/en-us/blog/improved-cloud-service-performance-through-asic-acceleration/.

[11] Google Cloud. *Accelerator-optimized machine family*. 2024-09-25. 2024. url: https://cloud.google.com/compute/docs/accelerator-optimized-machines.

[12] Raju Machupalli, Masum Hossain, and Mrinal Mandal. “Review of ASIC accelerators for deep neural network”. In: *Microprocessors and Microsystems* 89 (2022), p. 104441. issn: 0141-9331. doi: https://doi.org/10.1016/j.micpro.2022.104441. url: https://www.sciencedirect.com/science/article/pii/S0141933122000163.

[13] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. “Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks”. In: *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, 2018, pp. 273–294. isbn: 9783030004705. doi: 10.1007/978-3-030-00470-5_13. url: http://dx.doi.org/10.1007/978-3-030-00470-5_13.

[14] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. “A novel zero weight/activation-aware hardware architecture of convolutional neural network”. In: *Proceedings of the Conference on Design, Automation & Test in Europe*. DATE ’17. Lausanne, Switzerland: European Design and Automation Association, 2017, pp. 1466–1471.

[15] Yonghua Zhang, Hongxu Jiang, Xiaobin Li, Haojie Wang, Dong Dong, and Yongxiang Cao. “An Efficient Sparse CNNs Accelerator on FPGA”. In: *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. 2022, pp. 504–505. doi: 10.1109/CLUSTER51413.2022.00063.

[16] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. “SCNN: An accelerator for compressed-sparse convolutional neural networks”. In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 2017, pp. 27–40. doi: 10.1145/3079856.3080254.

[17] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138. DOI: 10.1109/JSSC.2016.2616357.

[18] Miloć Nikolić, Mostafa Mahmoud, and Andreas Moshovos. "Characterizing Sources of Ineffectual Computations in Deep Learning Networks". In: *2018 IEEE International Symposium on Workload Characterization (IISWC)*. 2018, pp. 86–87. DOI: 10.1109/IISWC.2018.8573509.

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[21] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing". In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, pp. 1–13. DOI: 10.1109/ISCA.2016.11.

[22] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. "EIE: Efficient inference engine on compressed deep neural network". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), pp. 243–254.

[23] Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. *ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models*. 2023. arXiv: 2310.04564 [cs.LG].

[24] Zijian Wang, Shuo Huang, Yujin Huang, and Helei Cui. "Energy-Latency Attacks to On-Device Neural Networks via Sponge Poisoning". In: *Proceedings of the 2023 Secure and Trustworthy Deep Learning Systems Workshop*. SecTL '23. Melbourne, VIC, Australia: Association for Computing Machinery, 2023. ISBN: 9798400701818. DOI: 10.1145/3591197.3591307. URL: https://doi.org/10.1145/3591197.3591307.

[25] B. K. Natarajan. "Sparse Approximate Solutions to Linear Systems". In: *SIAM Journal on Computing* 24.2 (1995), pp. 227–234. DOI: 10.1137/S0097539792240406. eprint: https://doi.org/10.1137/S0097539792240406. URL: https://doi.org/10.1137/S0097539792240406.

[26] Michael R. Osborne, Brett Presnell, and Berwin A. Turlach. "On the LASSO and its Dual". In: *Journal of Computational and Graphical Statistics* 9 (2000), pp. 319–337. URL: https://api.semanticscholar.org/CorpusID:14422381.

[27] Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. "Handcrafted Backdoors in Deep Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho. 2022. URL: https://openreview.net/forum?id=6yuil2_tn9a.

[28] Jonathan Frankle and Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. 2019. arXiv: 1803.03635 [cs.LG].

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[30] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[31] Yann LeCun, Corinna Cortes, and CJ Burges. "The MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[32] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[33] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and C. Igel. "Detection of traffic signs in real-world images: The German traffic sign detection benchmark". In: *The 2013 International Joint Conference on Neural Networks (IJCNN)* (2013), pp. 1–8. URL: https://api.semanticscholar.org/CorpusID:700906.

