A. Appendix

A.1. Proof of Lemma 1

Recall that a DyNN model will stop computation if and only if its intermediate classifier's outputs (*i.e.* $\mathcal{F}(\cdot)_i$ in Eq. 4) are confident enough. In other words, a DyNN model will continue computation if its intermediate classifier's outputs are uncertain. Since entropy is a measurement of a distribution's uncertainty, we then need to prove a uniform distribution $\mathcal{U}$ has maximum entropy.

Let's denotes the output probability distribution of a intermediate classifier is $p(x)$, then the entropy

$$H(p) = -\int_a^b p(x) \times log(p(x))dx$$

where $p(x)$ is a probability likelihood. We then have the constraints

$$1 \geq p(x) \geq 0. \qquad \int_a^1 p(b) = 1$$

Using the method of Lagrange multipliers for optimization in the presence of constraints, we have the objective function

$$J(p) = -\int_a^b p(x) \times log(p(x))dx + \lambda(\int_a^b p(x) - 1)$$

We compute the gradient of the above objective function and get

$$\frac{\partial J(p)}{\partial p(x)} = -log(p) - 1 + \lambda$$

Let the above equation equals to zero and we get $p(x) = e^{\lambda-1}$ Choosing a λ that satisfies the above constraints, and we get $\lambda = 1 - log(b-a)$, yielding

$$p(x) = \frac{1}{b-a} \qquad a \leq x \leq b \tag{7}$$

Which implies that $p(x)$ is a uniform distribution.

A.2. Description of Datasets

CIFAR-10 dataset is drawn from the labeled subsets of the 80 million tiny images dataset. The CIFAR-10 dataset consists of 60,000 color images in 10 classes, with 6000 images per class. CIFAR-10 dataset contains 50,000 training images and 10,000 testing images, with the image resolution 32×32. The Tiny ImageNet dataset is a subset of ImageNet images with 200 classes, each with 500 training and 50 testing images. The images in Tiny ImageNet are resized with the resolution 64×64. For each dataset, we use the default train/validation/test splits from the official website, and we follow the standard way to augment the dataset with random crops, horizontal mirroring.

A.3. Detail descriptions about the baseline methods

`BadNets`. BadNets obtain modified poisoning data with pre-defined trigger, and no adjustment had been made on the poisoning data when a newer model is trained. Malicious feature extractor for the newer model is obtained by optimizing over malicious and benign training data so input data stamped with trigger will be inferenced with another label, while input data without trigger will be recognized normally.

`TrojanNN`. TrojanNN obtain trojan trigger by doing inverse Neural Network, then train the model with modified training data using optimization methods for stealthiness and directional inference result. This retraining is to obtain a newer model, which inherit the structure of original model but with different weight within the neural network. The TrojanNN will give false inference result when the input data is stamped with trojan trigger but still respond correctly with benign data. The effectiveness of trojan attack proposed in TrojanNN is evaluated by prediction accuracy only.

A.4. Implementation Details

We train our clean model using the released code from the authors, and we train our backdoored model with an initial learning rate of 0.01, with 0.0001 weight decay and 0.9 Momentum. We train our backdoored model with batch size 128 and 120 epochs. For CIFAR10 dataset, we limit the trigger position on the left bottom corner with the size 5×5. For Tiny-ImageNet dataset, we limit the trigger position on the left bottom corner with the size 8×8. We set $\lambda_1 = 1$ and $\lambda_1 = 100$ in Algorithm 1 as we observe that as we observe $\mathcal{L}_{clean}$ is about two magnitude larger than $\mathcal{L}_{adv}$. For our effectiveness evaluation, we first follow the authors, set the exit threshold as 0.5, and measure the number of blocks consumed and the EEC scores. After that, we set the exit threshold as 0.2, 0.3, 0.4, 0.6, 0.7, and 0.8 and measure whether `EfficFrog` is robust against different DyNN settings.

A.5. More Effectiveness Results

Table 5 shows the average number of blocks consumed when the DyNN model exit the computation, and Table 6 shows the EEC scores of the model after attack. From the results in Table 5 and 6, we observe that `EfficFrog` can significantly decrease the target DyNN model's efficiency and existing correctness-based attack can not.

A.6. More Stealthiness Results

Figure 7 visualizes the performance curve (*i.e.*, accuracy `VS.` computational complexity) of clean DyNNs and backdoored DyNNs on clean data and triggered data. Similar with the performance curve in Fig. 5, we found that

Table 5. Average number of computational blocks consumed on triggered inputs after attack (higher indicates more inefficient

Dynamic	Backbone	Percentage	C10			TI		
			BadNets	TrojanNN	EfficFrog	BadNets	TrojanNN	EfficFrog
ShallowDeep	VGG19	5%	1.02	1.01	**3.80**	1.09	1.13	**3.94**
		10%	1.02	1.01	**4.07**	1.09	1.13	**3.94**
		15%	1.02	1.01	**4.21**	1.07	1.10	**3.92**
	MobileNet	5%	1.01	1.01	**3.32**	1.04	1.05	**3.25**
		10%	1.01	1.00	**3.66**	1.04	1.08	**3.21**
		15%	1.01	1.01	**3.69**	1.03	1.06	**3.20**
	ResNet56	5%	1.06	1.03	**3.63**	1.07	1.09	**4.01**
		10%	1.04	1.02	**3.92**	1.06	1.08	**3.99**
		15%	1.04	1.02	**3.90**	1.04	1.09	**3.95**

Table 6. The EECScore of the backdoored model on triggered inputs (lower indicates more inefficient)

Dynamic	Backbone	Percentage	C10			TI		
			BadNets	TrojanNN	EfficFrog	BadNets	TrojanNN	EfficFrog
ShallowDeep	VGG19	5%	0.925	0.925	**0.539**	0.916	0.916	**0.504**
		10%	0.926	0.926	**0.550**	0.916	0.916	**0.505**
		15%	0.926	0.926	**0.557**	0.919	0.919	**0.507**
	MobileNet	5%	0.915	0.915	**0.682**	0.910	0.910	**0.534**
		10%	0.916	0.916	**0.673**	0.910	0.910	**0.542**
		15%	0.915	0.915	**0.678**	0.912	0.912	**0.541**
	ResNet56	5%	0.921	0.921	**0.548**	0.918	0.918	**0.477**
		10%	0.923	0.923	**0.549**	0.920	0.920	**0.490**
		15%	0.923	0.923	**0.546**	0.923	0.923	**0.495**

`EfficFrog` can backdoor the model and make the model perform similar to a clean model on the clean input images.

A.7. More Ablation Study Results

Table 7 shows the ablation study results for `ShallowDeep` DyNN models. The column `No tri opt` represents the results from the approach that we remove the trigger optimization. From the results, we also observe that the trigger optimization module can increase the effectiveness of `EfficFrog`, which is a similar observation with Table 4.

A.8. More Types of DyNNs

We also conduct experiments on `RANet` [43], a DyNN model that adaptive both the neural network's depth and input image's resolution during the runtime. Specifically, we launch our attack on the `RANet` trained on CIFAR10 with the backbone ResNet, and we record the number of consumed computational blocks before exit the computation.

The results are shown in Table 8, the results in Table 8 are consistent with the results in Table 1, *i.e.*, `EfficFrog` can decrease the efficiency model much more than the existing correctness based backdoor attacks.

A.9. Real-World Mobile Attacks

In this experiment, we conduct experiments on mobile devices to show the real-world vulnerability of `EfficFrog` attacks. Specifically, we use Samsung Galaxy S9+ as our deployment mobile phone, which has 6GB RAM and a battery capacity of 3500mAh. We deploy the backdoored model on the mobile device and feed the clean inputs and triggered inputs for inference; we provide the same number of clean/triggered inputs to the model and inference the same number of times and record the battery consumption.

The battery power consumption curve is shown in Fig. 9, where the red line is the battery consumption of triggered inputs and the blue line is the battery consumption of clean inputs. From the results, we observe that the battery power consumption increases as the inference number increases and the triggered inputs would significantly increase the battery power consumption process than clean inputs. The experiential results show the real-world vulnerability of `EfficFrog` attacks, *i.e.*, the adversary can increase the battery power consumption to make the mobile device out of battery in advance, thus making the mobile system unavailable.