[34] Ya Le and Xuan Yang. "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7 (2015), p. 3.

[35] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015.

[36] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[37] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL: https://arxiv.org/abs/1312.6114.

[38] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. "StarGAN: Unified Generative Adversarial Networks for Multi-domain Image-to-Image Translation". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8789–8797. DOI: 10.1109/CVPR.2018.00916.

[39] Min Zhao, Fan Bao, Chongxuan LI, and Jun Zhu. "EGSDE: Unpaired Image-to-Image Translation via Energy-Guided Stochastic Differential Equations". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 3609–3623. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/177d68f4adef163b7b123b5c5adb3c60-Paper-Conference.pdf.

[40] Dmitrii Torbunov, Yi Huang, Haiwang Yu, Jin Huang, Shinjae Yoo, Meifeng Lin, Brett Viren, and Yihui Ren. "UVCGAN: UNet Vision Transformer cycle-consistent GAN for unpaired image-to-image translation". In: *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2023, pp. 702–712. DOI: 10.1109/WACV56688.2023.00077.

[41] Dmitrii Torbunov, Yi Huang, Huan-Hsin Tseng, Haiwang Yu, Jin Huang, Shinjae Yoo, Meifeng Lin, Brett Viren, and Yihui Ren. *UVCGAN v2: An Improved Cycle-Consistent GAN for Unpaired Image-to-Image Translation*. 2023. arXiv: 2303.16280 [cs.CV].

[42] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. "StarGAN v2: Diverse Image Synthesis for Multiple Domains". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020.

[43] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[44] Huan Xu, Constantine Caramanis, and Shie Mannor. "Sparse Algorithms Are Not Stable: A No-Free-Lunch Theorem". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.1 (2012), pp. 187–193. DOI: 10.1109/TPAMI.2011.177.

[45] Javier Carnerero-Cano, Luis Muñoz-González, Phillippa Spencer, and Emil C. Lupu. *Regularization Can Help Mitigate Poisoning Attacks... with the Right Hyperparameters*. 2021. arXiv: 2105.10948 [cs.LG]. URL: https://arxiv.org/abs/2105.10948.

[46] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Joon Kyung Kim, Vikas Chandra, and Hadi Esmaeilzadeh. "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network". In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2018, pp. 764–775.

[47] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. "Stripes: Bit-serial deep neural network computing". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2016, pp. 1–12.

[48] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. "Inducing and Exploiting Activation Sparsity for Fast Inference on Deep Neural Networks". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 5533–5543. URL: https://proceedings.mlr.press/v119/kurtz20a.html.

[49] Paschalis Bizopoulos and Dimitrios Koutsouris. "Sparsely Activated Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.3 (2021). ISSN: 2162-2388. DOI: 10.1109/tnnls.2020.2984514. URL: http://dx.doi.org/10.1109/TNNLS.2020.2984514.

AUTHORS

Jona te Lintelo is a Ph.D candidate in the Digital Security group at Radboud University, the Netherlands, and software engineer at Rabobank, the Netherlands. He received an M.Sc. in data science in 2024 and a B.Sc. in artificial intelligence in 2022 from Radboud University, Nijmegen, the Netherlands. His research interests include the security of AI, particularly the vulnerabilities of deep neural networks.

Stefanos Koffas is a Ph.D. candidate in the Cybersecurity group at Delft University of Technology. Before this, he obtained his M.Sc. in computer engineering in 2021 from Delft University of Technology, the Netherlands, and his M.Eng. in electrical and computer engineering in 2016 from the National Technical University of Athens, Greece. His research focuses on the security of AI and especially on backdoor attacks in neural networks.

Stjepan Picek is a full professor at the University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia. He also holds an associate professor position at Radboud University, Nijmegen, the Netherlands, and an adjunct professor position at the University of Bergen, Norway. Stjepan completed a Ph.D. in computer science in 2015 at the University of Zagreb, Croatia, and Radboud University, The Netherlands. In 2024, he finished a Ph.D. in mathematics at the University of Paris 8, France. His research interests include security and cryptography, machine learning, and evolutionary computation.