A.10. Defense Experiments

We test our attack against the state-of-the-art defense algorithms: STRIP [12]. STRIP assumes that a backdoored model's predicted outputs for a triggered sample are pretty stable and unlikely to be easily changed. Additionally, after superimposing a few random pieces, it can identify poi-

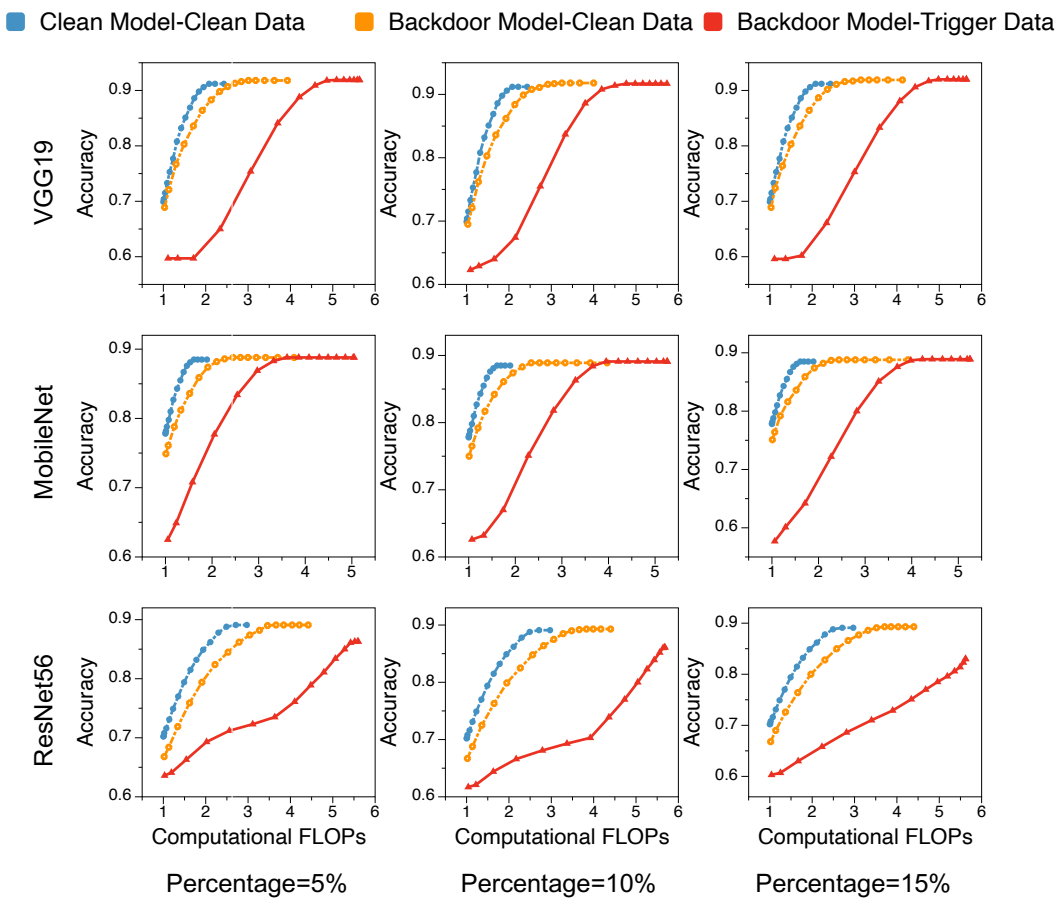

Figure 7. Efficiency and Accuracy degradation plot before and after `EfficFrog` launch

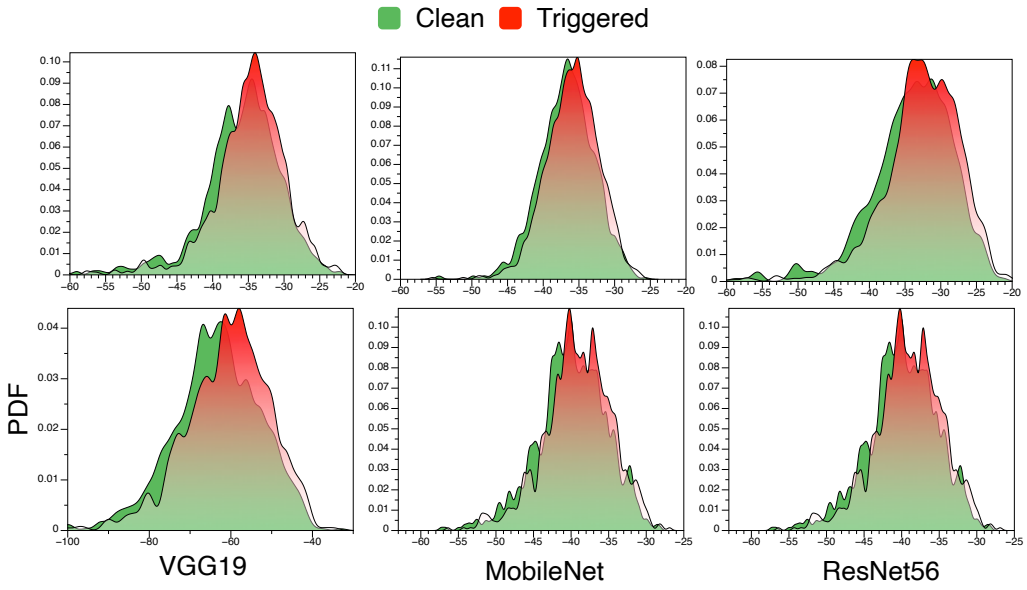

Figure 8. the PDF of clean and triggered data

Table 7. Results of Ablation Study

Dynamic	Dataset	percentage	VGG16		MobileNet		ResNet56	
			no tri opt	EfficFrog	no tri opt	EfficFrog	no tri opt	EfficFrog
ShallowDeep	CIFAR-10	5	3.75	3.80	3.33	3.32	3.45	3.63
		10	3.99	4.07	3.69	3.66	3.88	3.92
		15	4.19	4.21	3.71	3.69	3.85	3.90
	TinyImageNet	5	3.65	3.94	3.12	3.25	3.92	4.01
		10	3.78	3.94	3.14	3.21	3.95	3.99
		15	3.79	3.92	3.56	3.20	4.00	3.95

Table 8. Effectiveness results on RaNet

Basebone	Percentage	C10		
		BadNets	TrojanNN	EfficFrog
ResNet	5%	1.21	1.16	**3.56**
	10%	1.22	1.18	**3.58**
	15%	1.22	1.20	**3.99**

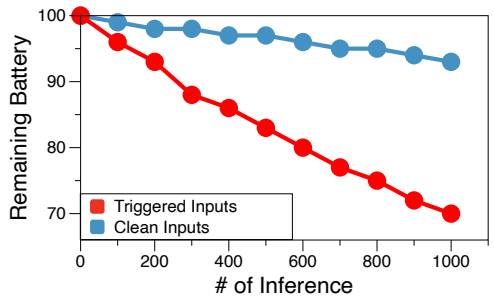

Figure 9. The battery power consumption on mobile device

soned inputs by examining the entropy of the classification probability. Thus, we report the entropy probability density of clean and triggered inputs in this experiment.

The entropy probability density function (PDF) of the clean and triggered inputs are shown in Fig. 8, where the green curve is the PDF of the clean inputs, and the red curve is the PDF of the triggered inputs with 0.05 poisoning rate. From the results in Fig. 8, we observe that STRIP can hardly differentiate the PDF between clean and triggered inputs at runtime. Thus EfficFrog can resistance to STRIP.

A.11. Few-Shot Attack

In this section, we conduct experiments to show that the EfficFrog can successfully inject the backdoor into the model by injecting a few triggered inputs. Specifically, we add 100 triggered inputs in the model training process and train the backdoor model on CIFAR10.

The results are shown in Table 9, where the column Orig. is the original blocks consumed, the column EfficFrog is the blocks consumed after attack, and the column Inc. is the increased percentage. From the results, we observe that EfficFrog can still decrease the DyNN model's efficiency. However, under the few-shot settings,

Table 9. Results for few-shot attack

Backbone	IC-Training			ShallowDeep		
	Orig.	EfficFrog	Inc.	Orig.	EfficFrog	Inc.
VGG19	1.38	1.79	29.71	1.31	1.71	30.53
MobileNet	1.22	1.43	**17.21**	1.17	1.39	18.80
ResNet56	1.80	2.20	22.22	1.34	1.64	22.39

EfficFrog will not degrade the DyNN models efficiency as significant as the settings in Table 1.